Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de prototipo de sensor IoT usando la red SigFox

Autor: Ramón Pérez Hernández
Tutor: Germán Madinabeitia Luque

Departamento deIngeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de prototipo de sensor IoT usando la red SigFox

Autor:
Ramón Pérez Hernández

Tutor:
Germán Madinabeitia Luque
Profesor colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Desarrollo de prototipo de sensor IoT usando la red SigFox

Autor: Ramón Pérez Hernández

Tutor: Germán Madinabeitia Luque

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A todos los que creyeron en mí;
aquí el resultado de mi esfuerzo

Agradecimientos

Parece mentira que lleve pensando en qué escribir en esta pequeña sección de *Agradecimientos* desde antes incluso de empezar este proyecto. Y, conforme avanzaba en su desarrollo y en la redacción de esta memoria, se iban fraguando estas palabras que quiero plasmar a continuación.

Agradecer, en primer lugar, a mis valedores en este proyecto: A Germán Madinabeitia Luque, por su labor como tutor, y por darme la motivación y la esperanza, con sus pequeñas dosis de humor, para concluir este proyecto con un resultado satisfactorio. Y también al equipo de Wellness Smart Cities, con Jaime Cafranga Bohórquez como cabeza visible, por la proporción de todo el material para el desarrollo completo del proyecto, y por todos los consejos, enseñanzas, ánimos y apoyos proporcionados en estos duros meses de trabajo.

También quiero acordarme de todos los profesores que me han ayudado y apoyado para lograr superarme día a día, desde los años de colegio hasta la universidad.

Este final tampoco hubiera sido posible sin la ayuda de todas las personas que han sido protagonistas de mi vida en estos últimos años. Desde el comienzo de este sueño hecho realidad, hasta los gritos de euforia por llegar al final, pasando por los eternos veranos de playa, sol y diversión. Nombraros a todos, con los despistes puntuales que tengo, es imposible, pero si has sonreído leyendo este pequeño párrafo recordando los buenos momentos que nos han dado estos años, estás incluido en la lista.

A mi familia. La que me ha soportado en los malos momentos; pero, sobre todo, con la que he disfrutado de todo lo bueno que ha venido a lo largo de mi pequeña, pero gran vida. Padres, hermana, tíos, primos, abuelos. Nada de esto hubiese sido posible de no ser por vosotros.

Por último; gracias a ti, María. Cuando pensaba que éste sería un año más, como otro cualquiera, apareciste para cambiarlo todo. Empezando por esas madrugadas eternas, donde sacábamos a relucir todas nuestras inquietudes e intereses. Continuando por esos momentos inolvidables que dibujan nuestra historia. Y, ahora, viviendo el fin de una etapa, pero preparándonos para todo lo nuevo que se avecina. Con unas gotas de ilusión, una pizca de la vitalidad que me proporcionas, y una gran porción de todo mi amor, este párrafo es totalmente tuyo.

Ramón Pérez Hernández

De San Fernando a Sevilla, 2015

Resumen

Este proyecto, en colaboración con Wellness Smart Cities, está centrado en el estudio de las posibilidades que puede ofrecer la red SigFox para la integración y comunicación de dispositivos a través del Internet de las Cosas (IoT).

Para ello, se hará uso de un modelo concreto de módem para realizar las pruebas; el TD1204 (de Telecom Design). Dicho dispositivo, usado como un sensor de la red SigFox, permitirá monitorizar diversas alarmas definidas en los casos de uso a probar; como pueden ser estado de batería, temperatura, movimiento o posición.

Esos casos de uso serán implementados en una aplicación final; que permitirá, además de la monitorización, el envío de datos al backend de SigFox, para poder visualizarlos en dicha página web o redirigirlos a un servidor propio para implementar gráficas y alarmas en tiempo real.

Por último, y ante la creciente aparición de soluciones basadas en la red SigFox, se pretende comparar su funcionamiento con respecto a otra tecnología ya usada para estos casos de monitorización de sensores, como es GPRS, para estudiar las ventajas y limitaciones de una futura migración a SigFox en cuanto a los casos de uso estudiados se refiere.

Abstract

This project, in collaboration with Wellness Smart Cities, is focused on the study of the possibilities that SigFox network can provide for devices integration and communication through the Internet of Things (IoT).

To do this, we will make use of a particular model of modem for testing; the TD1204 (Telecom Design). This device, used as a sensor of SigFox network, will allow monitoring various alarms defined in the use cases to test; such as battery status, temperature, movement or position.

These use cases will be implemented in a final application that, in addition to monitoring, will allow sending data to SigFox backend, in order to see them in that web page or redirect to your own server to implement graphs and alarms in real time.

Finally, and given the increasing solutions which are appearing based on the SigFox network, it is expected to compare its performance in relation to other technology already used in these cases of sensors monitoring, such as GPRS, in order to study the advantages and limitations of a future migration to SigFox in regards to the use cases studied.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

"Habrá dos tipos de negocios en el siglo XXI: aquellos que estén en Internet, y aquellos que ya no existan"

- Bill Gates, cofundador de Microsoft -

SigFox pretende convertirse en la primera red de Internet de las Cosas del mundo. Se ha lanzado en España a través de las amplias infraestructuras de Abertis Telecom, que conectarán el país con la red global de SigFox [1]. Esto será la clave para el desarrollo del ecosistema global en un amplio rango de sectores de Internet de las Cosas.

La cooperación entre ambas empresas permitirá a Abertis ofrecer una solución de conectividad de doble vía hecha a medida para Internet de las Cosas. La red SigFox está atrayendo el interés de numerosos sectores; incluidos energía, ciudades inteligentes, supervisión medioambiental, seguridad, atención sanitaria y transporte; ámbitos muy activos en España.

1.1. Motivación

Actualmente, en Wellness Smart Cities, los productos utilizados en el ámbito de ciudades inteligentes están centrados en la comunicación GPRS, pero surge la necesidad de tener un menor consumo de energía para los equipos que son alimentados por pilas.

Este proyecto nace con la idea de estar preparados para adaptarse a SigFox si el mercado lo demanda, evaluando todas las ventajas y desventajas que ofrecería, en comparación con GPRS, para estudiar así la viabilidad de una posible migración a la tecnología SigFox.

1.2. Objetivos

Este proyecto tiene, como objeto, el desarrollo de un prototipo de firmware para el módem TD1204, que permita monitorizar y notificar al backend de SigFox [2] diversas alarmas; como pueden ser estado de la batería, temperatura, movimiento o posición.

Se hará uso, para ello, del SDK disponible para la familia de módems TD12xx [3], que incluye un IDE basado en Eclipse para poder programar aplicaciones para la placa de evaluación. Se aprovecharán también las librerías y código ya desarrollados por Telecom Design, incluido en su repositorio de Github [4] (sólo accesible para desarrolladores).

Se pretende, también, conocer el proceso de comunicación del sensor con el backend, y las posibilidades que ofrece el mismo para la visualización de datos o la redirección de los mismos a un servidor web que permita gestionar dichos datos y mostrar estadísticas.

Como resultado, se verificará que existe un funcionamiento correcto en cada uno de los casos de uso definidos, comprobando que se reciben los mensajes correspondientes a cada una de las alarmas en el backend de SigFox.

Por último, se extraerán todos los datos necesarios para realizar la comparativa entre el funcionamiento de un módem SigFox y el de un módem GPRS. Aspectos como el consumo energético, el ancho de banda, el alcance y los costes de datos y materiales deben ser tenidos en cuenta en dicha comparativa.

1.3. Estructuración del proyecto

La estructura de este proyecto está enfocada en adquirir, finalmente, los conocimientos necesarios para programar aplicaciones para la familia de módems TD12xx; centrándonos en el modelo TD1204, en concreto.

La correcta interpretación de los resultados y su extrapolación para justificar las ventajas y desventajas del uso de esta tecnología jugarán también un papel importante en este proyecto.

Para ello; el desarrollo del proyecto, reflejado en esta memoria, tratará los siguientes aspectos, secuencialmente:

- En primer lugar, se pretende un acercamiento al lector al estado del arte y el mundo del Internet de las Cosas, con conceptos generales y aspectos claves del mismo. Se comentarán las distintas tecnologías utilizadas en este ámbito, pasando finalmente a mencionar la elección escogida para este proyecto, que será SigFox.
- Una vez elegida la tecnología, se tratarán conceptos más concretos de la misma; desde aspectos técnicos hasta los equipos que hacen uso de ella. Por último, se evaluarán dichos equipos y se extraerán aquellos que serán útiles para el proyecto: la familia TD12xx.
- Ya elegida la tecnología y el equipamiento, nos centraremos en detalles hardware y software del equipo escogido (el módem TD1204); desde el estudio de su arquitectura, hasta las posibilidades que nos ofrece para su implementación; contando con comandos Hayes (AT) para su configuración, o un SDK completo para el desarrollo de aplicaciones.
- Aprovechando ese SDK, nos centraremos en el desarrollo de una aplicación que implemente todos los casos de uso requeridos. Se utilizará Eclipse como IDE y C como lenguaje de programación. Se partirá de la especificación de los casos de uso, pasando por la solución elegida, y la implementación definitiva.
- Una vez desarrollado el código, se pasará a la prueba del mismo, comprobando que todas las alarmas provocadas durante la ejecución quedan reflejadas en el backend de SigFox, y que se pueden extraer estadísticas de la misma redirigiendo los mensajes enviados al backend a un servidor LAMP previamente especificado.
- Una vez el funcionamiento sea el deseado, se estudiarán los aspectos de consumo y rendimiento para realizar la comparativa final entre SigFox y GPRS, extrayendo las conclusiones pertinentes.

2 ESTADO DEL ARTE. IOT

"Si piensa que Internet cambió su vida, piense de nuevo. ¡El Internet de las Cosas está a punto de cambiarlo todo de nuevo!"

- Brendan O'Brien, Chief Architect y cofundador de Aria Systems -

La evolución de las TIC a partir de mediados del siglo XX ha provocado una verdadera revolución; no sólo en el mundo de las tecnologías, sino también en nuestras propias vidas. Si bien es cierto que cada uno de los avances surgidos ha tratado de satisfacer las necesidades presentes de la época en la que se originó, se ha dado una tendencia natural de mejorar lo ya implementado.

Esta mejora paulatina de las tecnologías no sólo está enfocada en la funcionalidad, sino también en la simplificación. Así, podemos ver que los grandes ordenadores primitivos que ocupaban salas enteras se han reducido a ordenadores portátiles y de sobremesa en la actualidad. Es uno de los muchos ejemplos que reflejan esa tendencia seguida: el afán de conseguir reducir al máximo las dimensiones físicas, el consumo o la complejidad del hardware y/o software del equipo, pero manteniendo o mejorando sus prestaciones y funcionalidad.

Internet también ha contribuido a esa revolución. Su drástica evolución en los últimos años ha provocado que ya no sea posible pensar en un mundo sin Internet. En la construcción de este nuevo mundo, el poder de la información tiene un papel primordial.

La búsqueda y obtención de información fue, de hecho, uno de los principales objetivos de Internet en sus primeros años. Pero ya no sólo pensamos en ordenadores o grandes máquinas que procesan datos; las fuentes de dicha información son, cada vez, más heterogéneas, y cualquier tipo de objeto cotidiano ya tiene cabida en Internet. Lo que antes eran personas conectadas por máquinas en una red se ha transformado en un complejo escenario, donde multitud de dispositivos (no sólo computadores) se interconectan entre sí para el traspaso de información útil.

En ese entorno es donde entra el concepto de **Internet de las Cosas** (IoT), que pasará a estudiarse con mayor profundidad a continuación.

2.1. IoT. Conceptos previos

El Internet de las Cosas se ha convertido en uno de los conceptos más populares y comentados en los últimos años. Si bien no cuenta con una definición formal, podría definirse como el concepto basado en *"la interconexión digital de cualquier tipo de objeto, tanto con cualquier otro de su alrededor (M2M) como con las personas, a través de Internet".*

Sus raíces se remontan al Instituto de Tecnología de Massachusetts (MIT), en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y las tecnologías de sensores emergentes [5]. Kevin Ashton, en aquellos años, fue el primero en mencionar el concepto de Internet

de las Cosas. Sus ideas [6] pueden resumirse en lo siguiente:

"Hoy en día, los ordenadores (y, por tanto, Internet) dependen casi por completo de los humanos para obtener información. Prácticamente la totalidad de los datos alojados en Internet han sido generados por humanos, pero tenemos atención, precisión y tiempo limitados; luego no podemos considerarnos como el medio ideal para obtener información del mundo real.

Si hubieran equipos que pudieran saber todo acerca de todas las cosas, obteniendo los datos sin ayuda de nadie, podríamos controlarlo todo; reduciendo gastos, pérdidas y costes".

Otro de los investigadores del MIT que ha profundizado en el concepto de Internet de las Cosas desde sus orígenes es Neil Gershenfeld. En su obra *Cuando las cosas empiecen a pensar*, de 1999, mencionó varias ideas interesantes con respecto al tema [7], como son:

"Además de intentar que los ordenadores estén en todas partes, deberíamos intentar que no estorbaran".

"Se debe hacer uso de la tecnología sin atender las necesidades de ésta".

"Los objetos deben poder tener identidad, acceder a otros objetos y detectar su entorno".

Es decir; hablamos de dispositivos con **identidad propia** (identificables de forma única en la red), capaces de **procesar información de manera independiente** (sin intervención humana), y cuya entrada en el mundo de Internet **no supondría una pérdida en el rendimiento** de la red global (no siendo un estorbo).

Lo que fueron simples ideas y comentarios surgidos de un grupo de investigación se ha convertido en una realidad en los últimos años. Ligado a esto, cada vez, más dispositivos aparecen en nuestra vida cotidiana. Surge, así, una definición formal del Internet de las Cosas, propuesta por el IBSG (de Cisco) [5], que lo definía como *"el punto en el tiempo en el que se conectaron a Internet más "cosas u objetos" que personas".*

De hecho, sitúan ese punto en torno a los años 2008 y 2009, coincidiendo con la explosión del concepto de Big Data (cronología interactiva disponible en [8]), cuyo afán es el procesado de millones de datos en tiempo real.

En la siguiente figura, podemos ver la estimación del número de objetos conectados al Internet de las Cosas hasta el 2020, comparando dichas cifras con el crecimiento de la población mundial en ese período.

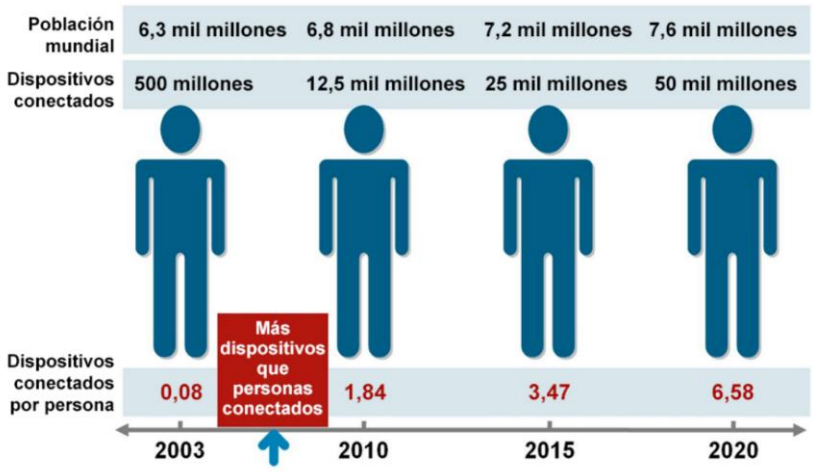


Figura 2-1. Estimación del número de objetos conectados en el mundo hasta 2020.*Cisco IBSG, abril de 2011.*

Vemos que, en los próximos 5 años, la cantidad estimada de objetos conectados se duplicará prácticamente. Todo esto sin contar que gran parte de la población mundial todavía no tiene acceso a Internet [9], y que las futuras innovaciones en el campo de las TIC podrían provocar un aumento en dicha cifra.

2.2. Campos de aplicación

Una vez clara la idea que gira en torno al mundo del IoT, ¿podemos sacarle partido? La novedad del concepto y la popularidad que está adquiriendo actualmente hace que se convierta en una gran **oportunidad de negocio** para las empresas, con un impacto económico y social brutal; tanto en la adaptación de los puestos de trabajo como en el ahorro de costes a largo plazo, por citar ejemplos [10]. Esto abre un mundo incalculable de posibilidades, incluso mayor que el abierto por la era digital en sus tiempos.

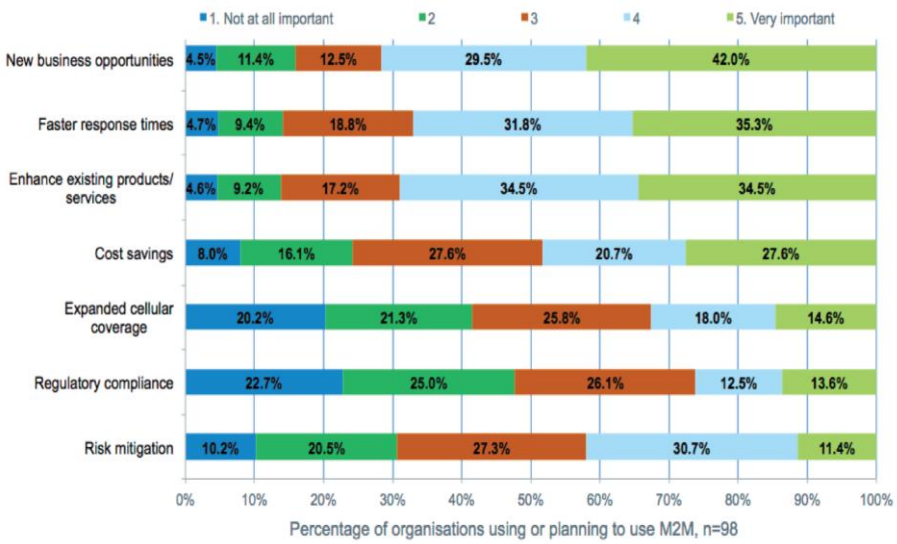


Figura 2-2. Principales razones de las empresas para invertir en IoT, destacando las nuevas oportunidades de negocio. *Encuesta interactiva del CBS.*

Entre esas **posibilidades**; podemos destacar los siguientes sectores de actividad [11]:

- Agricultura y medio ambiente.
- Atención sanitaria.
- Automovilismo.
- Construcción.
- Domótica.
- Electrónica de consumo.
- Fabricación y cadena de suministro.
- Servicios de emergencia y seguridad.
- Servicios públicos.
- Transporte y ciudades inteligentes.
- Venta minorista y ocio.

Entre todos los ejemplos anteriores, además de todas las posibilidades imaginables, se pretende destacar la aplicación en **ciudades inteligentes** [12], que da pie a la realización de este proyecto.

En este entorno, se pretende aprovechar las cualidades del IoT para medir parámetros externos (como temperatura, energía, luz, humedad, etc.) de forma automática y sin la interacción del ser humano. Una vez capturados los datos, estos se enviarían a un centro de procesamiento, para tomar las decisiones adecuadas en tiempo real en función de los resultados obtenidos.

Los casos aplicables para ciudades inteligentes son numerosos: control del estado de los semáforos en función del volumen de tráfico en cada instante o del número de peatones que frecuentan la calle, supervisión

del nivel de llenado de contenedores de basura para optimizar la recogida de residuos urbanos, o monitorización del número de aparcamientos libres en un parking, por citar algunos ejemplos de casos reales.

2.3. Tecnologías principales

Los conceptos introducidos en el estudio anterior de las ciudades inteligentes, suponen un buen punto de partida para dar varias pinceladas sobre las **tres capas**, o fenómenos, que posibilitan el uso del Internet de las Cosas [12] [13]. Cada una de ellas se ocupa de una tarea; desde la **extracción de los datos** captados, su **envío y recepción**, y su posterior **procesamiento** para obtener los resultados deseados.

2.3.1. Miniaturización de los componentes

Este concepto se comentó en la introducción del capítulo; y es que resulta fundamental apostar por dispositivos cada vez más pequeños, ocupando menos espacio, pero manteniendo su funcionalidad e incluso mejorándola.

2.3.1.1. Los nuevos procesadores

El nuevo enfoque provoca, como resultado, la adaptación de los **procesadores** actuales a este nuevo tipo de dispositivos. Los dos requisitos fundamentales son claros: **más pequeños** y con **menor consumo**. Esto garantizaría la conectividad de los equipos prácticamente sin limitaciones de tiempo y espacio.

Los procesadores con arquitectura ARM [14], por ejemplo, cumplen con estas expectativas, a pesar de ser poco potentes en comparación con otras soluciones. De hecho, el dispositivo con el que se han realizado las pruebas para este proyecto cuenta con esta arquitectura en su procesador.

2.3.1.2. Internet 0 como capa física

Se hablaba de conectividad entre equipos, pero no de la velocidad de transmisión de datos. En el caso del IoT, no se requerirá un gran ancho de banda, sino todo lo contrario. Con ello, nace el concepto de **Internet 0**, o *"sistema que permite conectar objetos a Internet a baja velocidad, pero de una manera mucho más barata y compatible con todo tipo de sistemas"*. Con esto, podemos incluir a esta nueva red de redes una heterogénea multitud de objetos.

Neil Gershenfeld también tenía palabras para el Internet 0 [7]; citando:*"el nombre de Internet 0 procede del empleo de una comunicación lenta para hacer más fácil su implementación".*

2.3.1.3. Sensores para la recolección de datos

Haciendo un recorrido inverso; se ha hablado del mecanismo interno de procesamiento de datos del dispositivo, y de la red física que permitirá la conectividad de los mismos. Pero falta hablar del origen; la obtención de la información. Aquí entran en juego los **sensores** y **redes de sensores**. Según la RAE, se define sensor como el *"dispositivo que detecta una determinada acción externa, temperatura, presión, etc., y la transmite adecuadamente"*. Aquí aparecen los conceptos, ya comentados, de obtención de información de manera independiente, y su posterior transmisión.

Las posibilidades en cuanto a tipología de sensores son inmensas: sensores de ultrasonidos, de luz, de distancia, táctiles, de temperatura, de humedad, de altitud, de presión, acelerómetros, y un largo etcétera. Y contando que las compañías involucradas en el IoT también tienen la posibilidad de diseñar y fabricar sus propios sensores personalizados para satisfacer sus necesidades, las cifras se disparan por completo.

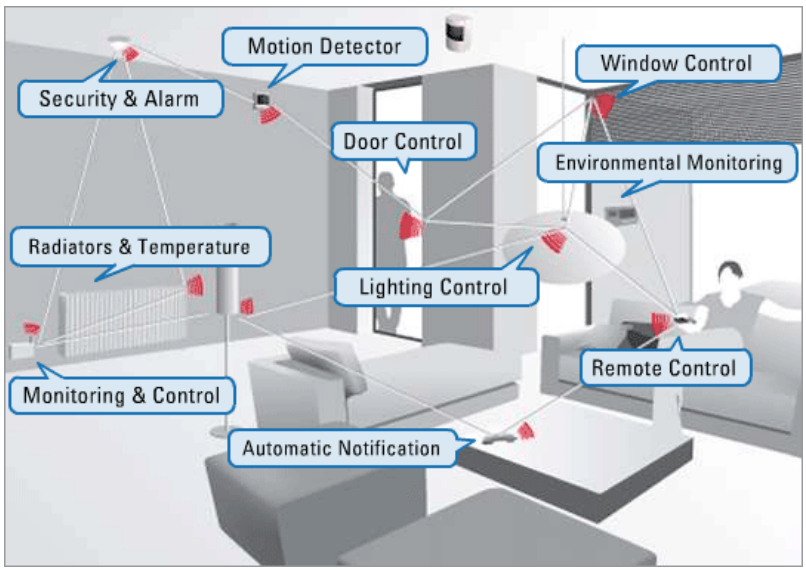


Figura 2-3. Ejemplo de red de sensores desplegada en una vivienda.

2.3.2. Infraestructuras de telecomunicación

Una vez los sensores capturan los datos, conviene transmitirlos a otro dispositivo de mayor potencia (desde el punto de vista hardware), para poder procesarlos de forma más rápida. Para ello, se necesita un **canal de comunicación** que soporte la transmisión entre entidades, utilizando **protocolos de comunicación**; tanto ya existentes (redes de área local con Ethernet, o transmisión inalámbrica vía conectividad móvil), como nuevos y orientados al IoT (como NFC o Bluetooth 4.0).

¿Qué papel juegan los **operadores de telecomunicaciones** en este nuevo escenario? La respuesta es la adaptación, ya que el entorno que plantea el IoT es distinto al de las redes actuales [15].

Partiendo de la base que se debe asegurar, como en cualquier red, la conectividad de dispositivos, se debe procurar que el diseño y comercialización de los nuevos tipos de redes se oriente a las necesidades de objetos inteligentes. ¿En qué difieren estos objetos, o sensores, de los dispositivos utilizados en las redes actuales?

En las redes actuales, las transmisiones se orientan al "Internet de las Personas": llevan grandes volúmenes de información cada poco tiempo, con tasas de transmisión cada vez mayores. Sin embargo; la naturaleza de los sensores es opuesta: permanecen inactivos la mayor parte del tiempo, y envían datos de poco tamaño esporádicamente (cuando es necesario), con menor uso de ancho de banda y bajo consumo.

Una de las alternativas que tiene en cuenta estos principios para atender las necesidades de los sensores nos la presenta **SigFox** [16]. Proporciona, para ello, una infraestructura de antenas y estaciones base totalmente independientes de las redes de telecomunicación existentes, proporcionando una red exclusivamente de baja velocidad para la transmisión de datos de pocos bytes de tamaño.

2.3.3. Aplicaciones y servicios de extracción de datos

Llegamos al punto final donde desemboca toda la información transmitida. Estos **centros de recolección y procesamiento de datos** permiten obtener resultados a partir de las medidas de los sensores, dándole uso a toda la información generada en el IoT.

Aquí cobra importancia el desarrollo de algoritmos y software que permitan interpretar, correctamente, la cantidad ingente de datos que se transmite. De esta forma, toda la red se vuelve manejable y operable, cobrando sentido y significado toda la información vertida por los sensores.

2.4. Tecnologías elegidas para el proyecto

Una vez comentados los tres grupos principales de tecnologías existentes, y tras estudiar las posibilidades que ofrece cada opción posible de cada grupo, se citan las tecnologías que se utilizarán para el desarrollo de este proyecto, diferenciando entre:

- **SigFox** para la comunicación, al proporcionar una infraestructura de telecomunicación ya construida e independiente de cualquier red existente, con un bajo ancho de banda, y mejor adaptado a la transmisión de mensajes pequeños frente a otras alternativas, como LoRa [17].
- El uso del módem **TD1204**, de Telecom Design [18], como dispositivo. Certificado por SigFox, se presenta como una placa de evaluación de pequeño tamaño, con procesador ARM de bajo consumo, e incorporando una serie de sensores (destacando temperatura, batería, acelerómetro o GPS) para la obtención de datos de monitorización.
- El uso del **backend de SigFox** [2] como punto final de la comunicación, que recibirá los datos enviados por el módem y los presentará a través de su página web. Estos mensajes se reenviarán, mediante callback, a un **servidor LAMP** propio, para la generación de estadísticas y alarmas en tiempo real.

Notar que se ha cambiado el orden del estudio realizado con anterioridad: primero, se presentará el servicio que proporciona la comunicación, para luego hablar sobre dispositivos que hacen uso de dicho servicio. Por último, y concluyendo con el orden antes citado, se hablará de la parte de procesamiento de datos y representación de resultados.

3 SIGFOX

"SigFox ha aceptado el desafío de ser la red del Internet de las Cosas"

- Ludovic Le Moan, fundador y director general de SigFox -

Centrándonos en aspectos de conectividad de dispositivos dentro del Internet de las Cosas, resulta indispensable acercar al lector algunos conocimientos básicos sobre **SigFox**, la tecnología utilizada en este proyecto para ese cometido.

Figura 3-1. Logo de SigFox.

Si bien el objeto de esta memoria no es el de hacer un estudio exhaustivo de la red SigFox y su trasfondo, se deja como referencia la base sobre la que se sustenta este apartado: tanto su página oficial [16],como un documento oficial formato Whitepaper [19], que incluyen información extendida sobre todo lo aquí expuesto, para aquellos interesados en la materia.

3.1. Descripción de la red

SigFox se autodefine como *"la primera y única sociedad que ofrece una conectividad móvil mundial para el Internet de las Cosas, totalmente dedicada a las comunicaciones de baja velocidad. Se reinventa así la conectividad, disminuyendo radicalmente los precios y el consumo de energía para los dispositivos conectados, y proporcionando una red altamente escalable".*

Desgranemos, una a una, cada una de estas ideas, junto a otros conceptos de interés, para conocer con mayor profundidad el funcionamiento de esta tecnología.

3.1.1. Conectividad

Aunque SigFox se venda como una solución de cobertura mundial, la realidad es que todavía sigue en **estado de implementación**. La cobertura de la red ya cubre prácticamente la totalidad de países europeos como Francia (donde nació), Países Bajos, Portugal, Reino Unido y España, y pretende estar presente en un total de 60 países durante los próximos 5 años. En definitiva; se asegura cobertura prácticamente total, pero solamente en los países antes mencionados, restringiéndose al **continente europeo** en la actualidad.

Los encargados de implementar la red y ofrecer la cobertura y los servicios de SigFox son los llamados **SNO**; más conocidos como los **operadores de la red SigFox** (Abertis, en el caso de España, bajo el nombre de Cellnex Telecom [20]).

Esto abre una oportunidad de negocio para aquellos aspirantes a desarrollar el ecosistema del Internet de las Cosas en su región, siempre que demuestren capacidades de poder financiar, construir y mantener la red. A cambio, SigFox ofrece su soporte para el despliegue definitivo de la red. La siguiente figura refleja esta relación, con las tareas encomendadas a cada parte.

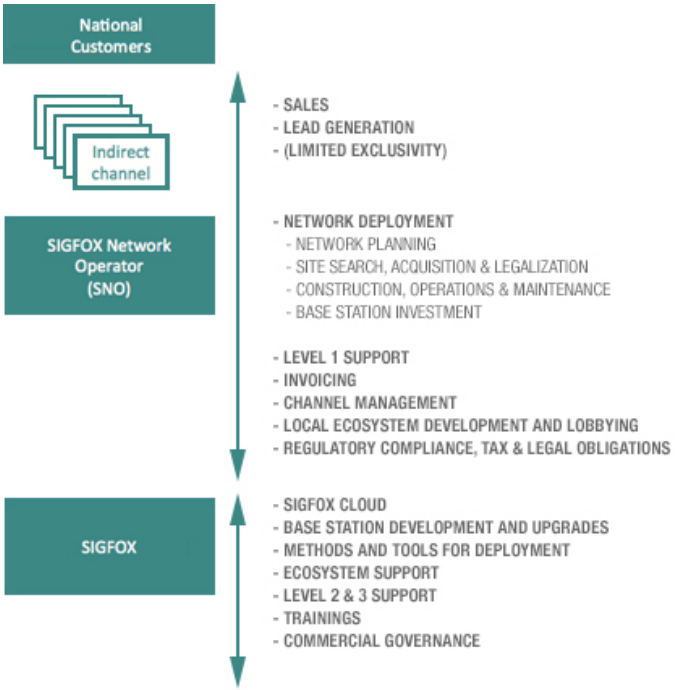


Figura 3-2. Roles del SNO y de SigFox. *Página oficial de SigFox.*

3.1.2. Velocidad de transmisión

Recordamos que, mientras los operadores de telecomunicación centran sus esfuerzos en construir redes cada vez más avanzadas para proporcionar anchos de banda mayores (como demuestra la evolución de las redes de telefonía móvil), las aplicaciones para el IoT se alejan de este concepto.

La razón es simple: los dispositivos están en **reposo** la mayor parte del tiempo, y transmitiendo **mensajes de pequeño tamaño**. Un gran ancho de banda tiene sentido en el entorno opuesto; datos de gran tamaño transmitidos cada poco tiempo. Esto supone un aumento en el consumo de energía; inaceptable en las soluciones IoT, ya que se alejan de su ideario de bajo consumo y ahorro en costes. Por ello, SigFox orienta su red a **transmisiones de baja velocidad** (con rangos de 10 bps a 1 kbps), aportando una solución que pretende cumplir esos principios.

3.1.3. Tecnología radio

Si nos fijamos en **aspectos radio**, veremos que la tecnología usada concuerda con los requerimientos de baja velocidad que necesitan las redes en el IoT.

En concreto, SigFox se presenta con una tecnología **UNB**, o de**banda ultra estrecha**, para conectar los dispositivos a su red. Se caracteriza por usar un ancho de banda muy reducido, ocupando poco espacio en el rango de frecuencias y empleando tasas de baja velocidad. Los módem telefónicos son ejemplos de dispositivos que trabajan en banda estrecha.

En cuanto a las **bandas de frecuencia** se refiere, la red SigFox funciona en las bandas **ISM**, o **bandas de frecuencia sin licencia**, coexistiendo con otras tecnologías radio en dichas frecuencias, pero sin riesgo de

colisión o problemas de capacidad. Más concretamente, trabaja sobre **868 MHz** en **Europa** (definida por la ETSI y CEPT) y sobre 915 MHz en Estados Unidos (definida por la FCC).

Se estima que cada estación base es capaz de dar servicio a un millón de objetos conectados, pero la red es **escalable**, de manera que puede alojar más objetos con tal de aumentar la densidad de las estaciones base.

Por último, la densidad de las celdas está en un rango de 30-50 kilómetros en áreas rurales, y de 3-10 kilómetros en áreas urbanas, lo que garantiza **alcanzabilidad** hasta esas distancias por cada estación base.

3.1.4. Protocolo de comunicaciones

SigFox se presenta con un **protocolo de comunicaciones propietario**, compatible con los transceptores existentes, y dirigido hacia un número de plataformas cada vez mayor.

El protocolo permite el envío tanto unidireccional como bidireccional de mensajes de **hasta 12 bytes de carga útil**, cuyo formato viene condicionado por los fabricantes de los dispositivos, para codificar la información de la manera más conveniente y con total libertad. A la carga útil se le suman campos como el ID del dispositivo.

El número de mensajes que se pueden enviar está limitado, basado en **suscripciones**: en función de la suscripción contratada (proceso explicado más detalladamente en el **Anexo A**), se permite un número máximo de mensajes al día (entre 0 y 140 mensajes). Esto tiene relación con la naturaleza de los dispositivos en el IoT; mensajes de pequeño tamaño, enviados con grandes intervalos de tiempo de separación.

Aunque se aplican medidas de **seguridad**; como la protección anti-respuesta, mensajes de aleatorización o secuenciación, los mensajes viajan vía radio de un punto a otro. Esto implica que los mensajes puedan ser interceptados, y de averiguarse la codificación que lleva, descifrados. Por lo tanto, las probabilidades de sufrir ataques contra la seguridad (como un *man in the middle*) se incrementan.

La simplicidad de este protocolo permite un fácil entendimiento y rápida aplicación, pero la falta de documentación pública al respecto impide un conocimiento más profundo del mismo.

3.1.5. Potencia de transmisión y consumo

Ambos términos van ligados, ya que la elección de la potencia de transmisión de las señales permitirá regular el consumo final de los dispositivos.

La **potencia de emisión** se caracteriza por trabajar con valores bajos; entre los -20 y 20 dBm, y siempre cumpliendo que la potencia total emitida por la antena no supere los 25 mW, de acuerdo a la legislación relativa a las bandas ISM.

Si entramos en aspectos de consumo, se puede comprobar un **consumo bajo de energía**, gracias a que la red de SigFox sólo se utiliza cuando el objeto necesita transmitir carga útil. Las cifras en las que se mueven los dispositivos son de entre 45 y 55 mA en la transmisión, y valores que bajan a los µA cuando está en reposo. Estos valores garantizan resultados muy eficientes energéticamente hablando, asegurando **la extensión de la autonomía** de los equipos **en años**.

3.2. Conclusiones de la solución que ofrece SigFox

Tras este estudio, punto por punto, de lo que puede ofrecernos SigFox en diversos aspectos claves, acabamos resumiendo los **requisitos** que cumple para dar el servicio de telecomunicación que demanda el Internet de las Cosas. A decir:

- **Facilidad de uso**, ya que la red ya está desplegada y lista para usar. Basta con integrar los dispositivos en la red, asegurando su conectividad, para comenzar a utilizar la red.
- **Operatividad**, facilitando servicios para la monitorización y gestión de dispositivos, ya que todos los objetos están conectados directamente a la red, y son fácilmente controlables desde el punto final de la comunicación.
- **Largo alcance**, evitando el tener que implementar infraestructuras locales complejas para alcanzar

todos los objetos, y asegurando la conectividad entre objetos separados por una gran distancia.

- **Independencia de la frecuencia**, garantizando una cobertura mundial y adaptable a cualquier entorno.
- **Bajo consumo de energía**, para incrementar la vida útil de la batería, reducir el mantenimiento y minimizar el impacto climático.
- **Bajo coste**, permitiendo conectar a la red cualquier tipo de objeto en grandes cantidades.

3.2.1. Aplicación en ciudades inteligentes

Si nos centramos en las **ciudades inteligentes** como ámbito de aplicación del Internet de las Cosas y de la tecnología M2M, comprobamos que se pueden aprovechar todas las **ventajas** de SigFox ya comentadas, de manera que:

- No haría falta desplegar una infraestructura de red local, puesto que ya estaría construida.
- Se dispondría de una vasta área de cobertura.
- Se aseguraría la conectividad de los dispositivos, tanto interior como exterior.
- Dichos dispositivos tendrían un consumo de energía muy bajo, permitiendo su autonomía durante un período prolongado.
- Sería rentable y aplicable en sensores de bajo coste.
- Tendría una fácil integración en sistemas informáticos.
- Daría soporte a un servicio de alta disponibilidad.

3.3. Recepción y procesamiento de datos

Tras hablar de la red, quedan pendientes por comentar los dos extremos de la comunicación: los dispositivos, emisores de mensajes, que están conectados a la red, y el punto final o **backend** de la comunicación, que recibe esos mensajes y los procesa para generar un resultado.

En el caso de SigFox, se ofrece el servicio llamado **SigFox Cloud** para el segundo propósito, que ofrece una aplicación web conocida como **SigFox Backend** [2]. Desde ella, se pueden gestionar los dispositivos, visualizar los mensajes transmitidos por los mismos y configurar de integración de los datos, entre otros.

Además, el servicio da la oportunidad de poder **redirigir** todo el volumen de información que llega al backend **a cualquier aplicación** ejecutada en un servidor o centro de procesamiento de datos.

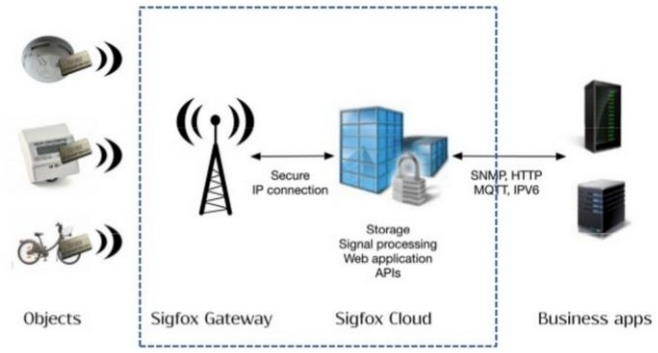


Figura 3-3. Topología de una red basada en SigFox, con SigFox Cloud como receptor de datos y las aplicaciones como procesadoras de éstos. *Página oficial de SigFox.*

Esa aplicación puede recibir los datos de dos maneras:

- Utilizando la **API** que proporciona el backend, basada en **HTTP REST** (GET o POST, indistintamente); la cual, en función del recurso pedido, devuelve un resultado concreto, con una carga útil con formato **JSON**.
- Utilizando una **URL de callback**, identificando dicha URL a la aplicación web que desea recibir los mensajes. De esta forma, se registraría dicha URL en el backend, indicando los atributos que le interese recibir (por ejemplo, la carga útil del mensaje); y cada vez que llegase un mensaje al mismo, éste le **reenviaría** los valores pedidos en un mensaje con formato **JSON**.

Concretamente, la segunda funcionalidad será la que aprovecharemos para recibir la carga útil de los mensajes que llegan al backend, representando las estadísticas de uso del dispositivo en tiempo real en nuestra aplicación web.

3.3.1. Backend de SigFox

Centrándonos en el **backend de SigFox**, presentamos las opciones de navegación que nos ofrece, con una breve explicación de cada una de ellas:

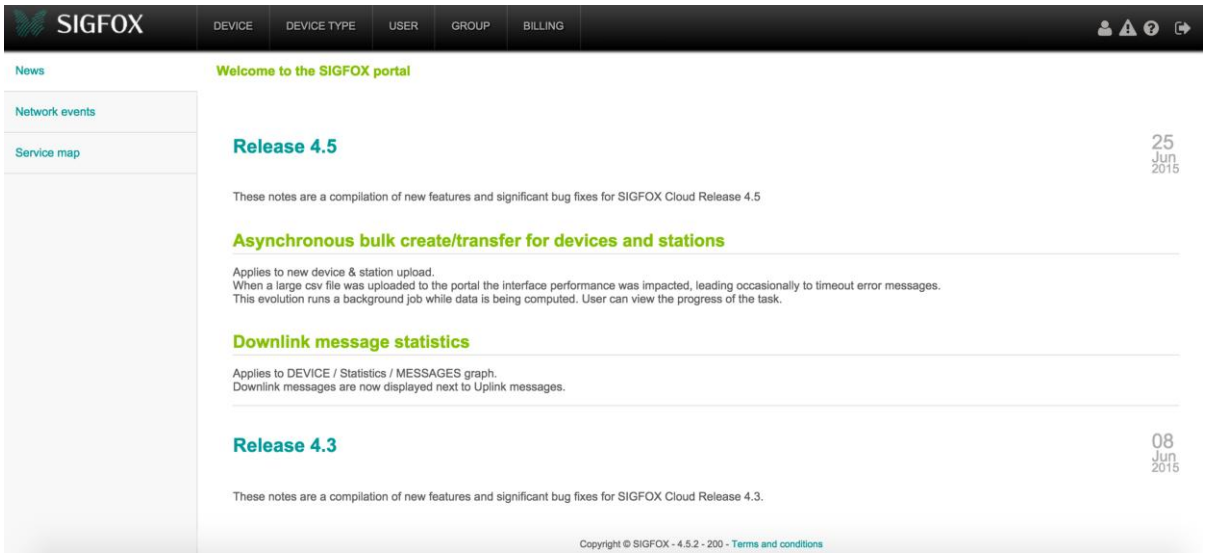


Figura 3-4. Backend de SigFox.Página principal.

Cuando accedemos al portal, se nos presenta una **página de bienvenida**, que nos notifica de las nuevas funcionalidades incluidas en la página. También tenemos acceso a una lista de **eventos de la red** y a un **mapa con la cobertura** actual en el país.

A través de la barra superior, podemos navegar por los distintos apartados de la página, diferenciando los siguientes (de izquierda a derecha):

- **Device**: nos muestra los **dispositivos** registrados en el backend, distinguidos por un identificador único. Entre otras opciones; nos muestra estadísticas con el número de mensajes enviados diariamente, notificaciones de eventos surgidos durante la transmisión (como saltos en el número de secuencia, que indican pérdida de información), y sobre todo, los **mensajes enviados**; con la fecha de recepción, el contenido del mensaje (con la codificación elegida por el fabricante del dispositivo), su traducción a ASCII (si se han enviado caracteres), su localización (mostrando un rectángulo formado por la latitud/longitud, sin decimales, en la que se encuentra el dispositivo), información sobre redundancia, el nivel de la señal recibida (en dB), y la URL de callback a la que se redirige (de haberla).

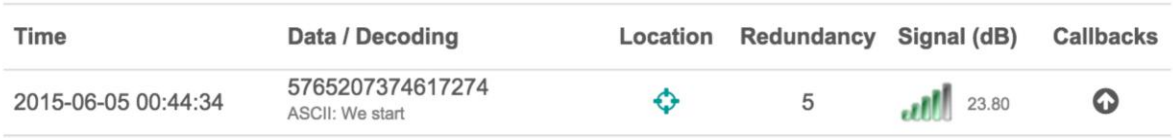


Figura 3-5. Ejemplo de mensaje recibido al backend, con traducción a ASCII disponible.

- **Device Type**: lista los **tipos de dispositivos** registrados en el backend. De esta forma, a cada conjunto de dispositivos le podemos asociar un tipo para gestionarlos de la misma manera. La opción más relevante a considerar en este apartado es el establecimiento de las **URL de callback** a cada tipo de dispositivo; pudiendo utilizar más de una URL para cada tipo, eligiendo entre GET o POST, y pudiendo seleccionar las variables que se desean obtener (entre otras; el identificador de dispositivo, la hora de llegada del mensaje, la potencia media de la señal, la latitud/longitud desde donde se envió el mensaje, o la carga útil).
- **User**: muestra los **usuarios**, pertenecientes a un grupo, que tienen acceso al backend.
- **Group**: gestiona los **grupos** configurados en el backend. A ellos se le pueden asociar usuarios, dispositivos o suscripciones. Además, SigFox le proporciona un usuario y contraseña para tener acceso a la **API REST**.
- **Billing**: se encarga de las **suscripciones** a SigFox, incluyendo los servicios contratados, el número de mensajes máximo permitido o el precio de la suscripción, como aspectos más relevantes.
- **Información rápida del usuario**: hace de resumen de la pestaña User, e incluye las **direcciones IP** con las que el usuario ha accedido al backend, junto la fecha de último acceso de cada una.
- Redirección a la **lista de eventos de red.**
- **Ayuda online**: dispone de **documentación** para el uso de callbacks y la API REST, información para el proceso de suscripción, y una breve mención al formato de los mensajes enviados.
- **Logout**: para cerrar sesión.

En próximos apartados, se entrará en más detalle en aquellas que sean de utilidad para el desarrollo del trabajo.

3.4. Dispositivos disponibles

Faltan por comentar los actores principales en el escenario de la red: los dispositivos, emisores de información. No nos centraremos en aspectos hardware o de necesidades de estos equipos, sino en las posibilidades que ofrece SigFox a los fabricantes para **integrar** sus dispositivos en la red.

La realidad es que SigFox se encuentra disponible a través de los **principales proveedores de chips y módulos del mercado** (entre otros; Silicon Labs, Texas Instrument, Intel, Telecom Design, o ETSI), ofreciéndoles soporte y facilidades para la integración de sus equipos en la red. SigFox, que busca la normalización de sus soluciones para la comunicación en el IoT, permite así la **interoperabilidad** entre equipos de distintos fabricantes.

Además de este soporte, SigFox ofrece la posibilidad de **certificar** los dispositivos con la marca **SigFox Ready**. Este proceso pretende clasificar los dispositivos en función de la cobertura y el alcance radio al que pueden tener acceso, con categorías de 0 a 3; siendo la 0 la que mejor calidad radio ofrece, y la 3 la que da una calidad más baja.

Figura 3-6. Etiqueta SigFox Ready presente en equipos certificados.

Por último; una vez fabricados y certificados los dispositivos, quedaría **desarrollar aplicaciones** para ellos, de manera que se podrían reemplazar soluciones existentes porque el uso de SigFox fuese más conveniente en diversos campos de estudio, o bien se podrían desarrollar aplicaciones completamente nuevas e innovadoras para su introducción en el mercado.Son muchas las organizaciones dedicadas al desarrollo de soluciones para SigFox, pero se destaca una sobre el resto:**Telecom Design**. No sólo por formar parte, también, de la lista de proveedores de dispositivos SigFox Ready, sino por ser el creador de la tecnología **Cloud-on-Chip**; considerada la primera solución utilizada en la red SigFox.

3.4.1. Telecom Design, Cloud-on-Chip y la familia de módems TD12xx

Como se comentaba en el apartado anterior, **Telecom Design** [21] se presentó en el mundo SigFox con su solución conocida como **Cloud-on-Chip** [22], como resultado de 15 años de investigación y desarrollo de soluciones IoT y M2M. El afán de esta tecnología es la de *"proporcionar inteligencia local o en la nube a cualquier objeto; de manera que, combinado con la red SigFox, permita conectar a Internet cualquier tipo de equipo alimentado por batería".*

Para ello, Telecom Design proporciona una solución para programadores de aplicaciones [23] basada en la plataforma **Sensor**. A través de ella, ofrece a los desarrolladores una API y un panel de control para gestionar los dispositivos involucrados en la comunicación. Esta utilidad, si bien interesante en contenido, no se ha considerado para el desarrollo del proyecto.

Los dispositivos SigFox Ready que ofrece Telecom Design para su integración en la red SigFox son los que forman la **familia TD12xx** [18]; módulos de comunicación presentados en placas de evaluación de tamaño reducido y bajo consumo, aptos para su introducción en el mundo del Internet de las Cosas, y con funcionalidad de módems para la transmisión de señales.

Con un prefijo común, cada modelo se identifica con un nombre único, y se diferencia del resto por las capacidades que ofrece. En la siguiente figura podemos ver esa clasificación.

Figura 3-7. Clasificación de los módem TD12xx en función de sus capacidades. *Página oficial de TD Next Modules.*

3.4.2. Elección de dispositivo: el módem TD1204

Siendo todos dispositivos certificados por SigFox, interesaba el estudio de distintos casos de uso para la aplicación de estos dispositivos en entornos reales. Si bien el **SDK de Telecom Design** [3], unido a las **librerías de comunicaciones** de su repositorio [4], nos proporcionan la base para realizar aplicaciones que tengan en cuenta aspectos básicos del microcontrolador como temperatura o batería, hay que recurrir a modelos más complejos para encontrar más casos de estudio.

Es el caso de la geolocalización. Disponer de un acelerómetro integrado y un GPS (externo, en nuestro caso) abre posibilidades más interesantes; como la detección de movimiento para ver si el dispositivo se ha caído o ha sido robado, y su combinación con la geolocalización para saber la ubicación del sensor.

Ya que el uso de una antena integrada supondría un aumento en el coste del dispositivo, se optó por escoger el **módem TD1204** como dispositivo con el que realizar la codificación y las pruebas de los casos de uso que se comentarán más adelante con más detalle. Este modelo, que incluye todas las características antes mencionadas (salvo la antena integrada, irrelevante en nuestro caso de estudio), será objeto de estudio en el próximo apartado y considerado para el transcurso restante del proyecto, en cuanto a la codificación y pruebas se refiere.

4 EL MÓDEM TD1204

"TD1204, el módulo de gama alta para geolocalización"

- Telecom Design, describiendo el TD1204 -

Presentamos, en esta sección, el equipo elegido para realizar la codificación de la aplicación requerida para el proyecto. El módem **TD1204**, de **Telecom Design**, se autodefine como *"el mejor compromiso entre facilidad de integración y costes, siendo el primer módulo de SigFox con sensores de **GPS y acelerómetro de 3 ejes** incorporados para la geolocalización y seguimiento del dispositivo"* [24].

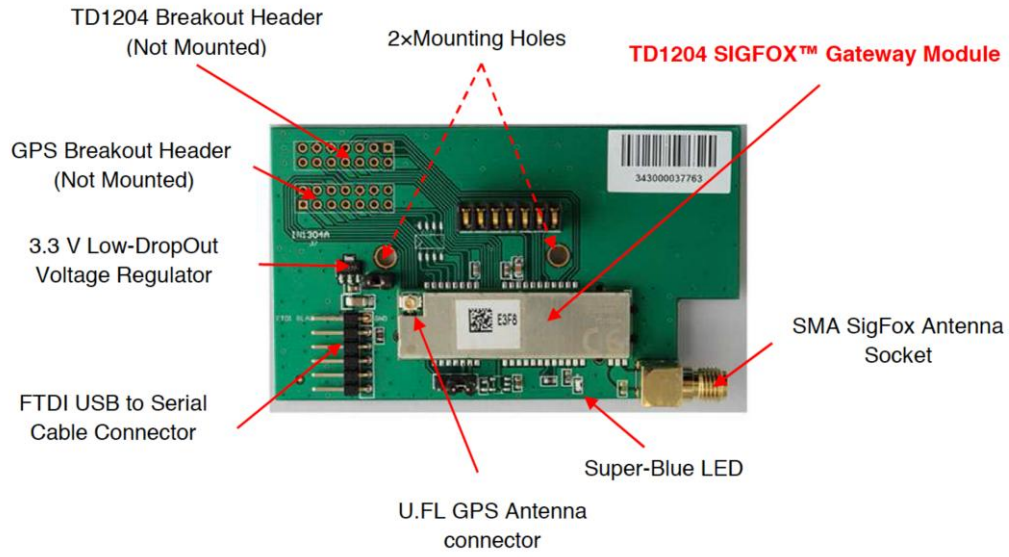


Figura 4-1. La placa de evaluación, con sus componentes principales. *Guía de uso del TD1204.*

4.1. Documentación y utilidades disponibles

En la sección del TD1204 de la página oficial de los módulos de Telecom Design [24], podemos encontrar documentación de utilidad sobre la placa de evaluación. También se cuenta con una sección en la página de desarrolladores de Telecom Design [25], pero toda la documentación se ha trasladado a la primera referencia antes indicada.

Los recursos que podemos encontrar en estas referencias (con las versiones utilizadas para este proyecto) son los siguientes:

- **Datasheet** (versión 1.3) del microcontrolador.
- Documento tipo **datashort**, con una breve descripción del producto.
- **Guía de uso** (versión 1.1) de la placa, con información para la puesta en marcha del dispositivo, la introducción a sus parámetros de configuración, y descripción física y del hardware de la placa.
- **Manual de referencia** (versión 1.4), para la configuración del módem a través de una colección de comandos Hayes (AT).
- **Firmware** actualizado de la placa (imagen 1450), en un archivo .bin, para poder hacer uso de dichos comandos Hayes.
- La aplicación **TD Loader** (versión 1.06, en un .exe), para poder cargar archivos .bin en la placa; ya sea el firmware antes comentado, o cualquier aplicación desarrollada. Añadir que la placa de evaluación sólo puede almacenar el contenido de un archivo .bin; luego la carga de un nuevo archivo sobrescribiría al que ya esté en la memoria del dispositivo.

Notar que, también, se cuenta con recursos disponibles en el directorio de documentación del repositorio de Github de Telecom Design [4], con acceso público (datasheet, guía de uso y manual de referencia, solamente).

En esta sección, no se pretende mostrar un estudio exhaustivo de las capacidades hardware que ofrece la placa de evaluación. Se mencionarán diversos aspectos generales de utilidad para introducir el desarrollo de programación de la placa. Para mayor detalle, remitimos a las referencias antes mencionadas.

4.2. Características generales

Se comentarán, a continuación, algunos detalles que merecen la pena ser destacados acerca de esta placa de evaluación. Tomamos como referencia los enlaces antes mencionados, además de la documentación que proporciona la página oficial de SigFox con respecto a esta placa [26].

De manera general, podemos describir este dispositivo como **energéticamente eficiente**, por la baja potencia que consume. Este ahorro de energía será de gran utilidad para aprovechar, sobre todo, el uso del **GPS** al máximo, por el excesivo consumo de batería que siempre va asociado al mismo. Siguiendo con aspectos eléctricos, también se caracteriza por el uso de bajas corrientes en sus conexiones.

Siguiendo con sus ventajas; se presenta como un producto de **alto rendimiento** para su uso como transductor de radiofrecuencia y receptor de GPS. Para ello, viene armado con un **procesador ARM Cortex M3**; que encaja en el entorno en el que se espera que trabaje el dispositivo, como se comentó en apartados anteriores, y trabajando con una frecuencia de reloj de 32 KHz. En cuanto a capacidad se refiere; con una memoria RAM de 16 KB y una memoria Flash de 128 KB, no se tendrán problemas de almacenamiento, salvo casos extremos de uso.

El **acelerómetro** que incorpora también se caracteriza por ser de consumo muy bajo de potencia, con capacidad de captar movimiento o caída libre en función de la programación elegida, con una precisión de hasta +/- 16 G, y con un amplio abanico de aplicaciones en función del caso elegido.

En cuanto a sus capacidades de **transmisión**, se adapta a la tecnología radio UNB que ofrece SigFox, y está preparado para la transmisión de larga distancia en las bandas ISM.

Como características adicionales, incorpora timers para poder realizar labores de **temporización**, y funcionalidades para poder formar una **red local de sensores**, disponiendo de modos de transmisión a la red local y de encaminador de mensajes a la red SigFox.

Además, permite **añadir multitud de sensores externos** mediante las conexiones vía UART, I^2C, ADC, DAC o GPIO, por citar algunos ejemplos.

Todo este comportamiento viene recogido, de manera esquemática, en el siguiente diagrama de bloques del dispositivo:

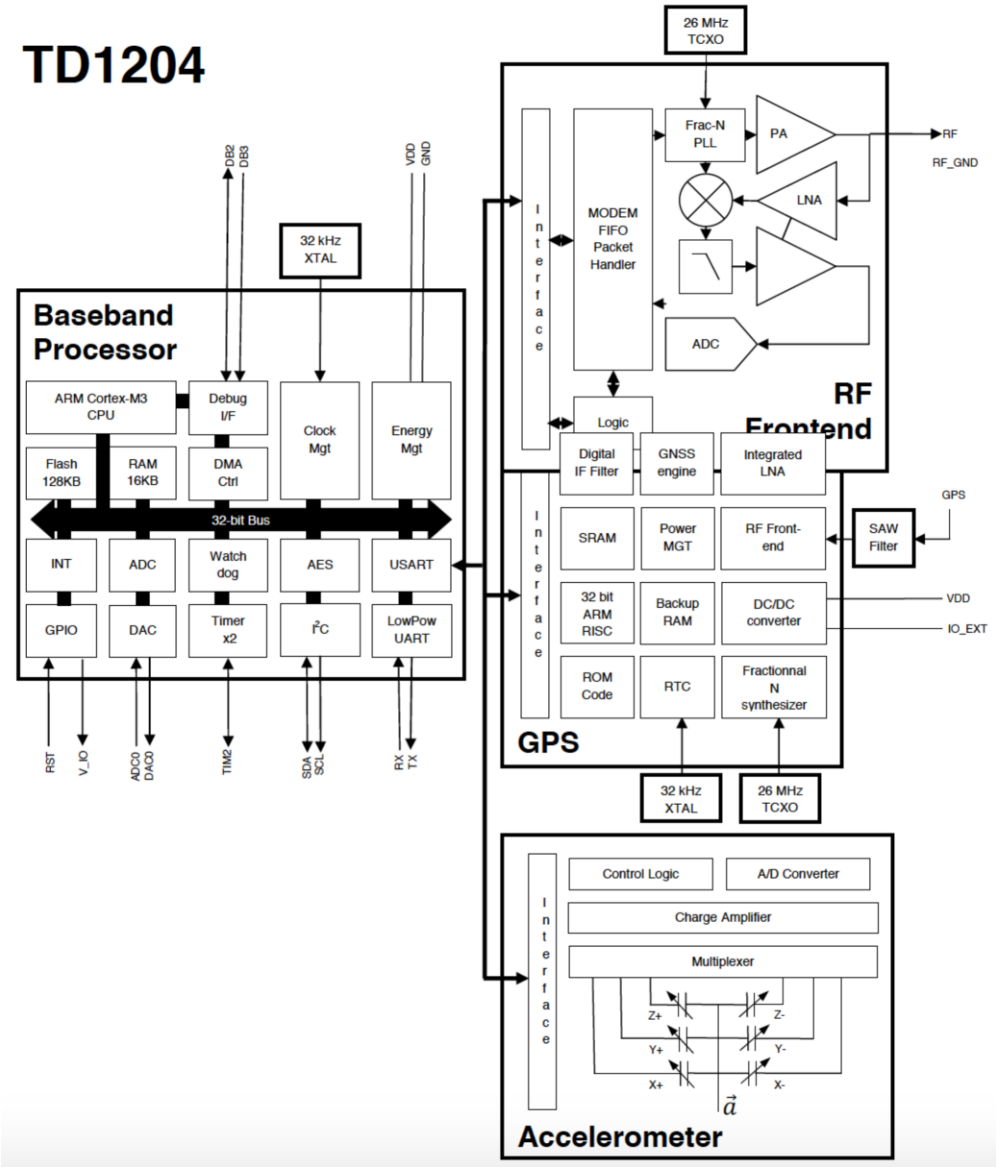


Figura 4-2. Diagrama de bloques del TD1204. *Datasheet del TD1204.*

4.3. Presentación de la placa de evaluación

La placa se presenta en un paquete en el que se adjunta la pareja **ID-clave** para acceder al panel de control del dispositivo en la página web de desarrolladores de Telecom Design (que no usaremos, aunque se deja su referencia para interesados [27]).

El paquete contiene una serie de componentes, a decir:

- La propia **placa de evaluación TD1204**.
- Una antena **GPS**.
- Un cable de **TTL a USB** para conexión por el puerto serie.
- Una **antena** para la transmisión vía radio en la banda de 868 MHz.

Siguiendo los pasos que se proponen en la guía de usuario antes comentada, se llegará a este **montaje final**, que permitirá aprovechar todas las posibilidades de la placa una vez se alimente; por ejemplo, cuando se conecte el cable USB al puerto serie del ordenador en el que se realicen las pruebas.

Figura 4-3. Montaje final del TD1204. *Guía de uso del TD1204.*

4.4. Configuración del módem con comandos Hayes

La guía de usuario del dispositivo nos orienta, tras realizar el montaje anterior, a conectarlo al ordenador para **actualizar el firmware** que incorpora, y a configurarlo mediante la colección de **comandos Hayes** que se presentan en el manual de referencia de la placa.

El conjunto de **comandos Hayes** [28] forma un lenguaje desarrollado por la compañía Hayes Communications, y que se ha convertido en el estándar empleado para configurar y parametrizar módems. Se conocen también como comandos AT ya que, la mayoría de estos comandos, comienzan con esa secuencia de caracteres.

A continuación, haremos un análisis de las posibilidades que nos ofrecen estos comandos para la configuración del dispositivo.

4.4.1. Drivers y programas necesarios

Desde la primera vez que conectemos la placa de evaluación al ordenador, ya podemos comenzar a configurarlo a partir del firmware que lleva cargado (aunque no está en su última versión). Restaría por instalar los **drivers y programas** necesarios en el equipo para poder actualizar el firmware y gestionar la placa de evaluación.

Citamos estas utilidades a continuación con sus referencias, aunque se mencionan con todo detalle en la guía de uso. Notar que todas las versiones utilizadas de estas utilidades son las correspondientes a sistemas Windows; entorno donde se realizará el proyecto completo:

- **Drivers del cable USB** (versión 2.12.00) [29], eligiendo el sistema operativo que corresponda (Windows, en el caso de este proyecto).
- La **aplicación TD Loader,** antes comentada, para la instalación de aplicaciones en la placa.
- La **aplicación Ublox u-center** (utilizada la versión 8.13) [30]. Su uso es opcional, pero permite

mostrar información detallada de las capturas realizadas por el GPS de la placa. Sólo está disponible para plataformas Windows.

- La **aplicación PuTTY** (release 0.64) [31], para conectarse a la placa por el puerto serie e interactuar con ella. Valdría cualquier otra opción para conectarse por el puerto serie, aunque se sugiere el uso esta aplicación.

Los **drivers** del cable y el **Ublox u-center** requieren seguir un proceso de **instalación**. Tras ello, **actualizaremos el firmware** de la placa utilizando el **TD Loader**, que se deberá configurar con estos parámetros:

- El **puerto serie** por el que la placa de evaluación se conecta al ordenador. El número concreto puede verse en **Panel de control > Hardware y sonido > Administración de dispositivos > Puertos (COM y LPT)> USB Serial Port (COMx)**. Ese valor **x** deberá ponerse en este campo.
- En la opción de **producto**, elegimos la opción **TD1204 EVB**.
- En el nombre del fichero, ponemos la ruta donde se encuentre el firmware **.bin** a usar.

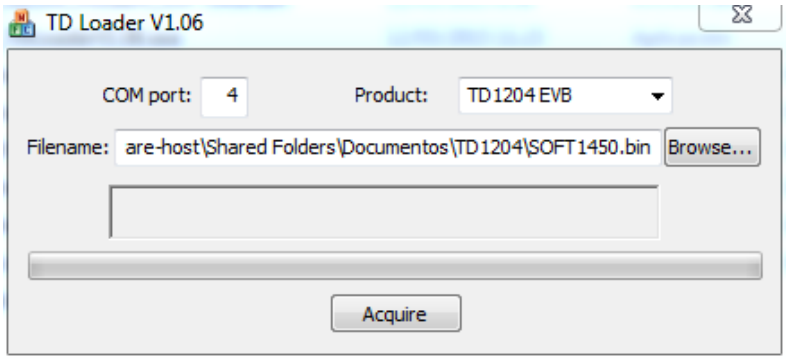


Figura 4-4. Configurando el TD Loader para la actualización de firmware.

Por último; pulsando en **Acquire** (con la placa conectada al ordenador vía USB), se lanzaría el firmware a la placa y se actualizaría tras completarse el proceso.

Tras esto, podemos pasar a configurar el módem a través de los comandos Hayes/AT. Para ello, hacemos uso de la herramienta **PuTTY**, con los parámetros de configuración mostrados en la figura (sustituyendo COM4 por COMx, siendo x el valor obtenido en el paso anterior y que hace referencia al puerto serie por el que se ha conectado la placa):

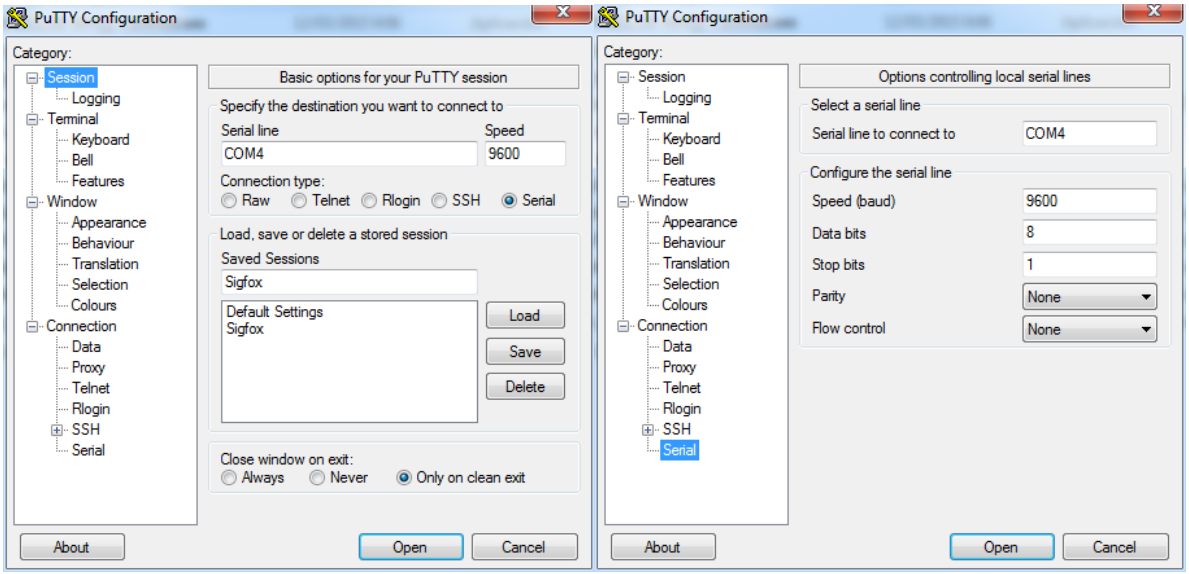


Figura 4-5. Configuración de PuTTY para conectarse a la placa por el puerto serie.

Con esto, y tras pulsar **Open**, nos conectaríamos a la placa a través del puerto serie, y podríamos hacer uso de los **comandos Hayes** indicados en el manual de referencia, y que se estudiarán con mayor profundidad en el siguiente apartado.

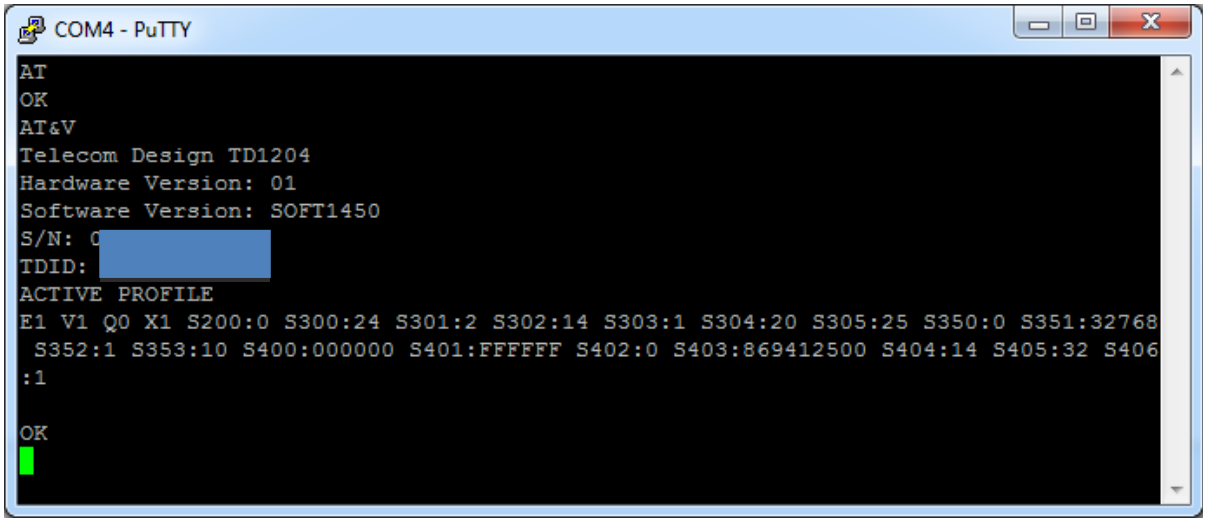


Figura 4-6. Utilizando comandos Hayes en la conexión con la placa por el puerto serie.

4.4.2. Comandos Hayes: posibilidades y limitaciones

En la figura anterior, se presentaba la comunicación con la placa a través del puerto serie, escribiendo una serie de comandos AT y esperando la respuesta del dispositivo. Si el comando se ejecuta correctamente, el dispositivo devuelve el mensaje OK, además de cualquier otro contenido que pueda generar dicho comando. En caso contrario, devuelve ERROR.

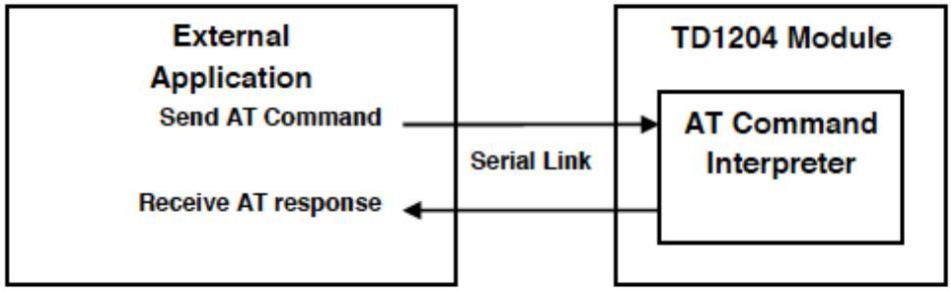


Figura 4-7. Comunicación entre el ordenador y la placa por el puerto serie con comandos AT. *Manual de referencia del TD1204.*

Aunque en el manual de referencia de la placa se pueden encontrar todos los comandos disponibles perfectamente detallados, conviene mencionar cuál es la **clasificación** que sugiere Telecom Design para estos comandos, distinguiendo entre los siguientes grupos.

4.4.2.1. Configuración básica y envío de mensajes

En este grupo, se pueden configurar **aspectos básicos** del dispositivo, como el formato de presentación de las respuestas, hacer eco, mostrar la ayuda, etc. Dentro de este grupo, aparece la posibilidad de **enviar mensajes al backend de SigFox** en diversos formatos: un solo bit, trama, mensaje o mensaje con asentimiento (sólo si el dispositivo tiene suscripción de comunicación bidireccional activa).

Tanto las tramas como los mensajes se envían utilizando 12 bytes como máximo, en hexadecimal. Por ejemplo, probemos a enviar un mensaje al backend (comando **AT$SS**) de la forma **AT$SS=48 6F 6C 61 21**, con el envío de 5 bytes.

Si nos vamos al backend, podremos ver el **mensaje** enviado, que se muestra en la siguiente figura:

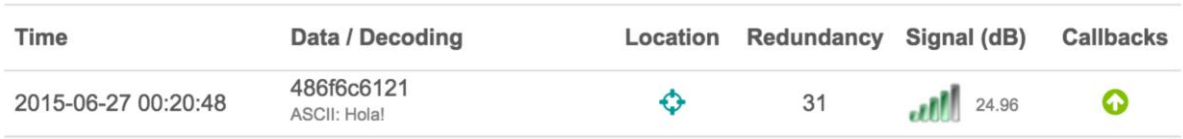


Figura 4-8. Mensaje enviado, visto desde el backend de SigFox.

Notar que la elección de los bytes anteriores no ha sido casual, y en la decodificación del mensaje en ASCII se puede ver por qué. Esto se relaciona con lo que se comentaba acerca de los mensajes en SigFox: su formato viene determinado por el fabricante. En concreto, Telecom Design ha elegido la **codificación** definida en la **recomendación T.50 (09/92) de la ITU**, llamada *Alfabeto internacional de referencia (anteriormente Alfabeto internacional nº 5ó IA5). Tecnología de la información – Juego de caracteres codificado de siete bits para intercambio de información*[32]. En dicha norma, se recoge una asignación única de caracteres a parejas de valores hexadecimales. Por ejemplo; 48 corresponde a la "H", y 61 a la "a" (utilizados en el mensaje anterior).

Esta asignación será fundamental para decodificar los mensajes que nos lleguen tanto al backend como al servidor web que genere las estadísticas, y así poder interpretar la información enviada.

4.4.2.2. Configuración del núcleo del dispositivo

Establece ciertos parámetros de configuración del núcleo del equipo, como puede ser el **watchdog** o la **asignación de pines para entrada y salida**.

4.4.2.3. Configuración de LAN

Este conjunto de comandos nos ofrece la posibilidad de configurar una **red local de sensores**, de manera que podemos elegir qué dispositivos son **emisores** de información, y qué otros dispositivos se encargan de **encaminarlos** a la red SigFox. Al disponer de sólo una placa de evaluación, no se ha considerado esta opción.

4.4.2.4. Plataforma Sensor

Aquí se introduce la posibilidad de vincular el dispositivo o una red de sensores a la plataforma **Sensor**, con diversos comandos para **monitorización** de diversos eventos en la placa; como pueden ser temperatura, batería o reinicio de la placa, entre otros, además del **envío de mensajes** a dicha plataforma.

Para los intereses del proyecto, tampoco hará falta recurrir a esta serie de comandos. Además, mencionar la complejidad del proceso de registro del dispositivo en Sensor. A pesar de seguir paso a paso el manual de referencia y la guía de uso, no se consiguió en ningún momento establecer conexión con esta plataforma y monitorizar la placa desde allí, por lo que se descartó su uso definitivamente.

4.4.2.5. Geolocalización

Recoge los comandos que hacen uso del **GPS** y el **acelerómetro**. Interesa comentar el uso del GPS; el cual, combinado con la aplicación **Ublox u-center** ya comentada, nos permite traducir toda la información que recibe el dispositivo vía GPS, que sigue la especificación NMEA, en estadísticas útiles para la geolocalización.

Como ejemplo básico, mencionar el indicado en la guía de uso de la placa, que sugiere el comando **AT$GPS=1,16,0,0xFFFF,1,1** (para más detalle, cada uno de los parámetros viene explicado en el manual de referencia). Tras lanzarlo, comienzan a mostrarse datos en el terminal periódicamente, sin posibilidad de volver a interactuar con la placa salvo que se desconecte y se vuelva a conectar.

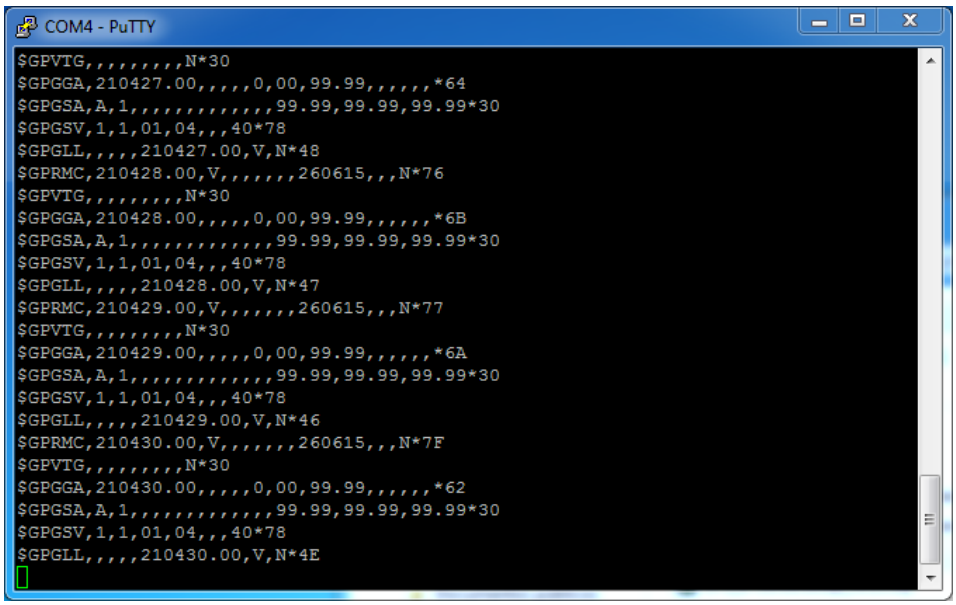


Figura 4-9. Salida del comando AT$GPS.

Para ver las estadísticas con un formato más legible, hacemos uso de la **aplicación** antes comentada. Para ello, abriremos la aplicación, y elegiremos el puerto **COMx** en el que se está realizando la conexión (deberemos cerrar la conexión a través de PuTTY o similares, de estar abierta, para poder hacer uso de dicho puerto en esta aplicación).. Para ello, seleccionamos el puerto como se indica en la figura. También se puede hacer desde la barra de herramientas, con **Reveicer > Port**.

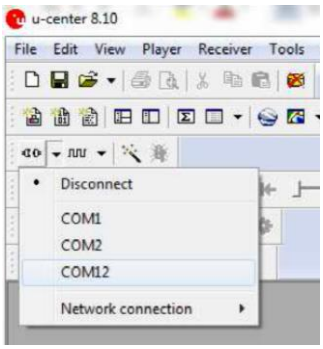


Figura 4-10. Elección del puerto serie en Ublox u-center. *Guía de uso del TD1204.*

Tras esto, y asegurándonos que la tasa del enlace es de **9600 bps** (se puede ver en **Receiver > Baudrate**), ya podremos visualizar las estadísticas que genera el GPS.

Notar que puede pasar un tiempo (minutos) hasta que se consiga la suficiente calidad de la señal GPS recibida para poder mostrar datos con una cierta precisión. Una vez ocurra, las posibilidades son múltiples: desde la información básica que se muestra en la parte derecha de la aplicación, hasta la generación de gráficas en tiempo real, estadísticas en hojas de cálculo que te muestran los valores máximos, mínimos y medios alcanzados, o el seguimiento de la posición a través de Google Earth.

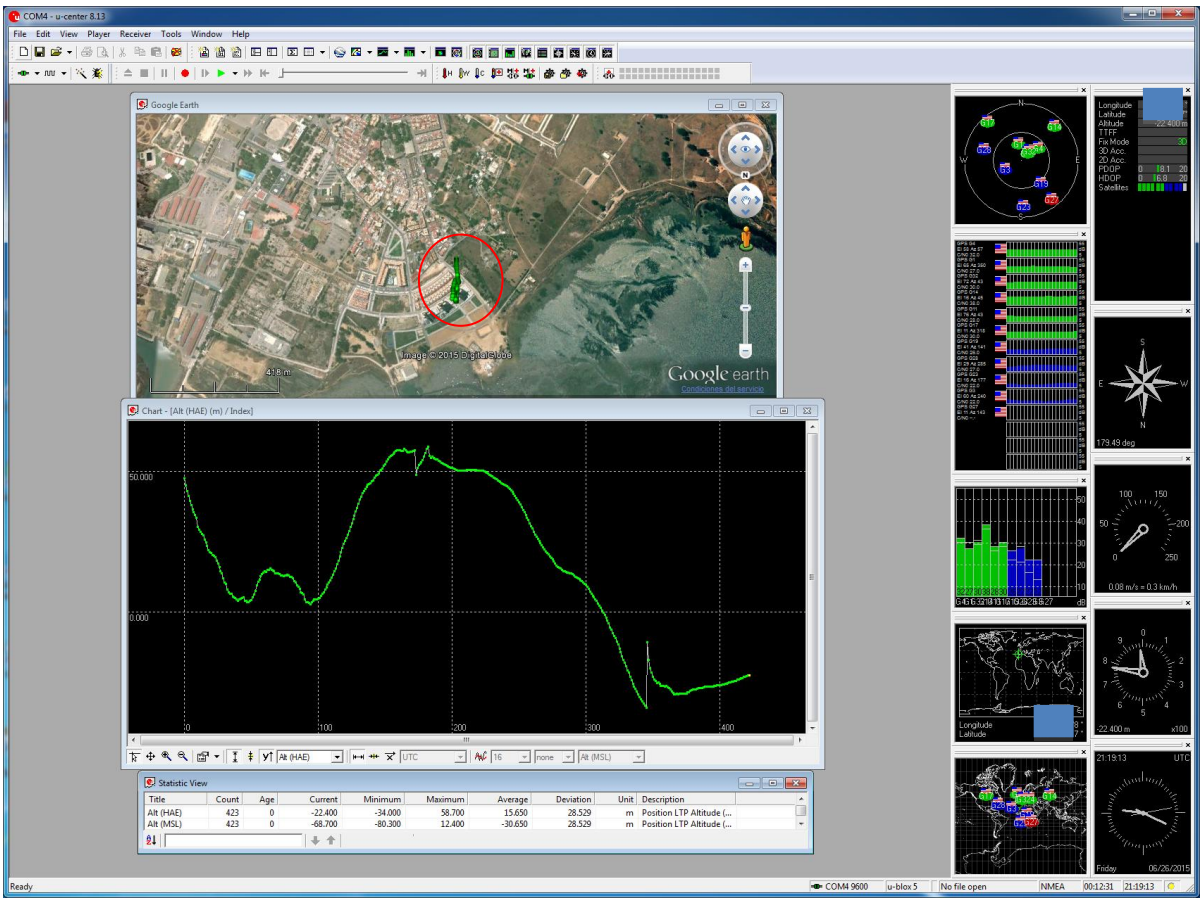


Figura 4-11. Ublox u-center en funcionamiento, con todas las posibilidades antes comentadas.

Recalcar, tras mostrar la figura anterior, la precisión del GPS. Vemos como se ha marcado como posiciones válidas todas aquellas que están dentro del círculo rojo de la figura. Aquí entran aspectos de conexiones a un número mayor o menor de satélites para comparar la posición, o el estar situado en zonas de interior o en el exterior.

4.4.2.6. Limitaciones observadas

Si bien la combinación de todos estos comandos permite la configuración y monitorización automática de la placa, su uso presenta grandes limitaciones, como la **imposibilidad de programar aplicaciones personalizadas**, con funcionalidades más complejas que los que permiten estos comandos. El problema que causa la inicialización del GPS es un buen reflejo de esto: cuando se configura, la placa entra en el modo de capturade GPS en primer plano, dejando de leer el puerto serie, luego cualquier comando Hayes introducido tras iniciar esta captura no será atendido por la placa.

Ante esta problemática, se presentan dos **alternativas** que permitirían la programación personalizada de la placa de evaluación: el **uso de Arduino** como intermediario entre la placa y el ordenador, o el aprovechamiento del **SDK y las librerías** de Telecom Design para programar el TD1204 directamente.

4.5. Integración con Arduino

Arduino [33] es una plataforma electrónica de código abierto con un hardware y software fácil de usar y programar, permitiendo realizar multitud de proyectos a partir de sus potentes características técnicas. Gracias a ello, es capaz de recoger información de diversos sensores conectados a él y controlarlos a través de las aplicaciones diseñadas. Estas aplicaciones pueden escribirse en el lenguaje propio de Arduino, y ser usadas desde el IDE que proporciona en su página oficial para su prueba y depuración.

Figura 4-12. Placa Arduino UNO R3, utilizada para las pruebas del proyecto. *Web oficial de Arduino.*

Aprovechando estas funcionalidades, se abre la posibilidad de utilizar el Arduino como **punto intermedio** entre la placa y el ordenador, de manera que sea Arduino quien le mande información a la placa, mientras el usuario programa la aplicación para Arduino, y no para la placa.

Restaría por **conectar adecuadamente** la placa al Arduino, de manera que el pin de recepción del puerto serie de la placa esté conectado al pin de transmisión del Arduino, y que el pin de transmisión de la placa vaya al pin de recepción del Arduino. Además, ambos dispositivos deben conectar sus pines de alimentación y tierra para que sean comunes. Los pines que se deben utilizar se pueden comprobar en la hoja de características o esquemáticos del Arduino [34] y del cable TTL-USB que lleva el TD1204 [35]. El resultado sería el siguiente:

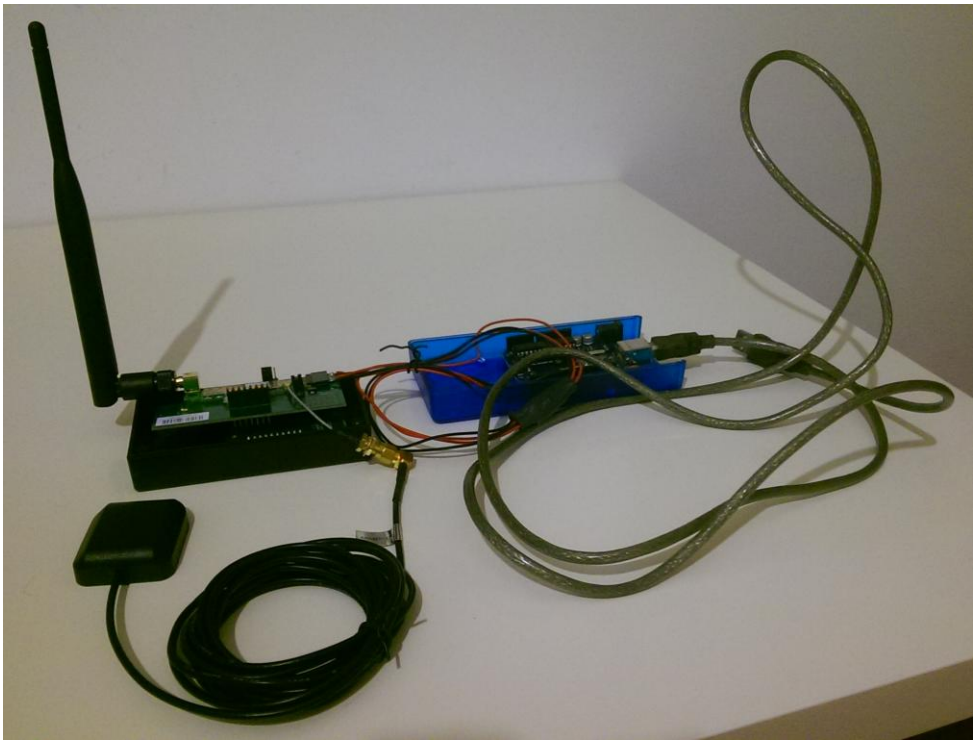


Figura 4-13. Montaje con conexión entre Arduino y TD1204.

Tras conectar el Arduino al ordenador, bastaría con programar aplicaciones para el Arduino y lanzarlas a través de su IDE. Con esto, lograríamos controlar el TD1204 a través de la aplicación diseñada para Arduino.

Esa aplicación debe hacer uso de la conexión serie entre Arduino y TD1204 con la librería **SoftwareSerial**, de Arduino.

Con esto, se pueden realizar programas sencillos, como el **enviar comandos AT desde Arduino** para que los ejecute la placa, devolviendo los resultados al Arduino y mostrándolos por el monitor serie (como si de una conexión de PuTTY se tratase).

El código de este ejemplo sería el siguiente:

```
01 /*
  02  File SoftwareSerialSigfox.ino
  03  Version 1.0
  04  By Ramón Pérez
  05
  06  The basecode is inspired by the example SoftwareSerialExample, which is
  07  in the public domain. It was modified by Tom Igoe, and based on Mikal
  08  Hart's example.
  09
  10  With this code, Arduino One will be able to:
  11  -Receive from hardware serial and send to software serial the AT command
  12  we want to use to TD1204.
  13  -Receive from software serial and send to hardware serial the response from
  14  TD1204.
  15
  16  */
  17
  18 #include <SoftwareSerial.h>
 19 #define SSerialRX 10  // RX is digital pin 10 (connect to TX to TD1204).
  20 #define SSerialTX 11  // TX is digital pin 11 (connect to RX of TD1204).
  21
  22 SoftwareSerial mySerial(SSerialRX, SSerialTX); // Instance of software serial port.
  23
  24 void setup()  // Initial configuration.
  25 {
  26   // Open serial communications (serial monitor) and wait for a port to open.
  27   Serial.begin(9600);    // Set the data rate.
  28
  29   // Start the software serial port to TD1204
  30   mySerial.begin(9600);  // Set the data rate.
  31
  32   // Informative messages
  33   Serial.println("Communication is opened");
  34   Serial.println("Please introduce the AT command you'd like to send");
  35   Serial.println("Then, wait for TD1204 response and introduce another command");
  36 }
  37
  38 void loop()   // Run over and over.
  39 {
  40   // Look for data from TD1204.
  41   if (mySerial.available()) {
  42     // Read the response and show it on serial monitor.
  43     Serial.write(mySerial.read());
  44   }
  45
  46   // Send data to TD1204.
  47   if (Serial.available()) {
  48     // Read the AT command and send it to TD1204.
  49     mySerial.write(Serial.read());
  50     delay(1);  // In order to see the AT command written in ASCII code.
51   }
  52 }
```

Notar la estructura del código, que parte de un proceso de **inicialización**, y luego se pasa a un bloque que se **ejecuta continuamente**. Esta lógica será de utilidad para comprender la programación de la placa que se justificará en próximas secciones.

Con este código, obtenemos el siguiente resultado, similar al de la figura 4-6:

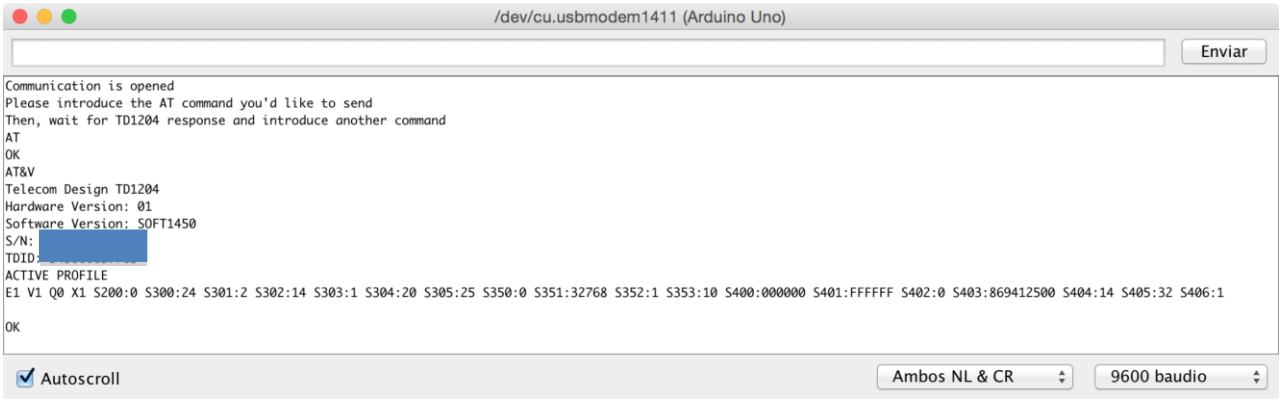


Figura 4-14. Resultado de la ejecución del código de ejemplo.

A pesar del sencillo lenguaje de programación que nos ofrece Arduino, y aunque es una opción de desarrollo que se está extendiendo considerablemente en múltiples entornos, la **falta de documentación con respecto a la familia TD12xx** y la complejidad de realizar aplicaciones sin hacer uso de las herramientas que proporciona Telecom Design han empujado a dejar de lado esta posibilidad, que queda abierta como una línea futura de investigación.

4.6. Elección del lenguaje de programación e IDE

Hemos podido ver la utilidad de los comandos Hayes para configurar aspectos básicos del dispositivo, pero también se hace notoria la escasez de opciones que ofrece para la programación de aplicaciones personalizadas. También se ha comprobado que Arduino, aunque potente en funcionalidad y posibilidades, resulta complicado de integrar ante la notoria falta de documentación al respecto.

Por eso mismo, se hará uso del **SDK y las librerías** que proporciona Telecom Design para la programación de aplicaciones, que nos permitirá diseñar la solución final a la problemática que se planteará.

4.6.1. Lenguaje de programación C

Para el desarrollo de la aplicación, se hará el uso del **lenguaje C**. Su uso está altamente extendido para la programación de microcontroladores, con un lenguaje más cómodo para el programador con respecto a otras opciones como el lenguaje ensamblador.

Se trata de un lenguaje de propósito general, sencillo y potente al manejar características de bajo nivel. Su elección viene también directamente influida por la codificación de **las librerías de Telecom Design**, que también vienen programadas en C.

Para acceder a estas librerías, será necesario **registrarse como desarrollador** en la página de desarrolladores de Telecom Design [36], para poder acceder al directorio privado de su repositorio de Github que contiene todo el código de las librerías [4]. Bastará, para ello, con introducir un correo y una contraseña para acceder al panel de control, y una vez accedido, entrar en el apartado *My Account* y sincronizar nuestra cuenta de Github (se requiere una cuenta ya realizada) con el repositorio de Telecom Design. Tras esto, se tendrá acceso al repositorio privado con el código.

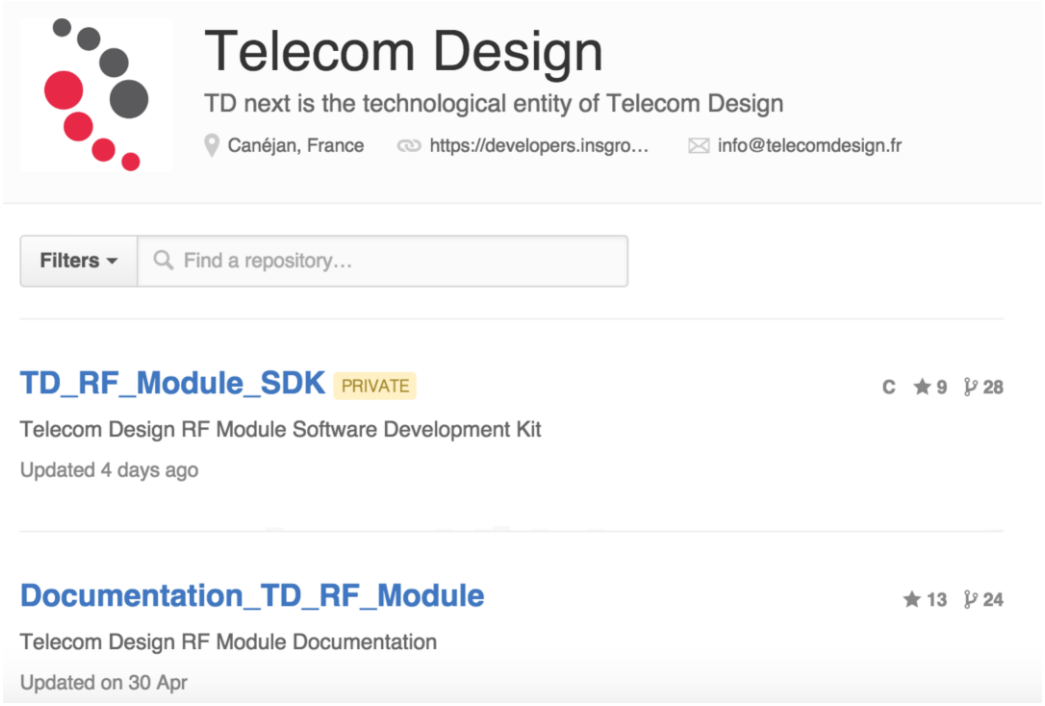


Figura 4-15. Acceso al directorio privado del repositorio de Github de Telecom Design, con las librerías en C para la programación de aplicaciones.

4.6.2. Eclipse como IDE

Tenemos las librerías, pero faltaría integrarlas en un entorno de desarrollo para facilitar el trabajo con las mismas, tanto en la codificación como en la compilación para obtener los ejecutables.

Telecom Design nos proporciona una versión adaptada a sus placas de evaluación de la **plataforma Eclipse** (descarga en [3], en el apartado *Tools*). A pesar de ser multiplataforma por definición, **esta adaptación sólo funciona en sistemas operativos Windows**.

Aunque esta limitación en el sistema operativo pueda suponer una desventaja, lo cierto es que la facilidad de configuración de Eclipse gracias a la documentación oficial permite **montar el entorno completo**, con todas las librerías, y comenzar con el desarrollo de aplicaciones **en pocos minutos**. Este procedimiento queda perfectamente detallado en el **Anexo B**, al estar almacenada esta documentación en el directorio privado del repositorio de Github.

Una vez quede montado el escenario, pasamos a la **codificación de la aplicación**, que será descrita en la siguiente sección con todo detalle.

5 PROGRAMACIÓN DE LOS CASOS DE USO DEL TD1204

"La función de un buen software es hacer que lo complejo aparente ser simple"

- Grady Boock; uno de los creadores de UML, entre otros hitos. -

Haciendo uso de las herramientas introducidas en la sección anterior, se pretende cumplir uno de los objetivos marcados en la realización de este proyecto: programar una **aplicación** para el módem TD1204, que responda a diversos **casos de uso** definidos previamente por Wellness Smart Cities, y que obtenga los **resultados** esperados de esa especificación.

Esta sección pretende incidir en la **metodología** seguida para encontrar la solución que se presentará finalmente, en base a los siguientes pasos a seguir:

- **Definición del problema** a resolver, con los **casos de uso** que se quieren tratar y todas las **especificaciones** de la solución que se desea obtener.
- **Planteamiento de la solución**, valorando todas las posibilidades extraídas del estudio de la definición del problema, y justificando posibles **modificaciones** sobre las especificaciones planteadas.
- **Diseño de la solución**, una vez se acuerden las modificaciones planteadas y se llegue a una visión común del resultado a obtener por ambas partes.
- **Codificación de dicha solución**, y obtención del **resultado final.**
- **Pruebas** y **verificación de funcionamiento**, que se tratarán en la siguiente sección.

5.1. Definición del problema

5.1.1. Casos de uso

Para el desarrollo de la aplicación, se desea la monitorización automática por parte del TD1204 de los siguientes **casos de uso**, que se enumeran a continuación:

- **Alarma de temperatura:** se debe mandar una alarma cuando se detecte que se ha superado un umbral de temperatura. Este umbral puede ser configurado desde el servidor.
- **Alarma de posición**: se enviará una alarma en el caso de que se detecte un cambio de posición con respecto a otra inicialmente conocida. Para ello, se trabajará con las librerías de manejo del GPS que

incorpora la placa de evaluación.

- **Alarma de batería baja**: se mandará una alarma si la batería está próxima a agotarse.
- **Alarma de detección de movimiento**: la placa enviará una alarma si detecta algún movimiento. Se hará uso de las librerías de manejo del acelerómetro que viene integrado en la placa.
- **Alarma de detección de caída libre**: por último, se debe enviar una alarma en el caso de detectar una caída libre del sensor. Igualmente, se requerirá el uso del acelerómetro.

5.1.2. Especificaciones

Si bien se deja libertad para el desarrollo del código que cumpla con los casos de uso antes citados, se mencionan ciertas **especificaciones** a cumplir en el proyecto:

- El código estará escrito en **lenguaje C**, e íntegramente en **inglés**, tanto por las variables utilizadas como por los comentarios añadidos.
- Se hará uso de las **librerías proporcionadas por el SDK de Telecom Design**, que serán la base para programar el envío de los **datos** que se lean desde **distintos tipos de sensores** (temperatura, acelerómetro, batería y GPS, en nuestro caso).
- El código estará integrado en **un solo fichero**. Cada **caso de uso** se tratará en una **función** concreta, con todas las **estructuras de datos** necesarias para su correcto funcionamiento, y programando las **alarmas** de envíos de datos al backend.
- La solución propuesta debe ser **flexible**, en cuanto a poder añadir **nuevos sensores sin modificar** el código ya realizado.
- **No** hay **limitación** en el **tamaño** del código y, a la postre, del ejecutable generado, puesto que la memoria es lo suficientemente grande (16 KB de RAM y 128 KB de Flash) como para alojar el programa sin ningún problema asociado a este aspecto.
- Se programarán los intervalos de alarmas e interrupciones sabiendo que el **límite de mensajes** permitidos **por día**, en base a la suscripción elegida, es de **140 mensajes**. El código no debe contemplar el cese de envío de mensajes una vez alcanzado ese límite.
- La **comunicación** será **unidireccional**, por la suscripción contratada, con origen el TD1204 y destino el backend.

5.1.3. Lógica de programación

Se debe hacer uso de las dos **funciones base** que implementan todas las aplicaciones desarrolladas para el TD1204. Su utilidad es similar a las usadas en Arduino, y son las siguientes:

- **Función setup:** es la **primera función** que se ejecuta en la placa, y sólo se llama **una vez**. Se ocupa de **inicializar** la placa y programar las **configuraciones** necesarias para que realice las tareas requeridas y obtenga los resultados esperados.
- **Función loop**: se llama **tras** finalizar la función **setup**, y **se repite indefinidamente** hasta que el dispositivo se apague o se resetee. En este bloque, se ejecutará toda la **lógica** de ejecución del código cíclicamente, permitiendo que el dispositivo funcione de forma **autónoma**.

En base a estas funciones, se desea que el código siga la secuencia recogida en el siguiente **diagrama de actividad**:

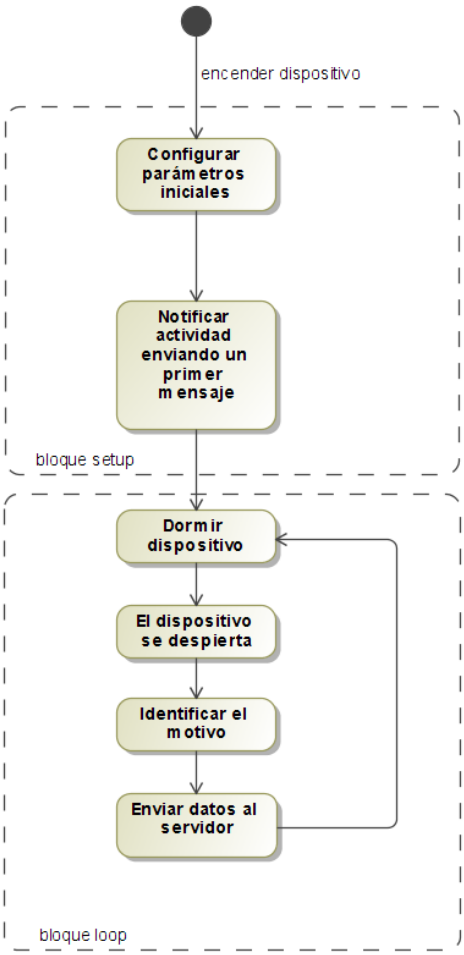


Figura 5-1. Diagrama de actividad con la funcionalidad básica que debería seguir la solución requerida.

Queremos incidir en varios detalles sobre algunos de los pasos antes mencionados:

- La **configuración inicial** deberá contemplar los **pines** de entrada/salida que se vayan a utilizar, el uso del **watchdog** para resetear el sistema en caso de bloqueos, y la programación correcta de los **sensores** a utilizar: acelerómetro, GPS, temperatura y batería.

  Además, se establecerá una **alarma de RTC**, que enviará datos de batería y temperatura periódicamente. En esta configuración inicial, se deberá especificar **cada cuánto tiempo** se van a medir y enviar estos valores al backend.

- Al **identificar el motivo** por el que la placa se **despertó**, distinguiremos estos casos:
  - **Interrupción de RTC:** se medirán temperatura y batería.
  - **Interrupción de temperatura/batería**: se leerá el valor que corresponda, y se comprobará el estado en el que se encuentra la placa en base al umbral que se haya superado.
  - **Interrupción de acelerómetro**: se activará el GPS, en búsqueda de la posición actual del dispositivo, y se comprobará cuál ha sido el motivo de la interrupción (movimiento o caída libre).

  En todos los casos, se concluirá **enviando los datos** leídos al backend de SigFox.

- Se dejará **libertad al programador** para implementar los **casos no especificados** en las líneas anteriores; como puede ser la captura de la primera posición conocida, que sirva para compararla con las nuevas posiciones y saber si ha habido un cambio en la posición con respecto a esa primera posición.

5.2. Planteamiento de la solución y actualización de las especificaciones

Con las **especificaciones** anteriores, comienza el estudio en profundidad de las posibilidades que nos ofrece la placa de evaluación para **implementar** todas las funcionalidades requeridas.

Si bien la solución final cumple con la mayoría de los requisitos pedidos, el modo de funcionamiento de la placa ha provocado cambios en algunos de ellos, variando incluso parte del enfoque que se mostraba en el diagrama anterior.

Por ello; este apartado mostrará, en orden de aparición, los **problemas** surgidos durante la codificación de la solución, y los **cambios** en los casos de uso y las especificaciones que han provocado como resultado. Junto a estos problemas, se comentarán **soluciones** propuestas para ciertos temas no mencionados en la especificación, y que se dejaron a libre elección del programador. Todos estos puntos han sido aprobados, uno a uno, por Wellness Smart Cities, al ser modificaciones de su especificación de partida.

5.2.1. Configuración desde servidor no disponible

El caso de detección de alarma de temperatura trabaja con **umbrales**; de manera que, al sobrepasar cierto umbral, se deberá notificar al backend este hecho. Además, se recalcaba que ese umbral debía ser **configurable desde un servidor externo**, y no como parámetro fijo en la aplicación.

Este requisito resulta **imposible** de cumplir debido a que sólo se permite la **comunicación unidireccional** en nuestro caso de estudio, por la suscripción realizada. De esta forma, nunca se dará el caso de que el backend u otra aplicación envíen datos a la placa, lo que desemboca en tener que **descartar** este requisito.

5.2.2. Histéresis en las alarmas de temperatura y batería

Volviendo a los **umbrales** antes mencionados, se presenta un problema adicional. Tanto temperatura como batería tendrán varias zonas de funcionamiento, delimitadas por dichos umbrales (que son valores concretos de temperatura o batería), y la notificación se enviará al backend si se produce un salto de una zona a otra. Sin embargo, podría ocurrir que el **valor** de temperatura o batería **oscilara en torno a un umbral**; detectándose continuamente estos cambios de zona, lo que podría generar una **cantidad ingente de mensajes**.

Ante este problema de posible generación excesiva de mensajes, se decide incorporar a la detección de cambio de zona un ejemplo simplificado de **histéresis**, caracterizada por *"conservar una propiedad en ausencia del estímulo que la generó"*. De esta forma, la programación de las alarmas tendrá en cuenta **zonas intermedias de funcionamiento,** en cuanto a niveles de temperatura y batería se refiere. Esto convierte al umbral en un **intervalo de valores**, y no en un valor concreto; de manera que se necesitará **pasar de esa zona intermedia** para detectar un cambio de estado y, como consecuencia, notificar al backend.

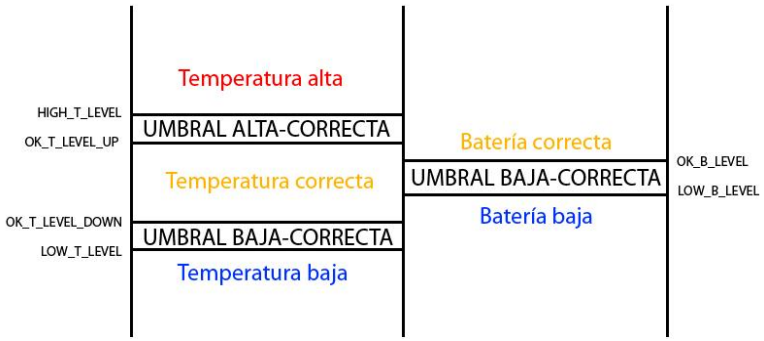


Figura 5-2. Zonas de trabajo y umbrales considerados en el TD1204. Notar que cada umbral es un intervalo, y no un valor concreto.

Se puede considerar como un caso de histéresis porque esa zona intermedia, teóricamente, pertenece a la zona de funcionamiento correcto. Por lo tanto, estamos **manteniendo el estado** de alerta (casos de temperatura alta/baja y batería baja) **en una zona** que sería **de comportamiento correcto** de la placa en cuanto a estas

magnitudes se refiere.

Por último; los valores considerados para los intervalos de cada umbral vienen determinados por las **características técnicas** del dispositivo, extraídas del datasheet.

Parameter	Value	Units
Operating Ambient Temperature Range T_A	-30 to +75	°C

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Supply Voltage Range[2]	V_{DD}		2.3	3.3	3.6	V

Figura 5-3. Rango de valores de temperatura y batería en los que funciona el dispositivo. *Datasheet del TD1204*.

En concreto; para la temperatura, se ha considerado la **temperatura ambiente** como parámetro, fijando los siguientes **valores** para dividir las zonas de trabajo:

- **Temperatura baja:** al bajar de 0 grados.
- **Temperatura correcta:** entre 10 y 55 grados.
- **Temperatura alta:** a partir de 65 grados.

El **valor de batería** se corresponde al **voltaje que entrega la batería a la placa**. Conforme se gasta la batería, ese valor va cayendo hasta el mínimo permitido por la placa para funcionar (2.3 V). De bajar de este valor, la placa dejaría de funcionar. Por ello, las zonas de trabajo consideradas se fijarán con estos **valores**:

- **Batería baja:** al bajar de 2.6 V.
- **Batería correcta:** a partir de 3 V.

En ambos casos, vemos que se deja un margen suficiente para no llegar a los valores extremos marcados por el fabricante.

5.2.3. Funcionamiento interno basado en cola de eventos

Retomando el estudio de las **alarmas** de temperatura y batería, conviene saber cuál es el **método** seguido en las librerías de Telecom Design **para la detección** de cambios en los valores de estas magnitudes, para realizar la codificación correspondiente a la hora de programar el código.

Para esta familia de módems, se ha podido comprobar que el método elegido es la utilización de un **planificador** o **scheduler**, al cual se le indica cada **cuánto tiempo** debe ser llamado, el **número de veces** que se llamará (pudiendo llegar hasta infinito) o la **función de callback** que se debe invocar cada intervalo de tiempo.

De esta forma, la placa **configurará los planificadores** que necesite utilizar (en concreto, uno para temperatura, otro para alarma, y un último para RTC) **en el bloque setup**, y se **dormirá automáticamente** al entrar en el bloque loop. Cuando pasen los intervalos de tiempo marcados en los planificadores, se llamarán a las funciones de callback indicadas en los mismos, y se realizará el tratamiento correspondiente.

Ante este cambio en el enfoque de la solución, mencionar varios detalles:

- Ya que las funciones de cada caso de uso se llaman gracias al planificador, y no a alguna sentencia incluida en el bloque **loop**, esta función puede estar **vacía**; a no ser que se codifique alguna acción a realizar periódicamente. Toda la lógica de monitorización de temperatura y batería recaerían sobre estas funciones de callback, y no sobre la función loop.
- No hace falta dormir la placa explícitamente, puesto que **la función loop se encarga de dormir la placa** automáticamente si no hay nada que procesar.
- Incidir en el detalle de que **no se puede detectar un cambio de estado** en el valor de temperatura o

batería **de manera instantánea**, ya que está limitado al intervalo fijado en el planificador. Tampoco se puede pretender poner un **intervalo muy pequeño**, ya que repercutiría directamente en el **consumo** del dispositivo.

- Tanto la gestión de los planificadores como cualquier llamada a otra función se basan en una **cola de eventos**. En dicha cola, se van almacenando los eventos, y se van **procesando por orden de llegada**. De esta forma, **no se podrán procesar dos eventos simultáneamente**; el más reciente siempre tendrá que esperar, en la cola, a que se procese el más antiguo.

5.2.4. Interrupciones incompatibles con el envío de mensajes

En las librerías del SDK de Telecom Design, se permite que el **planificador genere interrupciones** cada vez que se llama a la función de callback. Tendría sentido la incorporación de esta funcionalidad, ya que se pretende que la placa **se despierte por interrupción**, y no por un simple planificador.

El problema viene cuando, en la función de callback, se intenta llamar a **funciones de envío de mensajes** al backend, ya sea con una llamada directa o invocando otra función que realice esa llamada. Al intentar el envío, la aplicación **termina su ejecución**, invocando una interrupción software o **trap**.

Ante este problema, la solución adaptada fue **no considerar ninguna interrupción**, y utilizar las funciones del planificador en su forma más simple, como la descrita en la subsección anterior.

5.2.5. Imposibilidad de dormir la placa con GPS y acelerómetro activos

En puntos anteriores, se ha justificado el por qué debería estar vacía la función loop, quedando la placa de evaluación en estado de **reposo** hasta que fuese despertada por cualquiera de los casos de uso considerados.

La idea es lógica, ya que la placa no debería consumir energía en el caso de que no tenga nada que enviar, y debería estar a la espera hasta que algún evento ocurra.

El problema viene con dos de los casos de uso antes mencionados: **GPS y acelerómetro**. En todos los ejemplos propuestos por Telecom Design, se hacen uso de dos funciones (una para cada caso) en el bloque loop, con el nombre de *Process:* **TD_GEOLOC_Process y TD_ACCELERO_Process**. Estas funciones se ocupan de **procesar continuamente** el GPS y el acelerómetro, y notifica a la placa en caso de que ocurra alguna interrupción para actuar en consecuencia, si se ha configurado previamente. Al ejecutarse siempre cuando se activa su monitorización, **la placa nunca llega a dormirse**, aunque se le mande a dormir explícitamente con funciones específicas.

Cabría la posibilidad de omitir el uso de estas funciones para dormir la placa, como se ha comentado anteriormente, pero lo cierto es que **no pueden eliminarse del bloque loop**, y así lo especifica Telecom Design en las librerías del GPS y acelerómetro. De no usarse las funciones, ocurriría lo siguiente:

- El **acelerómetro** necesita monitorización constante para detectar un cambio en las fuerzas de aceleración que se ejercen sobre el dispositivo. Esa **monitorización** se debe activar de forma **explícita** con la función *Process*. Por lo tanto; de no llamarse, aunque se configure correctamente, nunca generará interrupciones y no se procesarán datos relativos al acelerómetro.
- El **GPS** genera todos los **datos de geolocalización** gracias a esa función; contando con atributos como fecha y hora proporcionadas por los satélites, latitud, longitud, altura, o calidad de la señal recibida. Estos datos se almacenan en una estructura, que es la que reciben las funciones configuradas para procesar la información recibida por el GPS. De no utilizarse la función *Process*, esa estructura estará siempre vacía, y tampoco se podrán procesar los datos relativos a geolocalización.

Por lo tanto, la **consecuencia** principal de este hecho será **renunciar a dormir la placa** cuando se tenga que monitorizar el GPS y/o acelerómetro.

Aunque se piense que puede tener un gran impacto en el consumo final, el **acelerómetro** incorporado en la placa se caracteriza por ser de **bajo consumo** (la función *Process* activa lleva a un consumo de 2 µA en reposo; prácticamente nulo), y el **GPS** sólo se utilizará cuando se requiera calcular una nueva posición, estando **apagado el resto del tiempo**, luego no se procesarían sus datos durante ese tiempo y el *Process* no se ejecutaría.

5.2.6. Acelerómetro con un solo modo de funcionamiento posible

Entre los casos de uso considerados para la aplicación, dos de ellos tienen relación directa con el acelerómetro: tanto la detección de **movimiento** como la de **caída libre**.

Ambos casos sugieren **modos de configuración distintos del acelerómetro** para poder distinguirse uno de otro. Si bien un movimiento puede ser en cualquier dirección y con una fuerza de aceleración mínima, la caída libre se produce en dirección vertical, perpendicular al suelo, y cayendo por acción exclusiva de la gravedad.

Sin embargo; el problema aparece cuando, a la hora de programar el acelerómetro para la detección de ambos casos, **no se permite más de una configuración** simultánea. Aunque se programen dos funciones distintas para cada caso y las configuraciones tengan parámetros distintos, la placa siempre **programará** el acelerómetro con la **última configuración elegida**.

Se podría pensar en una solución empleando programación **multihilos**, pero las librerías del SDK no incluyen ninguna funcionalidad al respecto, y las librerías estándar de hilos de C tampoco se pueden utilizar, luego esta opción quedaría descartada automáticamente.

Este inconveniente lleva a tener que **elegir** entre uno de los dos casos, descartando automáticamente el otro. La elección definitiva fue **mantener la detección de movimiento** como caso de uso, y descartar la caída libre.

El motivo es por el **entorno real** donde se moverán este tipo de dispositivos. Cabe la posibilidad de que los sensores estén en una posición fija en el **suelo**, si el terreno es propicio para ello, luego la caída libre en ese caso no tendría sentido. Por otro lado, se deben considerar también posibles **robos** de los dispositivos, lo cual se puede detectar si se está monitorizando el movimiento de la placa.

5.2.7. Trabajando con la primera posición capturada por el GPS

La configuración del acelerómetro para detectar únicamente el movimiento repercute directamente en la **configuración del GPS**. Sobre él recae el caso de uso de **cambio de posición**, del cual sólo se especifica que dicho cambio se debe realizar sobre una **posición previamente establecida**.

Podríamos **fijar en el código** esa posición inicial, pero esa configuración daría una solución **inflexible** en cuanto a la colocación del dispositivo, y requeriría cambiar la posición inicial de mover el dispositivo a otra posición. Otra opción directamente descartada, como se comentó para el caso de la temperatura, sería **configurar** esa posición desde un **servidor o aplicación externa** y enviárselo a la placa para que lo almacene.

La **solución** adoptada será **activar el GPS** en el proceso de configuración inicial del dispositivo, y cuando los datos recibidos tengan la suficiente calidad y precisión, se tomarán como primera posición conocida, y se desactivará el GPS para no desperdiciar consumo.

Como se debe tener esta primera posición para tratar con el resto de casos de GPS y acelerómetro, el temporizador asociado a la captura de la posición se fijará a infinito, luego **no parará hasta encontrar la posición**. Por ello, y por aspectos de ahorro de batería y consumo, el **modo** de funcionamiento del GPS será de **ahorro de energía**. Esto provocará que se tarde más en encontrar una posición válida y que la precisión disminuya, pero con una reducción considerable en el consumo con respecto a opciones más precisas.

Además, la **detección de movimiento** y **cambio de posición** no se habilitarán **hasta que se capture esa primera posición**. Esto tiene sentido porque un movimiento de la placa puede originar un cambio en la posición del dispositivo (por ejemplo; si ha sido robado), y se necesitaría esa posición inicial para saber si ha habido cambio en la posición.

5.2.8. Necesidad de un número mínimo de movimientos para lanzar la alarma

Si bien la detección de movimiento va encaminada a la prevención de robos del dispositivo, también podría ocurrir que la placa fuese movida o agitada por una acción distinta; por ejemplo, agentes meteorológicos como el viento.

Esto puede provocar que se den **muchos movimientos** en un corto instante de tiempo, y **notificar** todos esos movimientos al backend puede resultar **innecesario**, ya que no implicaría con total probabilidad un cambio de posición en la placa. Recordar, además, que el número de mensajes por día es limitado, y se estarían

gastando mensajes que se podrían dedicar a otro caso.

Por ello, se ha decidido que sólo se **notificará** el caso de detección de movimiento cuando se alcance un **número máximo de movimientos** (5, en nuestro caso), reiniciándose la cuenta tras mandar el mensaje correspondiente.

5.2.9. Relación entre el movimiento y el cambio de posición

En cuanto a la relación entre estos dos casos de uso, el tratamiento para la notificación es distinto: cada vez que se **detecte un movimiento**, se lanzará el **GPS** para capturar la **posición actual**. Tras esto, esta opción quedará **desactivada hasta que se capture** dicha posición, volviendo a habilitarse. Se hace para que no se vuelva a lanzar el GPS en caso de producirse otro movimiento mientras ya está habilitada una búsqueda.

¿Y qué **factores** juegan en la detección del cambio de posición? Hay que tener en cuenta diversos aspectos; no sólo la **distancia** entre el punto capturado y la primera posición fijada, sino la propia **precisión del GPS**, por ejemplo. En este caso, volveremos a utilizar el GPS en modo de **ahorro de energía**, lo cual repercute en la precisión de la posición obtenida.

Por ello, los límites marcados para considerar un cambio en la posición será que **la latitud o la longitud varíen en 0.1 minutos.** Si la diferencia de latitudes o longitudes supera ese valor, entonces se detectará este caso. Como símil, mencionar que una diferencia de 0.1 minutos en la latitud o la longitud sería equivalente, aproximadamente, a la distancia entre dos calles paralelas en una ciudad.

Esto formaría un **área rectangular,** donde la distancia entre el centro del rectángulo y el punto medio de cualquiera de los 4 lados sería de 0.1 minutos. De esta forma, de encontrarse la **nueva posición** capturada **fuera** de dicha zona, se consideraría que se está produciendo un cambio de posición.

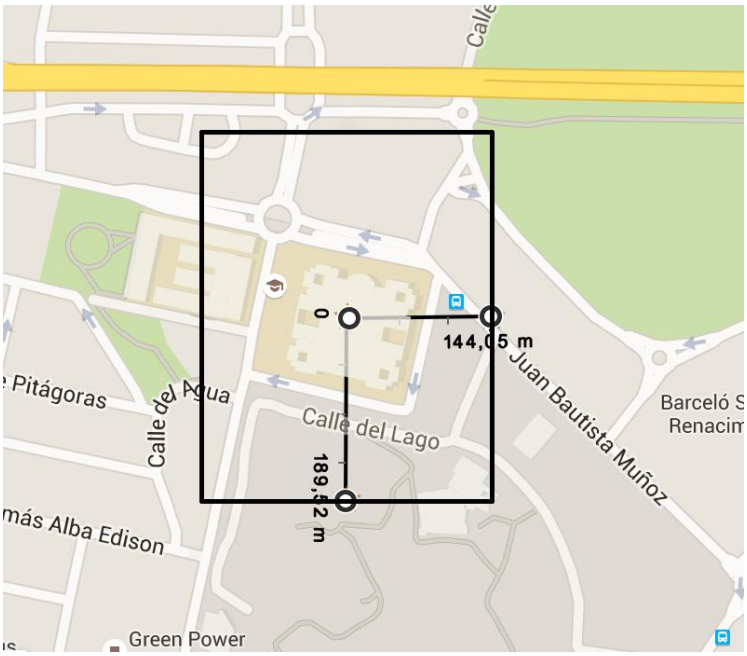


Figura 5-4. Ejemplo de dicho área rectangular, tomando como centro la Escuela Superior de Ingeniería, y traduciendo los 0.1 minutos en metros para latitud y longitud. *Captura de Google Maps.*

Con esto, **evitaríamos los problemas de precisión** del GPS, y **permitiríamos pequeños cambios de posición** de requerirse por el supervisor de la red de sensores. Notar que, en caso de robos, la probabilidad de que el dispositivo siga en esa zona sería baja, por la tendencia criminal de *"huir de la escena del crimen"*.

Como último detalle; la habilitación del GPS en este caso tendrá asociado un **temporizador**, para evitar un **consumo innecesario de energía**. Aquí no ocurre como en el caso de la primera posición, que sí debe ser

conocida para las comparaciones con otras posiciones, luego se cancelará la captura al sobrepasar ese temporizador.

5.3. Diseño de la solución

En base a la especificación original y a las modificaciones propuestas en el apartado anterior, se propone la siguiente **solución** para la aplicación final a implementar, comentando las **funciones** que se han implementado y el **diseño** seguido en su programación.

Este apartado contiene detalles exclusivamente de **modelado de software**, y no de codificación; utilizando **diagramas UML** para describir la solución propuesta.

5.3.1. Comportamiento general: funciones TD_USER_Setup y TD_USER_Loop

El nombre formal que reciben las funciones setup y loop en el SDK de Telecom Design es el de **TD_USER_Setup y TD_USER_Loop**. Para contemplar todas las posibilidades antes mencionadas, se han codificado ambas funciones siguiendo el siguiente **diagrama de actividad**:

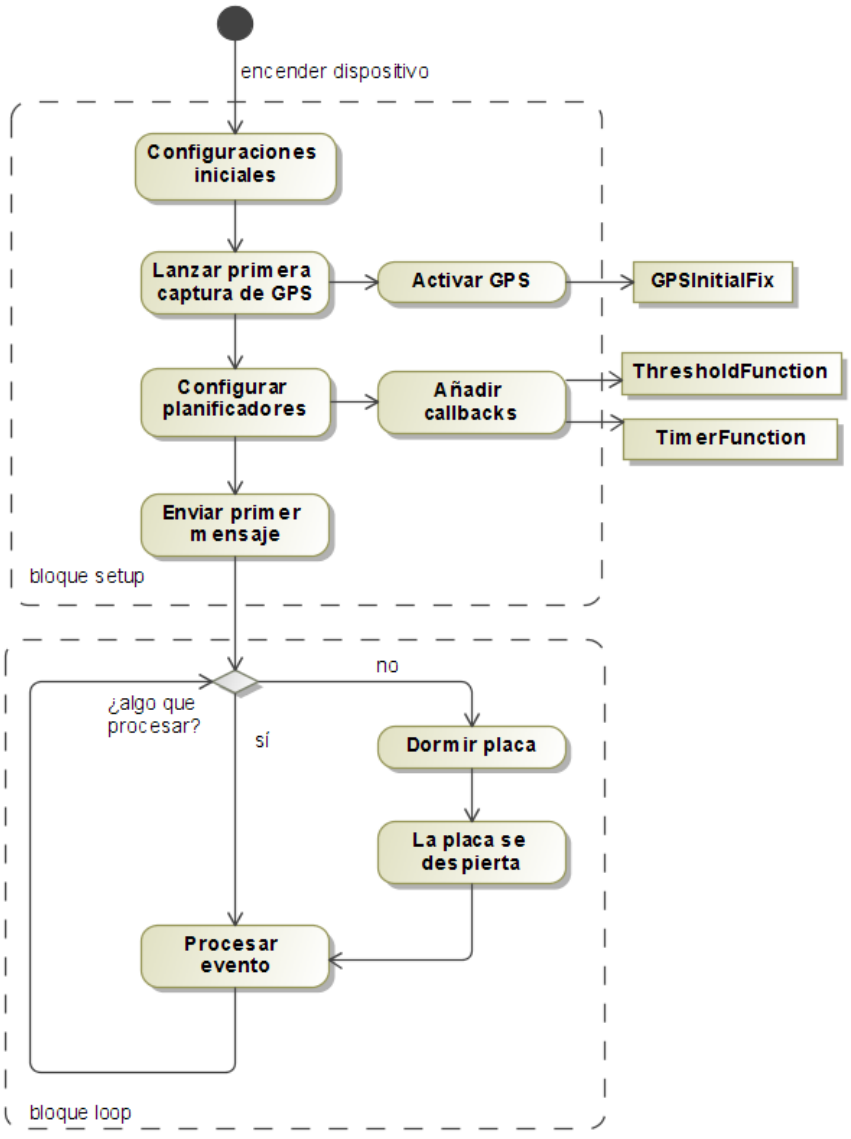


Figura 5-5. Diagrama de actividad para las funciones setup y loop.

Conviene mencionar algunos detalles de este proceso, para su mejor comprensión:

- Las **configuraciones iniciales** contemplan la activación del **watchdog**, la activación de los pines utilizados para encender y apagar el **LED** del dispositivo, y la **inicialización del GPS y acelerómetro** para poder procesarlos posteriormente cuando se activen.
- Como se mencionó en las modificaciones de la especificación, se lanzará **la captura de la primera posición** a través del **GPS**. Se utilizará la función **GPSInitialFix** para este propósito.
- Tras iniciar el GPS, tocará activar los **planificadores** para las alarmas de **temperatura y batería**, que tendrán la función de callback **ThresholdFunction**; y también para **RTC**, cuya función de callback será **TimerFunction**. Estas funciones se llamarán periódicamente cada vez que transcurra un cierto intervalo de tiempo, que se pasa como parámetro al planificador.
- Para cerrar el bloque setup, se enviará el **primer mensaje** al backend para notificar el inicio de actividad.
- El bloque **loop** se simplifica: la placa estará **dormida**, salvo que tenga algún **evento por procesar**. Estos pueden ser:
    - **Procesamiento de GPS**, mientras esté activo; que será en la captura de la primera posición (en el setup, que llama a GPSInitialFix), y cuando se detecte movimiento (función MoveDetected, que llama a la función GPSFix; ambas se comentarán posteriormente). Se utiliza la función **TD_GEOLOC_Process**, de las librerías del SDK.
    - **Procesamiento de acelerómetro**, una vez activo. En la aplicación, el acelerómetro no se activará hasta que se capture la primera posición, que se detecta en GPSInitialFix. Se utiliza la función **TD_ACCELERO_Process**, de las librerías del SDK.
    - **Procesamiento de eventos del planificador;** ya sea por alarma de temperatura/batería o por RTC.

Aunque se contempla en el diagrama el caso, se puede afirmar que la placa **no estará dormida nunca**. Esto se debe a que, de inicio, se activa el GPS. Cuando se encuentra la primera posición, se desactiva, pero pasa a monitorizarse el acelerómetro de forma continua. Luego; al menos, la función loop **procesará siempre el acelerómetro** cada ciclo de reloj, en busca de movimiento. Como se comentó anteriormente, esto **no desemboca en problemas de consumo**, ya que el consumo del acelerómetro en reposo es prácticamente nulo.

5.3.2. Capturando la primera posición con GPSInitialFix

En la función setup, llamamos a la función **GPSInitialFix** para **capturar la primera posición conocida** del dispositivo. Además; como se adelantó en la actualización de las especificaciones, los casos de **movimiento y cambio de posición** (que van ligados e involucran al acelerómetro, en primera instancia) no se monitorizarán hasta que se consiga esa posición.

Este comportamiento se puede resumir en el siguiente **diagrama de actividad**:

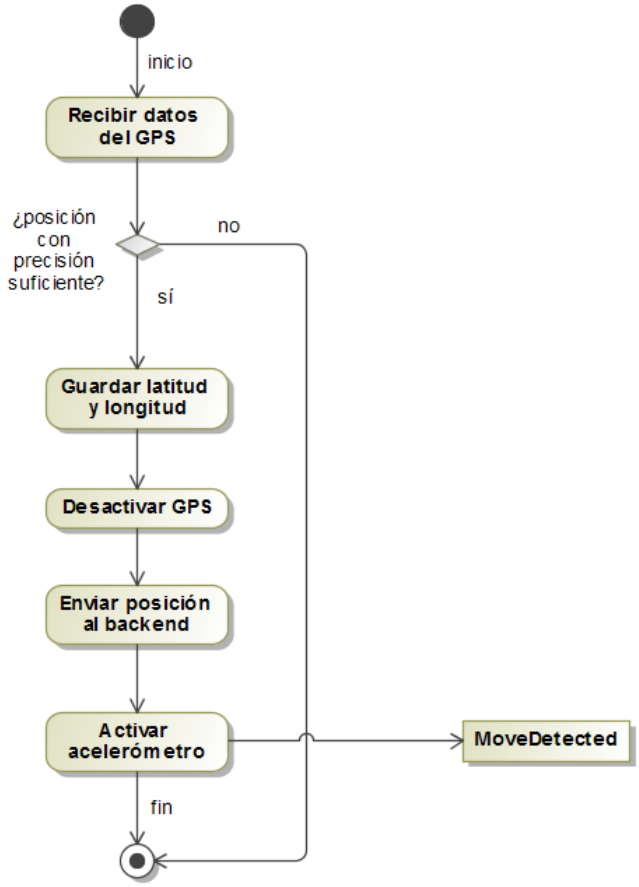


Figura 5-6. Diagrama de actividad para la función GPSInitialFix.

Se comentan, a continuación, algunas particularidades a destacar:

- Recordar que esta primera captura **no tiene temporizador** asociado; de manera que seguirá buscando una posición con **precisión suficiente** (que tenga datos de latitud, longitud y altura) hasta que la encuentre.
- Los datos se reciben con cada iteración de TD_GEOLOC_Process en la función loop. Por ello, de no tener la calidad suficiente, bastará con no hacer nada más, y esperar a recibir los nuevos datos.
- La **latitud y longitud** deberán ser **convertidas** convenientemente para almacenarse en variables **enteras sin signo**. Será el servidor que procese los datos el que se encargue de deshacer esa conversión para obtener el valor en grados y minutos. Además, se **almacenarán** en el dispositivo, para tenerlas disponibles para futuras comparaciones con nuevas posiciones detectadas.
- Tras **desactivar el GPS**, la función **TD_GEOLOC_Process no será llamada** en el loop hasta que se vuelva a activar.
- Una vez **activado el acelerómetro**, la función **TD_ACCELERO_Process se llamará** en el loop continuamente, cada ciclo de reloj, y generará una interrupción en caso de detectar movimiento. Cuando eso ocurra, se llamará a la función **MoveDetected**. Notar que la monitorización del acelerómetro nunca parará, a partir de este momento.

5.3.3. Detectando movimiento con MoveDetected

Esta función se ocupa de gestionar el caso de uso de **detección de movimiento**, una vez se obtiene la primera posición del dispositivo. Su funcionamiento se puede explicar a partir del siguiente **diagrama**:

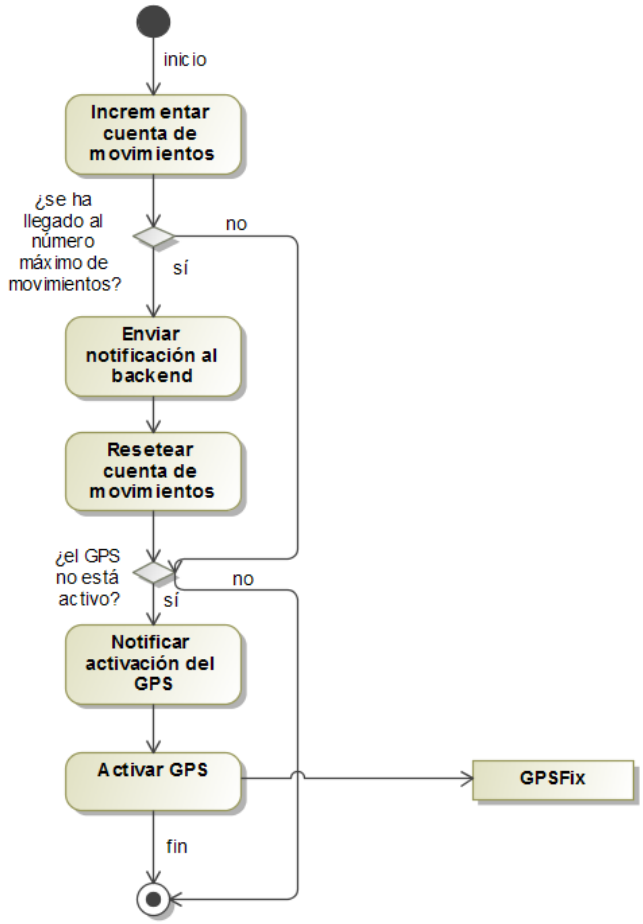


Figura 5-7. Diagrama de actividad para la función MoveDetected.

Vemos que, en este caso, también se han incorporado algunas de las **modificaciones** propuestas:

- La **cuenta de movimientos** nos permitirá enviar la **alarma** cuando se llegue a un **número máximo de movimientos**. De no alcanzarse, se acumularía en la cuenta y no se enviaría nada.
- **No** se puede lanzar una **captura GPS** si hay algún **proceso de geolocalización activo**. Por ello, se comprueba si el GPS está activo gracias a una variable **bandera** (fixing_enabled); un booleano que valdrá true si el GPS está desactivado, o false si hay una captura activa.
- En el caso de iniciar una **nueva captura**, **notificaremos la activación** del GPS **actualizando** el valor de la variable **bandera** a false. De esta forma; de detectarse movimiento en la placa, no se volvería a lanzar una captura GPS hasta que termine la que está activa, que se gestiona desde la función **GPSFix**.

5.3.4. Buscando un cambio de posición con GPSFix

A esta función llegarán los datos del GPS referente a una **nueva posición**, que se busca debido a la detección de movimiento en la placa.

Mostramos la lógica que envuelve a esta función con el **diagrama de actividad** siguiente:

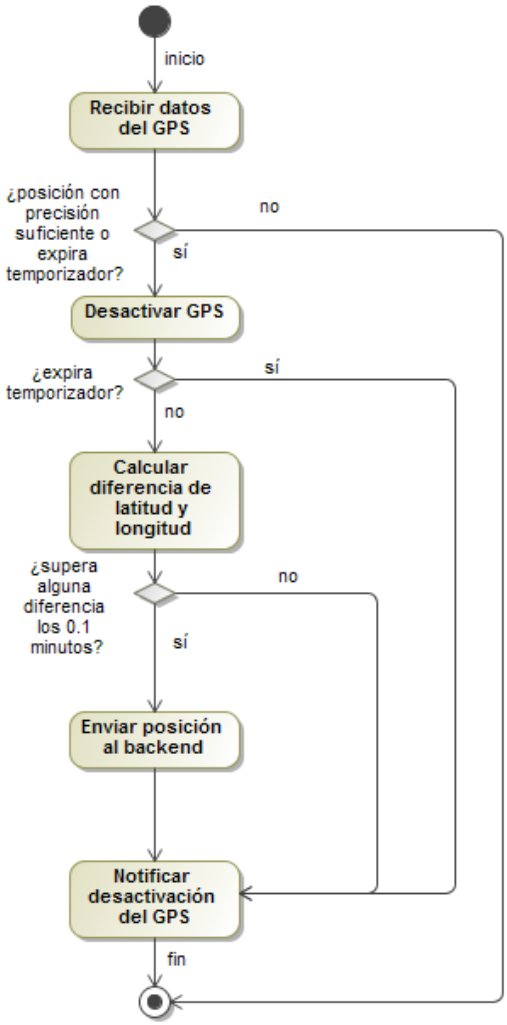


Figura 5-8. Diagrama de actividad para la función GPSFix.

Aunque parece similar a la función GPSInitialFix, sí que presenta **diferencias** en su comportamiento:

- La captura de esta posición tiene un **temporizador** asociado, ya que podría ocurrir que dicha posición necesitase mucho tiempo para obtenerse, o directamente que nunca fuese capturada. Esto provocaría un consumo mayor, por tener activo el GPS. La **limitación del consumo** en esos casos justifica el uso del temporizador.

  Como, en condiciones normales, tardará 2-3 minutos en obtener la posición, se fija el valor del temporizador en 10 minutos.

- Notar que, en esta función, se para el GPS, y luego se obtienen los datos. En GPSInitialFix, se hacía al revés. Se hace así para demostrar que **el orden de estas operaciones no influye en el resultado final**, puesto que los datos de geolocalización se reciben como parámetros de entrada de la función, no influyendo el estado del GPS para la obtención de los datos.

- Tanto si se consigue la posición con la precisión adecuada como si vence el temporizador, se debe **notificar** con la variable **bandera** fixing_enable que el GPS se ha desactivado, para que se pueda habilitar **una nueva captura desde la función MoveDetected**.

- Como se justificó anteriormente, se necesita **superar los 0.1 minutos** de diferencia de latitud o longitud para notificar un **cambio de posición**.

- Esa nueva posición **no se almacenará** en el dispositivo; solamente se enviará al backend para alertar

del cambio de posición. La única posición que **debe almacenarse** en el dispositivo es **la inicial**, para compararla con las nuevas posiciones detectadas.

5.3.5. Control de umbrales con ThresholdFunction

Vamos a comentar el resto de casos de uso que nos restan por monitorizar. En primer lugar, hablaremos de las **alarmas de temperatura y batería**, que se lanzarán cuando se detecte un **cambio de estado** en alguna de esas dos magnitudes.

Dichos estados aparecen reflejados en la figura 5-2, donde hablaremos de temperatura baja, correcta o alta, y de batería baja o correcta. Todas estas zonas estarán separadas por **umbrales**, formados por un intervalo de valores, y no por un valor concreto, como se justificó en su momento con la inclusión del concepto de histéresis.

Si bien la placa es capaz de obtener los valores de temperatura y batería a través de las funciones incluidas en las librerías del SDK, necesita la configuración de un **planificador** para hacer la supervisión cada cierto intervalo de tiempo. La función de callback llamada seguiría la siguiente **lógica**:

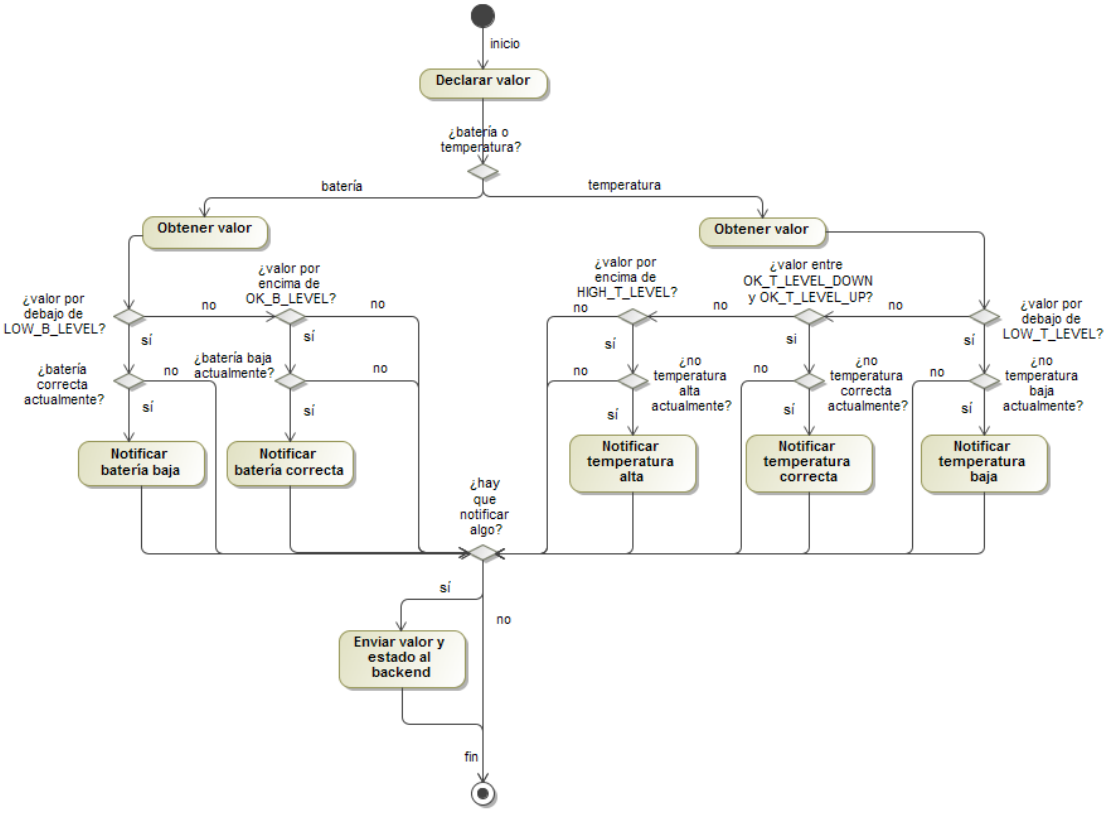


Figura 5-9. Diagrama de actividad para la función ThresholdFunction.

Sobre este procedimiento de monitorización de estados, cabe destacar los siguientes **detalles**:

- Esta función de callback **es llamada para los dos casos** a monitorizar. Esto se consigue con un **valor adicional** que se le pasa al planificador. Cuando se llama a esta función, se recibirá como parámetro ese valor, que permitirá **diferenciar** si la llamada es para monitorizar temperatura o batería.
- Podemos ver que, en primer lugar, se **declara** la variable que guardará el valor a tratar, y después se almacena en ella el valor concreto en función del caso a monitorizar. Se hace así porque se utiliza **la misma función**, en las librerías del SDK, para obtener el valor de temperatura/batería actual, pero pasándoles **parámetros distintos** (true si es temperatura, false si es batería).

  Además, se hace por motivos de **visibilidad de variables** del lenguaje C. El valor se lee dentro de un

bloque if-else, y el envío al backend (que utiliza ese valor) se realiza en otro bloque if. De no declararse al principio de la función, la visibilidad del valor se restringiría al primer bloque, y no se podría utilizar para el envío.

- Se deben **programar y almacenar en memoria los estados y umbrales** que se quieren considerar. La placa es capaz de medir el valor de temperatura y batería, pero no de asociarlos directamente a una zona de funcionamiento.

  En relación a esto, la función **se llamará siempre**, pasado el intervalo de tiempo configurado en la llamada al planificador en la función setup, e independientemente de si hay un cambio de estado o no en alguna de las magnitudes.

- Para darse un **cambio de estado**; además de estar en ese estado actualmente, el **estado anterior debe ser diferente** al actual. De esta forma, sólo **se notifica una vez** la entrada en un estado. Como consecuencia, **se debe almacenar el estado anterior** en la memoria del dispositivo.

- Vemos que, ambos casos, desembocan en la comprobación de si hay algo que **notificar** al backend. Para realizar esto, se utilizará un booleano, que cambiará su valor a true si se dan las condiciones de cambio de estado.

  Como cada llamada a esta función corresponderá unívocamente a uno de los casos de uso (en función del parámetro recibido por la función para discriminar entre ambos casos), no ocurrirá que el valor del booleano se vea modificado en otro punto distinto de los programados en el diagrama.

- La notificación al backend contendrá el **caso** monitorizado (temperatura o batería), el **estado** alcanzado, y el **valor** con el que se ha alcanzado.

5.3.6. Alarma de RTC con TimerFunction

Por último, hablaremos de la monitorización de los valores de temperatura y batería vía RTC, que está asociado al **reloj interno** del dispositivo.

El motivo de su uso es simple: podría ocurrir que **no se diesen frecuentemente cambios de estado** en temperatura o batería, lo cual implicaría que no podríamos conocer información precisa acerca de los valores concretos en los que se movería el funcionamiento del dispositivo.

Por ello; **cada cierto tiempo**, se **notificarán** al backend los valores de temperatura y batería en ese instante. Esto no deja de ser un **planificador**, que enviará datos siempre, en cada llamada a esta función.

Esto nos deja un **diagrama de actividad** bastante simple en funcionalidad:

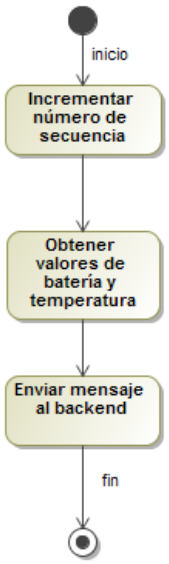


Figura 5-10. Diagrama de actividad para la función TimerFunction.

El único detalle a destacar es el uso de un **número de secuencia** en el mensaje. Se incluye para que, en recepción, se puedan **detectar pérdidas** en estos mensajes en caso de producirse, ya que este valor se incrementará con cada envío. También podría significar **el comienzo de una nueva monitorización**, si el nuevo número de secuencia recibido fuese 1 de nuevo.

5.4. Resultado final

Como último paso a seguir para llegar a la solución final, restaría por **codificar el diseño** planteado en el apartado anterior, obteniendo el **código** que responda a esa solución propuesta.

Si bien la **codificación** entra dentro de una de las fases de la metodología seguida, esta sección no pretende mostrar el código fuente de la aplicación sin más, sino incidir en diversos detalles del planteamiento de la solución final y en los motivos que han conducido a elegir las funciones y las sentencias implementadas finalmente, como hemos podido comprobar en los apartados anteriores.

Para más información en cuanto al código se refiere, remitimos al **Anexo C**, que contiene la documentación completa del código realizado, que se presenta como **resultado final** en la parte de codificación de la aplicación que respondiese a los casos de uso antes mencionados; y actualizados, en base a los problemas encontrados durante el estudio de la placa de evaluación y sus librerías.

Pero, antes de finalizar con la sección, sí que convendría hablar de **algunos detalles de programación** de la aplicación, que son los siguientes:

5.4.1. Codificación de los mensajes

Se pretende mostrar cómo se **codifican los mensajes** para ser enviados al backend. Recordamos que SigFox permite enviar una carga útil de 12 bytes como máximo, codificado según la elección adoptada por el fabricante del dispositivo homologado.

En el SDK, un mensaje se codifica con una **matriz de unsigned char**, con la longitud definida por el programador, pero sabiendo que sólo podremos enviar, como máximo, los 12 primeros valores almacenados.

Los mensajes considerados en esta aplicación utilizarán **tanto caracteres como números**, que se codificarán de la siguiente forma:

- **Caracteres**: escribiremos los caracteres **literalmente** en la matriz, entre comillas, tratándolos como unsigned char. A la hora de representarlos en el backend, se traducirán en base a la **recomendación T-50** de la ITU [32].
- **Números**: ya que las cifras que almacenan valores de temperatura, batería o posición se almacenan en valores unsigned int (4 bytes); a la hora de guardarlos en la matriz de unsigned char, se irán guardando byte a byte, utilizando los **operadores de desplazamiento de bits** que incorpora el lenguaje C. En recepción, se mostrarán en **hexadecimal**.

De esta forma, la placa podrá enviar los siguientes **mensajes**, en función del caso que se esté dando en ese instante:

- **Primer mensaje:** se envía al acabar de configurar la placa en el bloque setup. Envía la cadena “It works” al backend.
- **Primera posición vía GPS:** este mensaje se envía desde la función GPSInitialFix cuando se consigue la primera posición del dispositivo con la suficiente precisión. Se enviará el carácter ‘G’ (de geolocalización), seguido de la latitud y su dirección (‘N’ si es norte, ‘S’ si es sur), y de la longitud y su dirección (‘E’ si es este, ‘W’ si es oeste). Los valores de latitud y longitud se enviarán como un número entero, sabiendo que las 7 últimas cifras serán los minutos (con 5 dígitos decimales), y las anteriores serán los grados, sin decimales.
- **Movimiento:** enviado cuando se alcanza el número máximo de movimientos permitidos antes de lanzar la notificación. Envía la cadena “Movement” al backend.

- **Cambio de posición:** sólo se enviará este mensaje si se detecta un cambio de posición del dispositivo, tras haber obtenido una diferencia de 0.1 minutos en latitud o longitud. Su formato es el mismo que el de la primera posición de GPS, pero cambiando la 'G' inicial por la 'A' (de acelerómetro).
- **Alerta de cambio de estado de temperatura o batería:** se enviará este mensaje cuando haya que notificar un cambio de estado tanto de temperatura como de batería. Para ello; se enviará, en primer lugar, un carácter que indique la alarma producida ('B' si es batería, 'T' si es temperatura), seguido de un valor, de 1 byte, que represente el estado alcanzado. Finalmente, se incluirá el valor que produce la alarma, en 4 bytes
- **Alarma de RTC:** mensaje enviado cada vez que pasa el intervalo programado para el RTC. Se comenzará con el carácter 'R' (de RTC), seguido del número de secuencia almacenado en 3 bytes (que permite que no se repita su valor hasta pasados más de 300 años, contando que tenemos 140 mensajes como máximo al día). Finalmente, se añadirán los valores de temperatura y batería leídos, cada uno almacenados en 4 bytes. Notar que es el único mensaje que utiliza el máximo de 12 bytes permitidos de carga útil.

En la siguiente tabla, se pueden ver estos mensajes y su contenido byte a byte.

Mensaje enviado	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Primer mensaje	'I'	't'	' '	'w'	'o'		'r'	'k'	's'			
Primera posición	'G'	latitud					'N/S'	longitud			'E/W'	
Movimiento	'M'	'o'	'v'	'e'	'm'		'e'	'n'	't'			
Cambio de posición	'A'	latitud					'N/S'	longitud			'E/W'	
Alerta de temperatura	'T'	estado	valor									
Alerta de batería	'B'	estado	valor									
Alarma de RTC	'R'	número de secuencia					valor temperatura				valor batería	

Tabla 5–1. Mensajes enviados al backend, con su contenido byte a byte.

Notar que, cada mensaje, **comienza por un carácter distinto**. Esto no ha sido elegido al azar, sino para **diferenciarlos** en el servidor de procesamiento de datos. En función del primer carácter leído, se procesará el mensaje de una forma u otra, en función del contenido que siga a ese primer carácter.

5.4.2. Uso de variables globales

Un aspecto clave a tener en cuenta es el uso de **variables globales** en el código propuesto como solución. Su utilización, aunque rompe con la programación estructurada, resulta de mucha utilidad en la programación de dispositivos electrónicos, debido a que **reservamos un espacio de memoria** dedicado a estas variables, de interés general.

Se han considerado las siguientes **variables** como globales:

- El **instante actual** en el que ocurre cualquier evento, para pruebas de la aplicación.
- El **número de secuencia** utilizado en la alarma de RTC, para tener constancia de cuál es el siguiente valor a utilizar.
- La **latitud y longitud**, con sus direcciones, de la **primera posición** conocida. Recordamos que se necesita esta posición para compararla con las nuevas posiciones capturadas.

- El booleano **fixing_enabled**, para notificar el estado del GPS y así evitar lanzar más de una captura cuando se detecte movimiento en la placa.
- El **número de movimientos detectados** por el acelerómetro, para compararlo con el máximo valor de movimientos permitidos y lanzar la alarma en caso de alcanzarlo.
- Los últimos valores de **temperatura y batería** obtenidos por la alarma de RTC. Como se leerán siempre cada vez que transcurra el intervalo del RTC, conviene ir actualizando el valor capturado en estas variables, y no generar nuevas variables por cada llamada a TimerFunction.
- Los **estados actuales de temperatura y batería**, para compararlos con el nuevo estado alcanzado.
- La matriz de unsigned char que contendrá el **mensaje a enviar**, ya que se utiliza por varias funciones, y tampoco conviene generar una nueva matriz por cada mensaje enviado.

5.4.3. Elección de temporizadores para los planificadores

Además de la codificación de los mensajes enviados al backend, también entra en juego la previsión de **cuántos mensajes** se enviarán al backend sin sobrepasar el límite de los 140 mensajes al día.

Para ello, uno de los parámetros con el que se puede jugar es el **temporizador de los planificadores utilizados**; en referencia a los casos de temperatura, batería y RTC. Los valores fijados son los siguientes:

- **30 minutos** para las alertas de batería y temperatura.
- **1 hora** para la alarma de **RTC**.

Con estos valores, cabe la posibilidad de que se notifique una alerta de batería y/o temperatura junto a una alarma de RTC. En este caso, se llamará al **evento más antiguo** en la cola de eventos. En recepción, se notará un **retraso** de unos segundos (6-7 segundos, los que tarda la placa en enviar un mensaje al backend) entre cada mensaje, pudiendo **variar levemente los valores** leídos por esa diferencia de tiempo, aunque se acepta ese pequeño error en la codificación final.

Finalmente; a partir de estos datos, obtenemos la siguiente **previsión de mensajes**, como máximo (sin considerar detección de movimiento y cambio de posición).

Mensaje enviado	Cantidad
Primer mensaje	1 mensaje
Primera posición	1 mensaje
Alerta de temperatura	Cada 30 minutos, 2 cada hora: 2*24 = 48 mensajes
Alerta de batería	Cada 30 minutos, 2 cada hora: 2*24 = 48 mensajes
Alarma de RTC	Cada hora: 24 mensajes
Total	**122 mensajes**

Tabla 5–2. Previsión de mensajes enviados, al día, de cada tipo (sin contar detección de movimiento y cambio de posición).

Esto nos deja un total de **122 mensajes**, como máximo, al día. Los **18 mensajes** restantes (como mínimo), se destinarían a cubrir **los dos casos restantes**; los cuales, como justificamos, son menos probables de producirse, salvo intervención de agentes externos, robos u otros motivos que provoquen el desplazamiento del dispositivo de un punto a otro más distante.

6 PRUEBAS DE LA APLICACIÓN Y VERIFICACIÓN DE RESULTADOS

"Los malos programadores tienen todas las respuestas. Los buenos testers tienen todas las preguntas"

- Gil Zilberfeld, autor de "Everyday Unit Testing" -

Como paso final para concluir con el desarrollo del software requerido para la placa de evaluación, se debe considerar la realización de **pruebas** sobre la aplicación final, verificando así que su comportamiento es el correcto, y que todos los casos de uso se notifican correctamente cada vez que se producen.

Como se ha ido adelantando a lo largo del desarrollo de esta memoria, el objetivo final de la aplicación es lograr que los **mensajes lleguen al backend**, se **redirijan a un centro de procesamiento**, y se **traten** convenientemente para la **generación de estadísticas finales**. Cada una de estas partes debe verificarse una a una, de manera que el resultado final sea el esperado.

Este **proceso** se reflejará en esta sección. En primer lugar, se introducirán las **plataformas** utilizadas para la prueba de la aplicación, mostrando con detalle el papel que jugarán a la hora de realizar las pruebas. A continuación, definiremos las **pruebas** a realizar; desde los **pasos** que se seguirán para la supervisión de la aplicación, hasta los **resultados esperados**. Por último, se concluirá con el **resultado final** de las pruebas, verificando que coinciden con los que se esperaban en la definición de las pruebas.

6.1. Plataformas utilizadas

Las herramientas empleadas en este proceso de pruebas ya han sido introducidas en secciones anteriores, en mayor o menor medida. En este apartado, se hará hincapié en los **aspectos más relevantes** de cada una a la hora de emplearlas para la realización de las pruebas y la verificación de los resultados.

Presentamos, a continuación, las **plataformas** consideradas.

6.1.1. Backend de SigFox para recepción de mensajes

El **backend de SigFox** [2] ya fue presentado en la sección de SigFox, pero conviene recordar algunas **funcionalidades** de utilidad que servirán para el proceso de pruebas que se planteará a continuación.

Destacaremos las dos que se utilizarán a lo largo de esta sección, y que pasamos a mencionar con más detalle.

6.1.1.1. Receptor de mensajes

El **receptor de mensajes** enviados por cada dispositivo será fundamental para saber si los mensajes enviados llegaron correctamente al backend. En la figura 3-5 se mostró la recepción de un mensaje cualquiera, con todos los campos característicos. En concreto, atenderemos en mayor medida al **mensaje recibido** y a su **traducción en ASCII** (si es un mensaje en el que sólo se envían caracteres), para verificar que se está recibiendo la información correctamente codificada en función del caso de uso producido.

La pestaña de mensajes está accesible a través de **Device (barra superior) > ID del dispositivo > Messages (barra lateral)**.

6.1.1.2. Uso de URLs de callback

La segunda utilidad a considerar será la **configuración de la URL de callback** a la que reenviar los mensajes recibidos por el backend. Esta opción se puede configurar en la sección Callbacks, accesible vía **Device Type (barra superior) > Nombre del tipo de dispositivo > Callbacks (barra lateral)**. De esta forma, vemos que los callbacks **se configuran por grupos de dispositivos**, para tratar a todos de la misma manera.

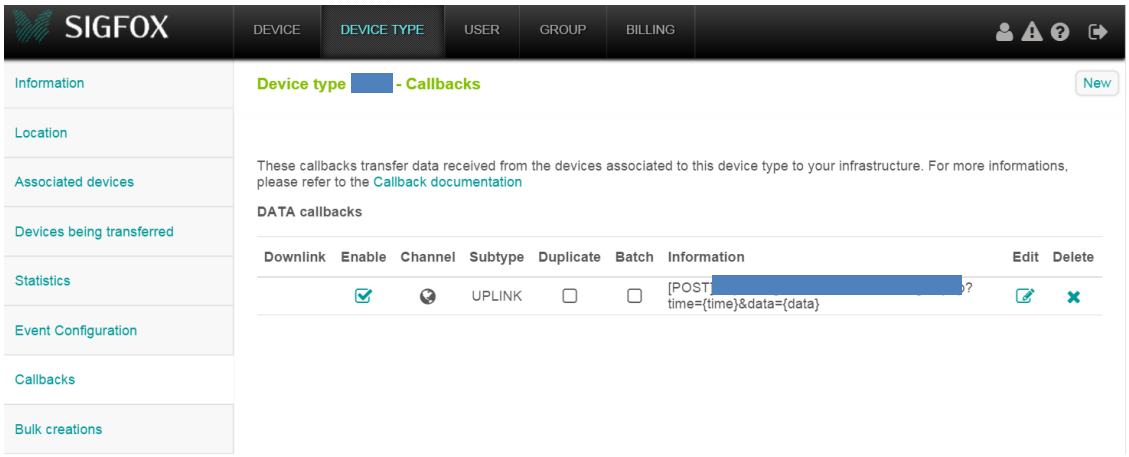


Figura 6-1. Sección de configuración de callbacks para la redirección de mensajes. *Backend de SigFox.*

Para añadir una nueva URL de callback, bastará con pulsar en New (esquina superior derecha), y completar las siguientes **propiedades**:

- **Type:** el **tipo de mensaje** que se puede enviar, distinguiendo entre **DATA** (carga útil), **SERVICE** (mensajes de operaciones realizadas por el dispositivo) y **ERROR** (tramas enviadas cuando se produce la pérdida de la comunicación con la placa). En nuestro caso, seleccionaremos la opción **DATA**. El segundo parámetro, en este caso, será el **sentido de la comunicación**, donde sólo se puede seleccionar **UPLINK** (sentido ascendente, del dispositivo al backend). De adquirir la suscripción para la comunicación bidireccional, también se podría elegir la opción **BIDIR** (bidireccional).

  En el caso de elegir **SERVICE**; como segundo parámetro, se puede elegir entre **STATUS** (que envía los niveles de temperatura y batería del dispositivo) y **ACKNOWLEDGE** (sólo disponible si se dispone de una suscripción de comunicación bidireccional; mostraría información acerca de la conexión de bajada). Para **ERROR**, no hay ninguna opción a elegir para ese segundo parámetro.

- **Channel:** permite elegir el **tipo de recurso** que se utilizará para identificar al callback, pudiendo elegir entre **URL** (nuestra opción; enviará cada mensaje en una respuesta HTTP independiente), **BATCH_URL** (que permite agrupar mensajes en una misma respuesta) y **EMAIL** (para enviar los mensajes a un correo electrónico).

- **Send duplícate:** opción que, de ser marcada, habilitará el **envío de mensajes por duplicado**, como prevención para casos de pérdida de información.

En el caso de seleccionar **URL**, se presentarán estos campos:

- **Url pattern:** aquí se deberá añadir la **dirección de la aplicación web** que va a recibir los datos, seguido de los **atributos** que se desean recibir del mensaje original. En nuestro caso, se ha elegido la **fecha de recepción** (time) y la **carga útil** (data).
- **Use HTTP POST:** marcando esta casilla, se enviará la respuesta utilizando el método POST. En otro caso, se utilizaría el método GET. La documentación del backend de SigFox recomienda el uso de **POST**, luego será nuestra opción seleccionada.

Figura 6-2. Añadiendo una URL de callback. *Backend de SigFox.*

De optar por alguna de las **otras** dos **opciones**, se deberán considerar los siguientes **detalles**:

- **URL_BATCH:** sólo se puede utilizar **HTTP POST**. Se dispondrá de la propiedad **Url pattern** para añadir la dirección de la aplicación web para los grupos de mensajes, y después otra propiedad llamada **Line pattern**, que permitirá elegir los atributos que se desean recibir.
- **EMAIL:** se darán los campos **Recipient** para añadir el correo al que se desea enviar los mensajes, **Subject** para personalizar el título del correo, y **Message** para escribir el contenido del mensaje, eligiendo los atributos que se quieran recibir.

Una vez seguidos estos pasos, aparecerá el callback configurado en la pestaña de Callbacks. Allí, aparte de editar sus propiedades o eliminarlo, podremos **activarlo o desactivarlo** desde la casilla **Enable**.

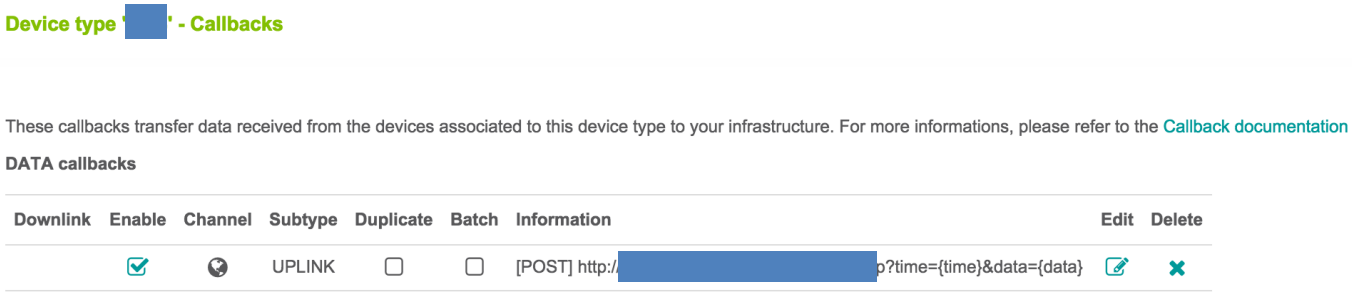


Figura 6-3. Pestaña Callbacks, mostrando los callbacks configurados. *Backend de SigFox.*

De estar habilitado, los mensajes recibidos a partir de entonces serán **reenviados** a la URL configurada anteriormente. Es más; los mensajes se reenviarán a todos los callbacks configurados para ese tipo de dispositivos, aunque sean de distinto tipo (URL, BATCH_URL o EMAIL).

Sabremos que un mensaje se ha reenviado correctamente si, en el receptor de mensajes, aparece el **icono** de Callbacks con color **verde**.

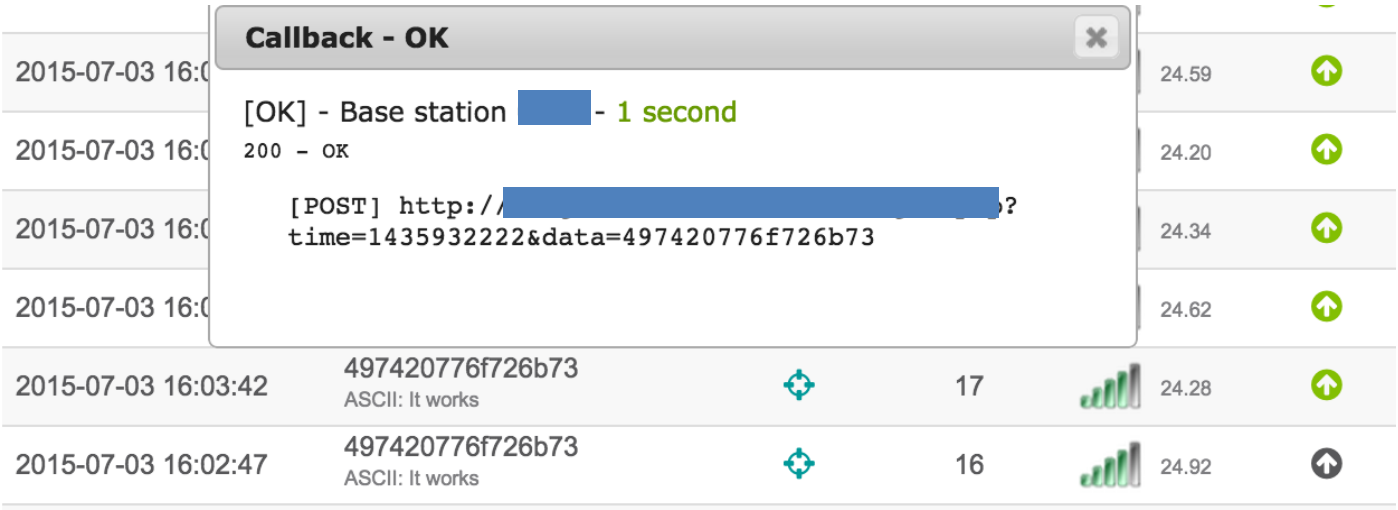


Figura 6-4. Comprobando el reenvío del mensaje a la URL de callback. *Backend de SigFox.*

También puede ocurrir que ese icono esté de color **gris**. En ese caso, no se está reenviando el mensaje porque no hay un callback configurado (como mínimo). La última opción sería presentarse de color **rojo**; que ocurre cuando se ha configurado un callback, pero la respuesta no llegó al producirse un error en el reenvío.

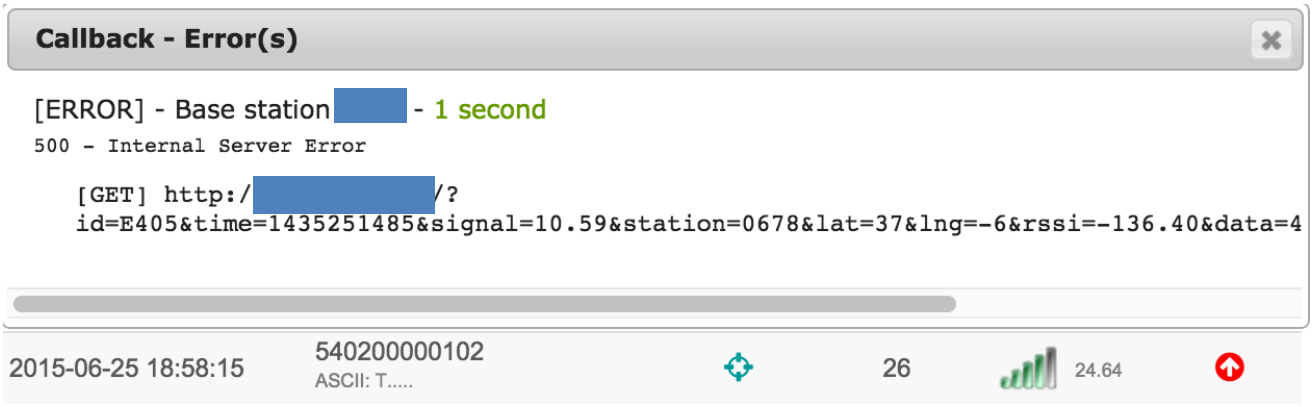


Figura 6-5. Mensaje con error en el reenvío. *Backend de SigFox.*

6.1.2. Servidor LAMP para generación de estadísticas

Hemos hablado de la redirección de mensajes a través de la opción Callbacks que ofrece el backend de SigFox, pero todavía no hemos profundizado en la herramienta utilizada para **procesar** esos mensajes reenviados para decodificar la carga útil y **representarlos** convenientemente según el caso de uso producido.

Para este proyecto, se ha optado por el uso de un **servidor LAMP**, alojado en una máquina Debian 8.1, y accesible por SSH a través de cualquier equipo, con tal de conocer la IP pública que lo identifica y el usuario con el que acceder.

Esta memoria no pretende enseñar el proceso de construcción del servidor, sino considerar que ya está configurado convenientemente y preparado para recibir los datos. En la siguiente página [39], se deja un manual de instalación básico de este tipo de servidor en una máquina Debian 8/8.1. A partir de entonces, se siguieron los siguientes **pasos**, obteniendo información extra y resolviendo los problemas surgidos durante el proceso con ayuda del recurso [40]:

- Creación de la **base de datos y las tablas** que reciban los datos decodificados de los mensajes. Para trabajar con un entorno más cómodo, se instaló y utilizó la utilidad **phpMyAdmin.**

- Creación de un **usuario específico** en la base de datos, que sólo tenga acceso a esas tablas previamente creadas, como medida de seguridad para no comprometer el resto de la información almacenada.
- Concesión de los **permisos** adecuados para permitir el acceso a la base de datos por la IP pública; ya que, por defecto, la configuración de MySQL sólo permite trabajar en local.
- Desarrollo de la **aplicación web** (alojando los ficheros en el directorio /var/www/html) que reciba los datos, los procese, los almacene en la base de datos y, finalmente, los recupere para mostrarlos adecuadamente según el caso de uso producido. Su codificación detallada se muestra en el **Anexo D**.
- **Pruebas aisladas** de la aplicación, verificando que se reciben los datos correctamente y se almacenan en la tabla de la base de datos que le corresponde, para posteriormente representarlo adecuadamente.
- **Configuración de la URL de callback,** tal y como se indicó en la **sección 6.1.1.2.**
- **Prueba de la aplicación** con el **dispositivo** en funcionamiento. Este paso se mostrará en los siguientes apartados.

6.1.2.1. Tablas utilizadas en la base de datos

En la base de datos de la aplicación web, se ha considerado el uso de **3 tablas distintas**, cada una destinada a un grupo de notificaciones de casos de uso concreto:

- **Tabla graphics:** se ocupará de almacenar los valores de temperatura y batería enviados por la alarma de **RTC**, para mostrarlos en dos gráficas separadas.

Nombre	Tipo	Descripción
id	int(11)	Clave primaria. Número de secuencia incluido en la carga útil de la alarma de RTC.
date	datetime	Clave primaria. Fecha de recepción en el backend de la alarma, indicada en el parámetro time que se recibe en la respuesta HTTP POST.
temperature	double	Valor de temperatura leído por la placa, en grados centígrados.
battery	double	Valor de batería leído por la placa, en voltios.

Tabla 6–1. Tabla graphics, con sus atributos (nombre, tipo y descripción).

- **Tabla alarms:** en esta tabla, se recogerán todas las alarmas enviadas por el dispositivo, referentes al **primer mensaje** enviado, detección de **movimiento**, y las alarmas de **temperatura y batería**. Cada una se mostrará en un párrafo separado en la aplicación, junto a la fecha de ocurrencia.

Nombre	Tipo	Descripción
date	datetime	Clave primaria. Ídem que en la tabla graphics.
message	varchar(100)	Mensaje que describe la alarma. Su composición forma parte de la lógica de la aplicación de recepción de mensajes procedentes del backend.

Tabla 6–2. Tabla alarms, con sus atributos (nombre, tipo y descripción).

- **Tabla maps:** recogerá las posiciones detectadas por el GPS de la placa, tanto en los casos de **primera posición conocida** como el de **cambio de posición**. Se mostrarán en un mapa, utilizando la API de Google Maps, con un título que mostrará el orden (secuencia) y la fecha de captura.

Nombre	Tipo	Descripción
date	datetime	Clave primaria. Ídem que en la tabla graphics y alarms.
latitude	double	Latitud de la posición capturada, en formato decimal.
longitude	double	Longitud de la posición capturada, en formato decimal.

Tabla 6–3. Tabla maps, con sus atributos (nombre, tipo y descripción).

Notar que, como la placa de evaluación funciona por cola de eventos, nunca ocurrirá que se reciban dos mensajes al mismo tiempo en el backend. Este detalle justifica el uso de la **fecha de recepción** como **clave primaria** en las tablas.

6.1.2.2. Representación de estadísticas

Tras recibir los datos y almacenarlos en la base de datos, se deberán **representar** convenientemente, a modo de **estadísticas de uso** de la placa de evaluación, de manera que resulte práctico y fácil de entender.

Por ello, se aprovechará la división en 3 tablas, detallada anteriormente, para mostrar **3 secciones** en la aplicación web de representación de estadísticas:

- **Gráficas** con los niveles de temperatura y batería (por separado), obtenidos de la tabla graphics. Se indicará también la fecha en la que se obtuvo cada dato, y si hay saltos en el número de secuencia.
- **Sucesión de párrafos** con los **mensajes de alarma** almacenados en la tabla alarms.
- **Mapa,** de la API de Google Maps, para mostrar las **posiciones capturadas por el GPS.** Junto al marcador, se indicará el orden de captura de la posición, y la fecha de captura. No se añade información acerca del caso de uso producido; pero, si el dispositivo se mantiene con batería, se diferenciarán porque la primera posición detectada (número 1) siempre será la primera posición conocida; y el resto, eventos de cambio de posición.

  En cambio, de apagar y volver a encender la placa, no se conocería, a priori, qué posición corresponde a primera posición conocida y cuál a cambio de posición. Sin embargo, se podría deducir a partir de la fecha de ocurrencia, comparándola con la fecha del primer mensaje enviado, que sí se notifica explícitamente.

El resultado final de la aplicación web, sin contenido en la base de datos, sería el siguiente; distinguiendo las 3 secciones anteriores:

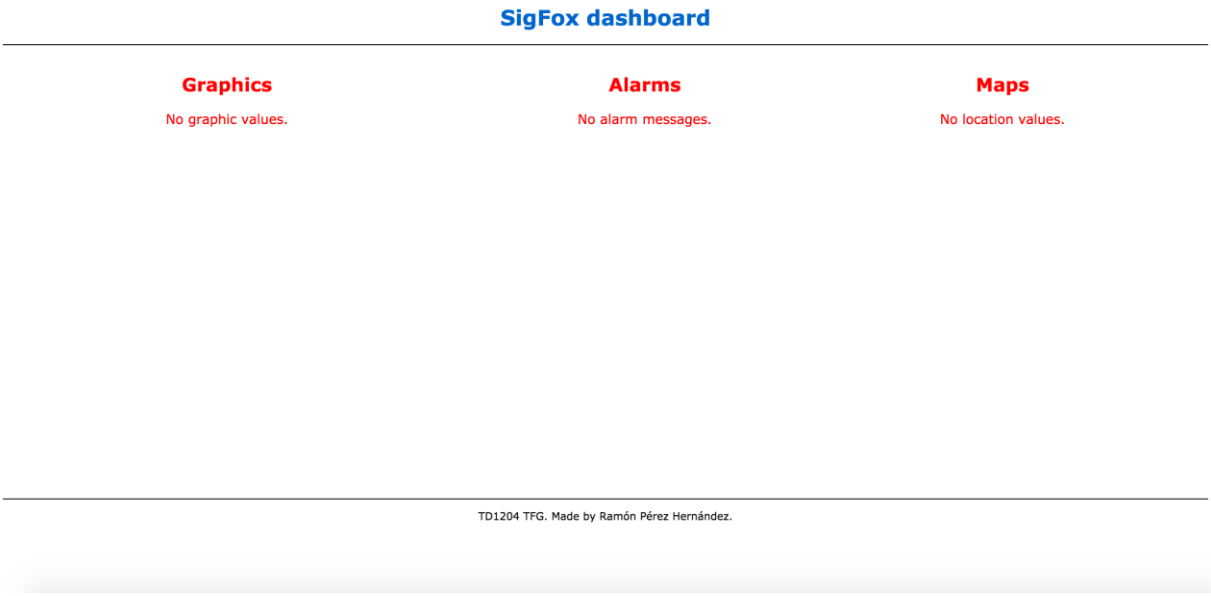


Figura 6-6. Estructura de la aplicación web para la representación de estadísticas (sin contenido a mostrar).

En la aplicación, se podrán visualizar todos los mensajes reenviados dinámicamente. La página está programada para recargarse automáticamente cada 10 minutos, aunque también se puede recargar con la acción del usuario sobre el navegador.

6.2. Definición de las pruebas

Una vez presentadas las herramientas utilizadas para este proceso de pruebas, se especificarán las **tres pruebas**, paso a paso, que probarán el escenario completo que se ha desarrollado en el proyecto: desde la **ejecución** del programa en la placa, hasta la **representación de las estadísticas** en el servidor, pasando por la **recepción y redirección de los mensajes** en el backend.

El objetivo final de estas pruebas será verificar que existe un **funcionamiento correcto** en cada uno de los **casos de uso** programados.

Pasamos a explicar estas **pruebas** con detalle.

6.2.1. Primera prueba

Esta primera prueba verificará el correcto funcionamiento de **todos los casos de uso** programados en la placa, **salvo el cambio de posición**, que será objeto de estudio en el resto de pruebas.

6.2.1.1. Procedimiento a seguir

- Se **modificarán** los valores de los **temporizadores** de cada uno de los **planificadores,** junto a los **umbrales de temperatura y batería** fijados para las alarmas.

  Se hace así para que la prueba **no se extienda en duración** para obtener valores suficientes para la generación de estadísticas (con temporizadores de 30 segundos para RTC, y 10 segundos para batería y temperatura), y para poder probar rápidamente que **se detectan correctamente todos los cambios de estado** para estas dos magnitudes, utilizando sus valores típicos de trabajo y estrechando los umbrales para detectar los cambios a partir de la fluctuación de los niveles en torno a su valor típico (3.32 V para batería, y 25 grados para temperatura).

  Los valores elegidos serán:

  - **B_CHK_INTERVAL:** 10 segundos.

- **LOW_B_LEVEL:** 3320 mV.
- **OK_B_LEVEL:** 3325 mV.
- **T_CHK_INTERVAL:** 10 segundos.
- **LOW_T_LEVEL:** 248 grados*10 (24.8 grados).
- **OK_T_LEVEL_DOWN:** 252 grados*10 (25.2 grados).
- **OK_T_LEVEL_UP:** 256 grados*10 (25.6 grados).
- **HIGH_T_LEVEL:** 260 grados*10 (26 grados).
- **RTC_INTERVAL:** 30 segundos.

- Se **ejecutará la aplicación** modificada durante 1980 segundos (33 minutos), para recoger todos los eventos con seguridad. En las pruebas, la placa estará conectada vía USB al ordenador; luego el nivel de batería registrado será la tensión que entrega el ordenador a la placa en esa conexión (que oscila en torno a 3.32 V).

  Durante la ejecución; tras fijarse la primera posición, se agitará la placa para probar la **detección de movimiento**. Como la configuración de geolocalización no ha cambiado, **no se deberá detectar cambio de posición,** ya que la placa no será desplazada en ningún momento. Además, el GPS estará situado frente a la ventana, para detectar la posición sin **que expire el temporizador** de cambio de posición.

- Se supervisará la ejecución desde la conexión a la placa por el **puerto serie** (vía PuTTY), en el **backend de SigFox**, y en el **servidor LAMP**, hasta finalizar la prueba, para comprobar los resultados obtenidos.

6.2.1.2. Resultados esperados

- Se **verificará** que **los mensajes** enviados por la placa **se reciben correctamente en el backend**, y que éste también realiza **el reenvío** hacia el **servidor LAMP** (se obtendrá el icono verde en la columna Callbacks, como se mostraba en la figura 6-4).
- Los **mensajes** visualizados en el **backend deberán corresponder**, caso por caso, a los **enviados por la plac**a, visualizados desde la conexión por el puerto serie.
- Igualmente, los **valores** mostrados en el **servidor LAMP** corresponderán a los **enviados por la placa**. También se deberá verificar que cada **caso de uso** se muestra en su **zona** correspondiente de la **aplicación web**.

6.2.2. Segunda prueba

En la segunda prueba, se comprobará el caso de **expiración de temporizador** en la detección de **cambio de posición**, actuando conforme al diagrama especificado en la figura 5-8. También se comprobará que **no se puede iniciar una nueva captura de GPS** mientras el **GPS** está **activo**, y que el acelerómetro **no detecta movimiento** mientras se busca la **primera posición**.

6.2.2.1. Procedimiento a seguir

- Se partirá de la aplicación de la primera prueba, cambiando el temporizador del GPS para el caso de cambio de posición. El nuevo valor de **FIX_TIMEOUT** será 10 segundos; un valor lo suficientemente pequeño como para que no le dé tiempo a la aplicación a recibir la posición con la precisión adecuada.
- Se **ejecutará la aplicación**, y se esperará a obtener la primera posición conocida. En ese período, se **agitará** la placa, comprobando que **no se detecta movimiento**, al no estar configurado el acelerómetro todavía.
- Tras conseguir dicha posición y habilitarse la captura de GPS, se **agitará** la placa para detectar un

único **movimiento**, y así comenzar con una nueva captura del GPS.

- Se **agitará** de nuevo la placa, comprobando que **no se inicia una nueva captura de GPS**. Con esto, se esperará a que **venza el temporizador.** Tras ello, **se volverá a agitar** la placa, comprobando que se realiza una **nueva captura**. Tras esto, se terminará la ejecución de la aplicación.
- Se **supervisará** la ejecución por la **conexión por el puerto serie**. En esta prueba, no se hará uso de las estadísticas del servidor LAMP. Se deshabilitará, de hecho, la redirección de mensajes al servidor, desmarcando la casilla Enable en el panel de control de Callbacks del backend (ver figura 6-1).

6.2.2.2. Resultados esperados

- Se deberá comprobar, en la conexión por el puerto serie, que **no se genera contenido** de **detección de movimiento** al **agitar** la placa mientras **busca la primera posición**.
- También se deberá comprobar que, durante la **segunda captura** de GPS, **tampoco** se genera **contenido de inicio de captura** al **agitar** nuevamente la placa.
- Por último, se verificará que se **notifica** correctamente la **expiración del temporizador**, y que se comienza con una **nueva captura** al **agitar** la placa.
- En cuanto al **backend** de SigFox, se deberá comprobar que **no se reenvían los mensajes** de esta prueba al servidor LAMP (el **icono** de Callbacks estará en **gris**).

6.2.3. Tercera prueba

La última prueba comprobará el caso de **cambio de posición**, y cómo actúa la aplicación web de representación de estadísticas cuando se envía una nueva batería de mensajes teniendo ya almacenada información en la base de datos (en concreto, de la primera prueba).

6.2.3.1. Procedimiento a seguir

- Se volverá a **habilitar el reenvío de mensajes** del backend al servidor LAMP, marcando la casilla Enabled en el panel de control de Callbacks del backend (ver figura 6-1).
- Se **eliminará** toda la **información** de la **tabla alarms** de la base de datos, **salvo la primera entrada**; correspondiente al primer mensaje generado en la primera prueba, para mostrar que se está produciendo una **nueva ejecución** de la aplicación.
- Se utilizarán los **mismos valores** especificados en la **primera prueba** (con el mismo valor de FIX_TIMEOUT, de 10 minutos), pero cambiando la **diferencia entre latitudes/longitudes** de 0.1 minutos a **0.001 minutos** (que equivale a una distancia del punto a cualquiera de los umbrales de unos 2 metros). El valor de **DIFF_LAT_LNG** será, entonces, de 100.
- Se **ejecutará la aplicación**, y se esperará a que se obtenga la **primera posición**.
- Posteriormente, se **agitará** la placa, detectando movimiento. Se esperará a que concluya con la **captura de la posición**.
- Se **repetirá** el último paso **hasta que se obtenga una nueva posición**, como producto de la detección de un **cambio de posición**. Al tener el GPS en interiores (frente a la ventana), el **error de precisión** del GPS provocará este cambio de posición.
- Se supervisará la ejecución desde la conexión a la placa por el **puerto serie** (vía PuTTY), en el **backend de SigFox**, y en el **servidor LAMP**, hasta finalizar la prueba, para comprobar los resultados obtenidos.

6.2.3.2. Resultados esperados

- Se mostrará, a través de **la conexión por el puerto serie**, que la placa detecta una **nueva posición** como fruto de un **cambio de posición**, con las coordenadas correspondientes.

- Se comprobará, en el **backend** de SigFox, que se recibe el mensaje de **detección de cambio de posición** correctamente, con el formato y contenido adecuado.
- Se verificará, en el **servidor LAMP**, que se han recibido las **dos nuevas posiciones** (la primera posición detectada, y la siguiente de cambio de posición), y que se puede **diferenciar** de la posición de la **primera prueba** a partir de las **fechas de ocurrencia;** comparándolas con la del **primer mensaje** de cada prueba.
- Además, se podrá comprobar, en el **servidor LAMP**, el **salto de número de secuencia** a la hora de leer los valores de RTC, al iniciar una nueva captura.

6.3. Resultado final de las pruebas

Finalmente, se mostrará el **resultado final** de cada una de las pruebas explicadas anteriormente, verificando el **correcto funcionamiento** de todas las partes implicadas en el proceso.

6.3.1. Primera prueba

6.3.1.1. Contenido generado en la conexión por el puerto serie

A través del **puerto serie**, se puede comprobar qué mensajes emite la placa en cada evento producido, por la programación realizada. Se muestra, a continuación, esta información, mostrando **la primera ocurrencia de cada caso de uso** (en orden), para evitar información excesiva y redundante.

- **Mensaje inicial.**

```
Starting fixing.
Sending message to backend: It works.
```

- **Batería baja.**

```
Low battery event. Value: 3319 mV
ThresholdFunction (battery case). Time 11
Sending message to backend with monitoring data.
```

- **Temperatura alta.**

```
High temperature event. Value: 26.6 degrees
ThresholdFunction (temperature case). Time 17
Sending message to backend with monitoring data.
```

- **Alarma de RTC.**

```
TimerFunction. SN 1. Time 31
Temperature: 26.6 degrees
Power supply voltage: 3327 mV
Sending message to backend with all the information.
```

- Obtención de la **primera posición conocida**.

```
First location: 37 deg 22.09690 min N   5 deg 58.86159 min W
GPSInitialFix. Time 52
Sending message to backend with first location data.
```

- **Detección de movimiento; notificación**, y **desactivación del GPS** al no pasar de 0.1 minutos de diferencia en latitud o longitud.

```
Movement detected.
MoveDetected. Time 68
Starting fixing.

Movement detected.
Movement detected.
Movement detected.
Movement detected.
MoveDetected. Time 72
Reached limit of movements. Sending a message right now.

…
```

```
Movement detected.
Movement detected.
Stop fixing.
Fixing on move enabled.
```

- **Temperatura baja.**

```
Low temperature event. Value: 24.4 degrees
ThresholdFunction (temperature case). Time 98
Sending message to backend with monitoring data.
```

- **Batería correcta**

```
Ok battery event. Value: 3326 mV
ThresholdFunction (battery case). Time 111
Sending message to backend with monitoring data.
```

- **Temperatura correcta.**

```
Ok temperature event. Value: 25.6 degrees
ThresholdFunction (temperature case). Time 668
Sending message to backend with monitoring data.
```

Comprobamos que se han producido todos los **casos de uso** contemplados en esta prueba. Restará por saber si estos valores han llegado a las dos plataformas utilizadas para la visualización de los mensajes.

6.3.1.2. Mensajes visualizados en el backend de SigFox

A continuación, veremos que han **llegado** todos los mensajes **al backend**; comprobando que tanto su **formato** como su **contenido** es el **correcto**.

Para continuar con el estudio de cada caso de uso, mostraremos los mensajes recibidos para los eventos mencionados en el subapartado anterior:

- **Mensaje inicial.**

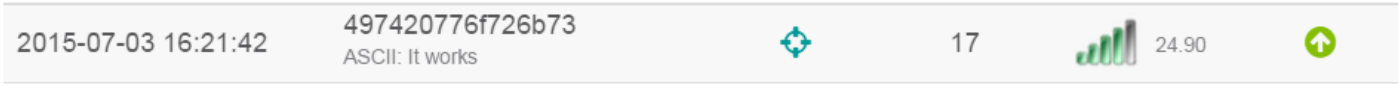


Figura 6-7. Mensaje inicial. *Backend de SigFox.*

Efectivamente; en la traducción a ASCII, vemos como se muestra el mensaje “It works”.

- **Batería baja.**

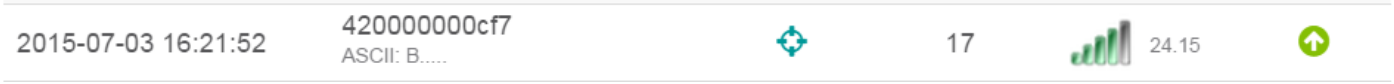


Figura 6-8. Alarma de batería baja. *Backend de SigFox.*

Vemos que la cabecera es correcta (B, de alarma de batería). El siguiente byte, a 0x00, corresponde a batería baja (0). Y el valor capturado es 0xcf7, que en decimal es 3319. Por lo tanto, el formato y contenido del mensaje es el correcto.

- **Temperatura alta.**

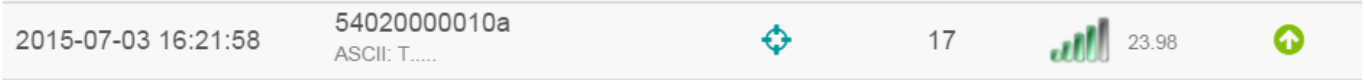


Figura 6-9. Alarma de temperatura alta. *Backend de SigFox.*

La cabecera indica que es una alarma de temperatura (T). El estado, que es 0x02, corresponde a temperatura alta (2). El valor sería el 0x10a, 266 en decimal. Luego, tenemos otro mensaje correcto.

- **Alarma de RTC.**

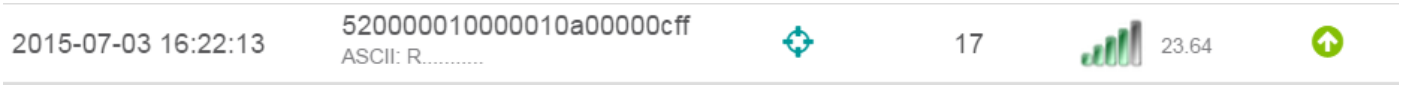


Figura 6-10. Alarma de RTC. *Backend de SigFox.*

La cabecera muestra que es una alarma de RTC (R). Le sigue el número de secuencia en 3 bytes, con valor 1. El valor de temperatura es 0x10a (266) y el de batería 0xcff (3327). Mensaje correcto.

- Obtención de la **primera posición conocida**.

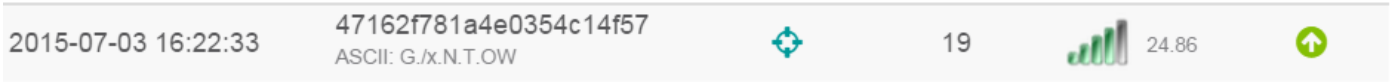


Figura 6-11. Primera posición conocida. *Backend de SigFox.*

Se encabeza con la G que indica geolocalización (para la primera posición). Le sigue la latitud, con valor 0x162f781a, que en decimal es 372209690 (es decir, 37º 22.09690'). La dirección será N. Para la longitud, tenemos el valor 0x0354c14f, que equivale a 55886159 (5º 58.86159'), con dirección W. Nuevamente, el mensaje es correcto.

- **Detección de movimiento** (sólo **notificación**).

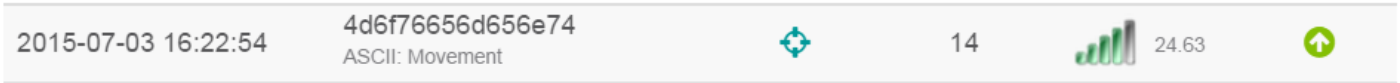


Figura 6-12. Detección de movimiento. *Backend de SigFox.*

Vemos que la notificación es la esperada, con el mensaje "Movement" en la traducción a ASCII.

- **Temperatura baja.**

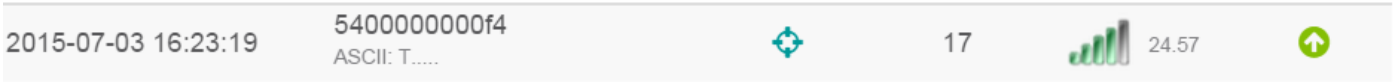


Figura 6-13. Alarma de temperatura baja. *Backend de SigFox.*

Se encabeza con la T de alerta de temperatura, y le sigue el código 0x00, de temperatura baja (0). El valor es 0xf4, que en decimal equivale a 244. Otro mensaje correcto.

- **Batería correcta.**

Figura 6-14. Alarma de batería correcta. *Backend de SigFox.*

Tenemos la cabecera B de alarma de batería. El estado es el 0x01, de batería correcta (1). El valor detectado es el 0xcfe, que equivale a 3326 en decimal. Mensaje correcto.

- **Temperatura correcta.**

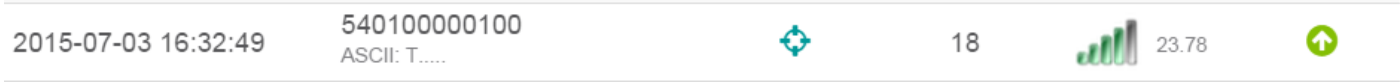


Figura 6-15. Alarma de temperatura correcta. *Backend de SigFox.*

El mensaje viene encabezado con la T de alarma de temperatura. Con el estado 0x01, se notifica batería correcta (1). El valor capturado es el 0x100, que es 256 en decimal. Mensaje correcto.

Concluimos que **todos los mensajes** mostrados anteriormente **se reciben correctamente en el backend**. De mostrar el resto de eventos capturados, se comprobaría que también se recibieron correctamente.

Antes de pasar al servidor LAMP, notar que todos los mensajes recibidos tienen el icono de Callbacks en verde, lo que demuestra que todos los mensajes han sido **reenviados correctamente** al servidor de procesamiento de datos. Esto se puede extender también al resto de mensajes no presentados, demostrándolo en la siguiente figura para los últimos mensajes recibidos en el backend.

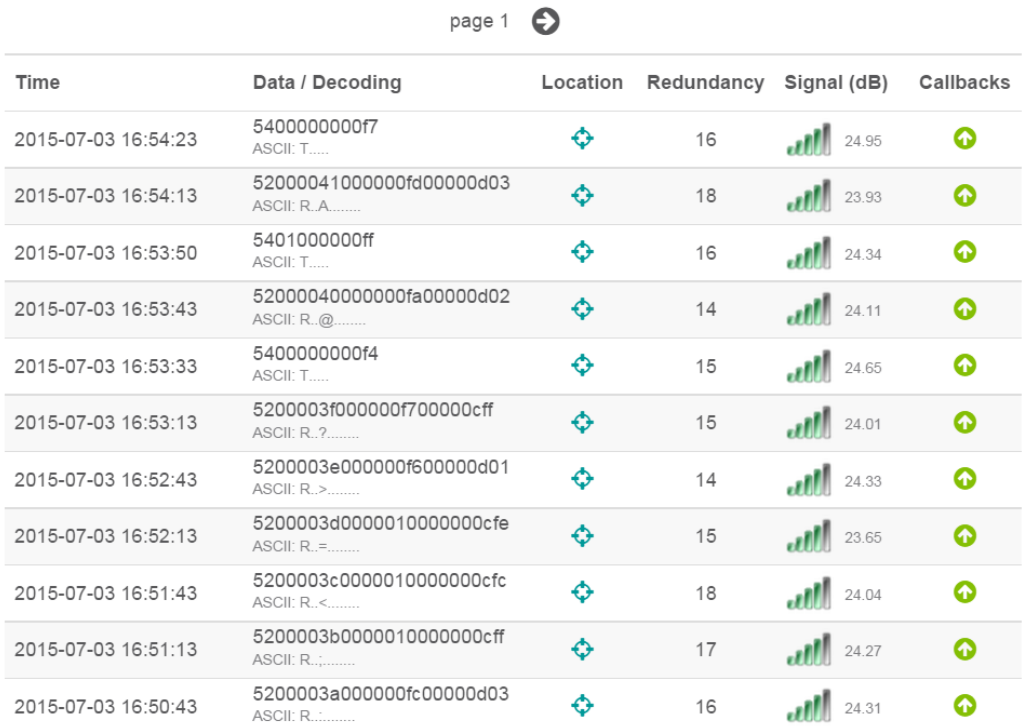


Figura 6-16. Panel de mensajes recibidos en la primera prueba, con los últimos mensajes recibidos por el backend y redirigidos correctamente al servidor. *Backend de SigFox.*

6.3.1.3. Generación de estadísticas en el servidor LAMP

Si visualizamos las estadísticas mostradas en el backend tras la realización de las pruebas, veremos que **todos los casos de uso se han notificado en su sección correspondiente**, como se puede apreciar en la siguiente figura. En el caso de las gráficas, además, vemos que no se ha notificado ningún mensaje de salto en la secuencia, luego han llegado todos los datos correctamente.

Se recuadrarán los **mensajes de alarma** estudiados en las subsecciones anteriores (primer mensaje,

temperatura, batería y movimiento), comprobando que también **se conservan los valores notificados** por la placa y la **fecha de ocurrencia.**

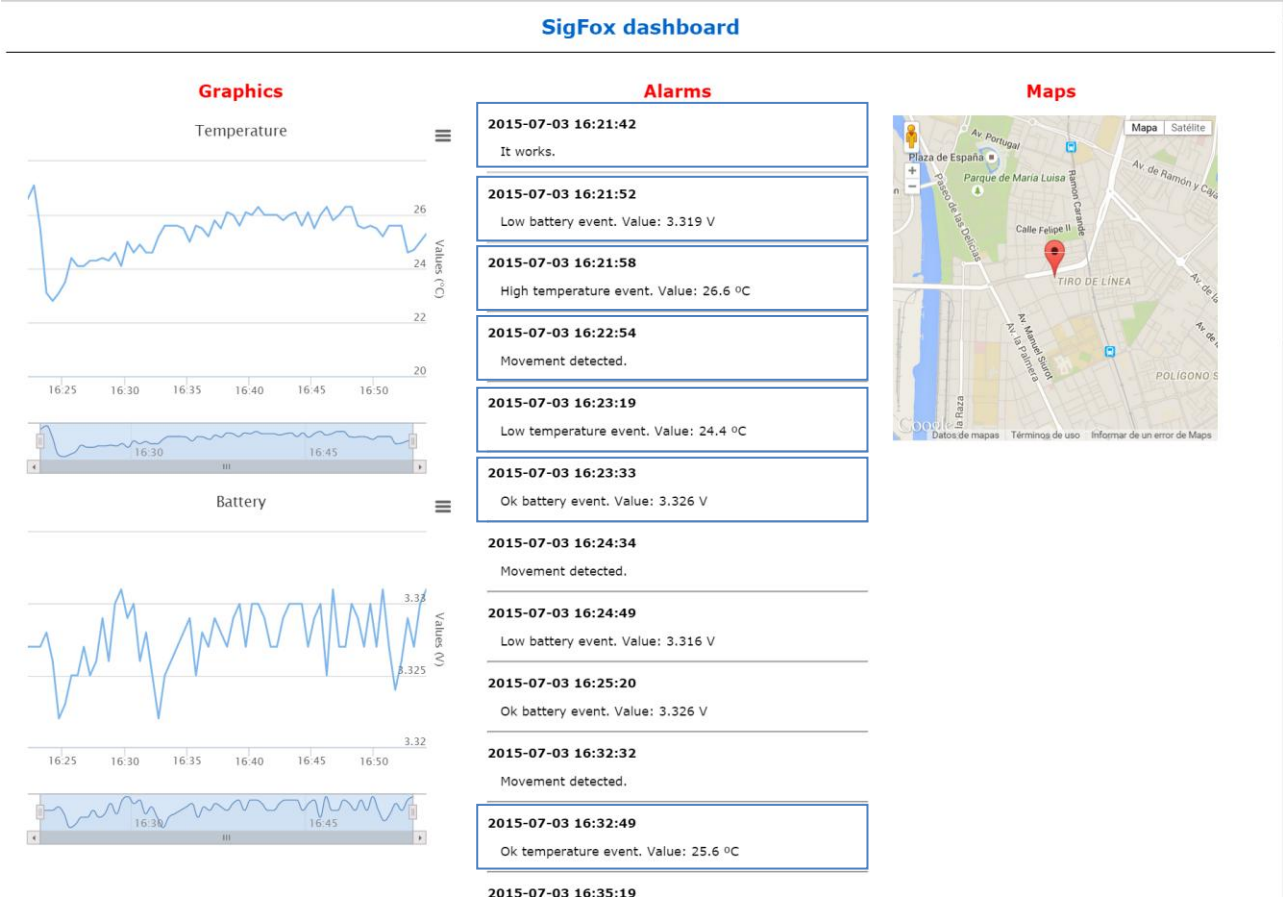


Figura 6-17. Estadísticas generadas en el servidor LAMP para la primera prueba.

Para comprobar que los mensajes de **alarma de RTC** y de la **primera posición conocida** también se reenviaron correctamente, mostraremos el contenido almacenado en la **base de datos,** en las tablas graphics y maps.

- **Alarma de RTC.**

id	date	temperature	battery
1	2015-07-03 16:22:13	26.6	3.327

Figura 6-18. Mensaje de alarma de RTC almacenado en la base de datos.

- Obtención de la **primera posición conocida.**

date	latitude	longitude
2015-07-03 16:22:33	37.368281666667	-5.9810265

Figura 6-19. Mensaje de la primera posición conocida almacenado en la base de datos.

El almacenamiento de la posición difiere en formato con respecto al capturado por la placa, ya que está representado en **grados con decimales**, y no en grados enteros y minutos con decimales. De hacer la

conversión (sumar a los grados los minutos divididos entre 60), obtendríamos ese resultado final, luego se conserva la información.

En base a toda la información mostrada, podemos concluir que esta **primera prueba** ha sido **superada** satisfactoriamente.

6.3.2. Segunda prueba

6.3.2.1. Contenido generado en la conexión por el puerto serie

Como se indicó en el procedimiento a seguir para la realización de esta segunda prueba, únicamente se supervisará la **información** generada por la placa en la **conexión por el puerto serie**.

Se procede a mostrar dicha información, limitándonos únicamente a la información referente a la **detección de movimiento y GPS**, por orden de aparición.

- Comienza la **búsqueda de la primera posición**. Agitar la placa no genera ningún mensaje.

```
Starting fixing.
```

- Se encuentra la **primera posición**. Al agitar la placa, se detecta **movimiento**, y se inicia la **captura del GPS**.

```
First location: 36 deg 27.79867 min N   6 deg 11.86618 min W
GPSInitialFix. Time 54
Sending message to backend with first location data.

Movement detected.
MoveDetected. Time 80
Starting fixing.
```

- **Movemos** la placa durante la captura, y vemos que **se notifica el movimiento**, pero **no** se inicia una **nueva captura**.

```
Movement detected.
```

- Al **vencer el temporizador**, se notifica que **se vuelve a habilitar la captura** de GPS. Si **agitamos** de nuevo la placa, se detecta el **movimiento** y **se inicia la captura** de nuevo.

```
Stop fixing.
Timeout reached.
Fixing on move enabled.

Movement detected.
MoveDetected. Time 105
Starting fixing.
```

- Esperamos a que **venza el temporizador**, para comprobar que el funcionamiento vuelve a ser el correcto.

```
Stop fixing.
Timeout reached.
Fixing on move enabled.
```

Se concluye, así, que el comportamiento de la placa en esta prueba ha sido el esperado, obteniendo un **resultado satisfactorio**.

6.3.2.2. Mensajes visualizados en el backend de SigFox

Se puede comprobar que, efectivamente, **no se han reenviado los mensajes** generados en esta prueba, ya que los **iconos** de la columna **Callbacks** se muestran de color **gris**.

Time	Data / Decoding	Location	Redundancy	Signal (dB)	Callbacks
2015-07-05 19:59:50	420000000cf7 ASCII: B.....		5	23.84	
2015-07-05 19:59:37	420100000cff ASCII: B.....		4	23.51	
2015-07-05 19:59:30	5200000300000010a00000cfc ASCII: R.....		4	23.62	
2015-07-05 19:59:20	420000000cf7 ASCII: B.....		4	24.09	
2015-07-05 19:59:07	420100000cff ASCII: B.....		5	24.19	
2015-07-05 19:59:00	5200000200000010a00000d00 ASCII: R.....		5	24.03	
2015-07-05 19:58:53	47159f94db4e03a5a23a57 ASCII: G....N...:W		5	24.91	
2015-07-05 19:58:36	5200000100000010800000cf7 ASCII: R.....		4	23.79	
2015-07-05 19:58:30	420000000cf7 ASCII: B.....		5	24.01	
2015-07-05 19:58:20	540200000105 ASCII: T.....		5	23.10	
2015-07-05 19:58:00	497420776f726b73 ASCII: It works		5	24.45	

Figura 6-20. Panel de mensajes recibidos en la segunda prueba, mostrando que no se han reenviado los mensajes al servidor. *Backend de SigFox.*

6.3.3. Tercera prueba

6.3.3.1. Contenido generado en la conexión por el puerto serie

Se muestra, a continuación, la **información** proporcionada por la placa, que se ha leído a través de la **conexión por el puerto serie**. Se omite cualquier contenido que no tenga que ver con la geolocalización y el acelerómetro. El proceso será el siguiente:

- Comienza la **búsqueda de la primera posición**.

```
Starting fixing.
```

- Se encuentra la **primera posición**. Al agitar la placa, se detecta **movimiento**, y se inicia la **captura del GPS**.

```
First location: 36 deg 27.79450 min N   6 deg 11.87397 min W
GPSInitialFix. Time 159
Sending message to backend with first location data.
```

- Se detecta un **cambio de posición**, notificándose la nueva posición al backend. Notar que la diferencia entre la nueva latitud y la primera es 0.00147, mayor que 0.001.

```
Stop fixing.
Position change detected: 36 deg 27.79303 min N   6 deg 11.87044 min W
GPSFix. Time 249
Sending message to backend with location data.
Fixing on move enabled.
```

De esta forma, se puede comprobar que **se notifica correctamente el cambio de posición**. Ahora, habrá que comprobar que la información se recibe y procesa adecuadamente.

6.3.3.2. Mensajes visualizados en el backend de SigFox

A continuación, se muestra el **mensaje de cambio de posición** que llega al backend, que es el último caso de uso que queda por comprobar.

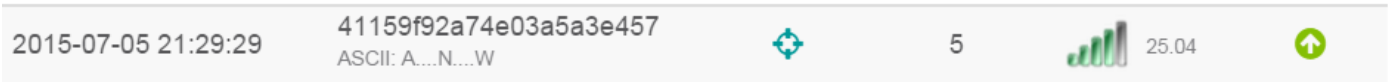


Figura 6-21. Detección de cambio de posición. *Backend de SigFox.*

El mensaje viene encabezado por el carácter A (de acelerómetro, ya que el cambio de posición es consecuencia de un movimiento previo). A continuación, tenemos la latitud, de valor 0x159f92a7, que en decimal es 362779303 (es decir; 36º 27.79303'), con dirección N. La longitud será 0x03a5a3e4, que en decimal equivale a 61187044 en decimal (6º 11.87044'), con dirección W. Por lo tanto, el mensaje se envía correctamente.

6.3.3.3. Generación de estadísticas en el servidor LAMP

Finalmente, vamos a ver el aspecto que presenta la aplicación web de generación de estadísticas, tras concluir con esta tercera prueba, y manteniendo la información generada en la primera prueba (salvo los mensajes de alarma, de los cuales sólo se ha mantenido el primer mensaje).

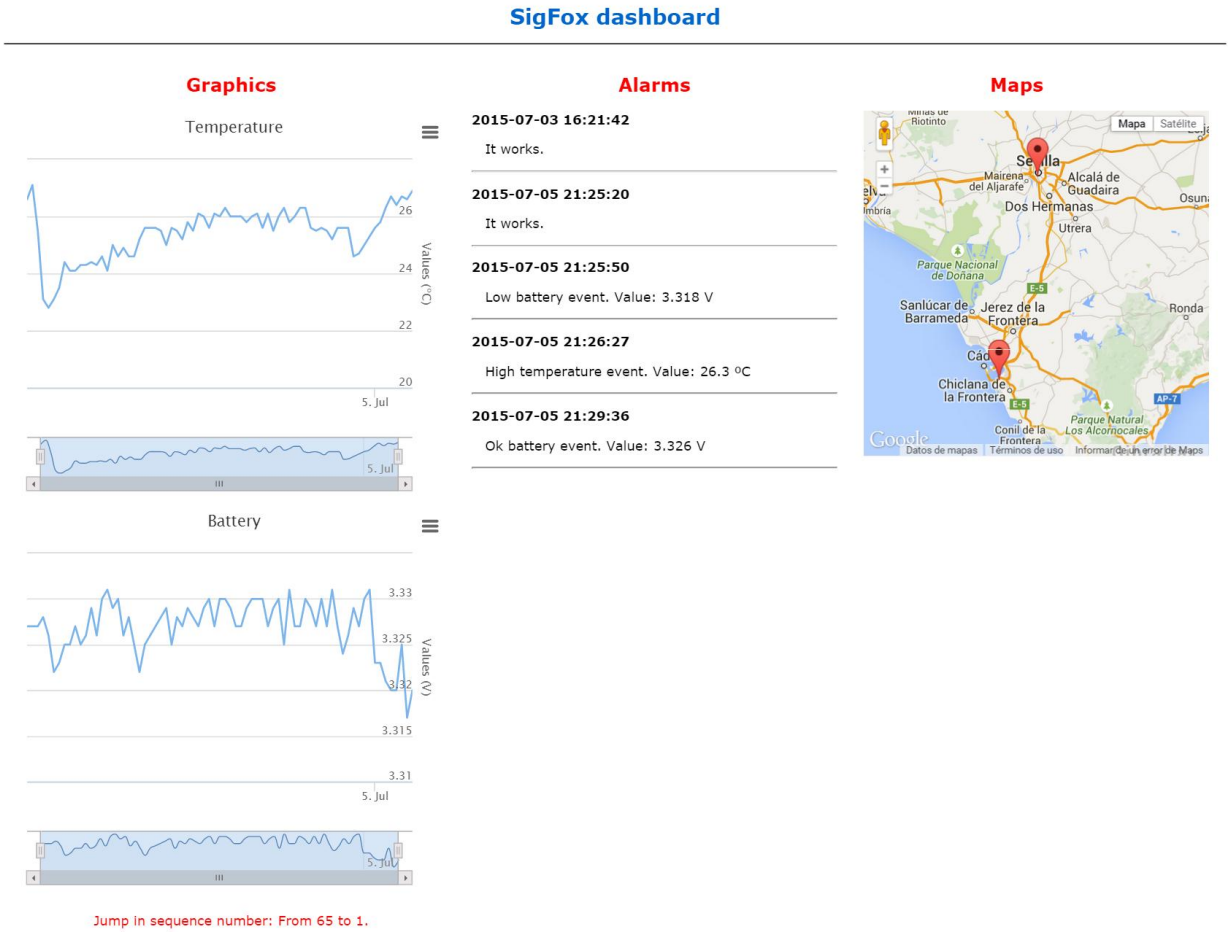


Figura 6-22. Estadísticas tras concluir la tercera prueba

Vamos a fijarnos, sección a sección, qué novedades se presentan con respecto a la primera prueba:

- En la sección de **gráficas**, vemos que se notifica un **salto en la secuencia**. Recordamos que se han conservado todos los datos de la primera prueba, luego los valores leídos en la alarma de RTC de la tercera prueba se han almacenado a continuación de éstos.

Si nos fijamos en la **fecha de las gráficas**, en la parte inferior, vemos que hay una línea que marca el **cambio de día**. Este hecho se puede ver, también, en los datos almacenados en la tabla base de datos.

63	2015-07-03 16:53:13	24.7	3.327
64	2015-07-03 16:53:43	25	3.33
65	2015-07-03 16:54:13	25.3	3.331
1	2015-07-05 21:25:56	25.6	3.323
2	2015-07-05 21:26:20	25.8	3.323
3	2015-07-05 21:26:50	26.3	3.321

Figura 6-23. Información almacenada en la tabla graphics tras la tercera prueba, destacando el salto en la secuencia y en la fecha.

- Las **alarmas** se muestran correctamente, sin ninguna notificación adicional. Destacar la fecha de los dos mensajes "It works", para justificar la diferenciación de una posición inicial con la de un cambio de posición.
- Por último, en el **mapa**, vemos que se han detectado **dos nuevas posiciones,** si nos acercamos en el mapa hacia la zona donde se ha capturado la nueva posición.

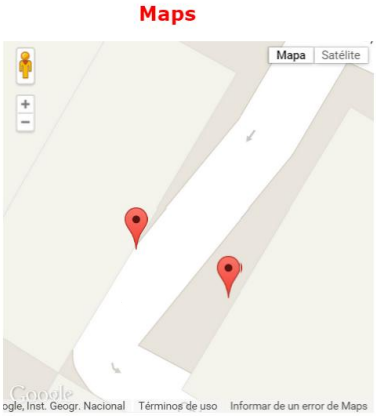


Figura 6-24. Detección del cambio de posición en la tercera prueba.

También se muestra su almacenamiento en la tabla maps. Se realiza la misma operación que la empleada para transformar en decimal la primera posición conocida. Para **diferenciar** una de otra, bastará con pasar el cursor sobre el marcador, y se mostrará el orden de la captura (2 para la primera posición, y 3 para el cambio de posición), junto a la fecha de ocurrencia.

date	latitude	longitude
2015-07-03 16:22:33	37.368281666667	-5.9810265
2015-07-05 21:27:58	36.463241666667	-6.1978995
2015-07-05 21:29:29	36.463217166667	-6.1978406666667

Figura 6-25. Información almacenada en la tabla maps tras la tercera prueba, con las dos nuevas posiciones capturadas

¿Pero cómo saber que, efectivamente, la posición 2 es una primera posición? Si nos fijamos en la base de datos, la primera posición almacenada (posición 1), corresponde a la primera posición capturada de la primera prueba, y que coincide con la fecha del primer mensaje de bienvenida recogido en las alarmas (3 de julio, sobre las 4 y media de la tarde). Las otras dos posiciones corresponden en fecha

con el segundo mensaje de bienvenida (5 de julio, sobre las 9 y media de la noche). La primera posición de ellas (la 2) será la primera posición conocida), y la segunda (la 3) será el cambio de posición, como ya se ha comentado anteriormente.

6.4. Conclusiones tras la realización de las pruebas

Tras este proceso de pruebas sobre la aplicación desarrollada, se verifica que **todos los casos de uso funcionan correctamente**, sin excepción. La aplicación final sólo cambiará en los valores de temporizadores y umbrales de detección, con respecto al código utilizado en las pruebas; pero la lógica es la misma, que es lo que se ha probado en esta batería de pruebas.

7 ASPECTOS FINALES

"Todo concluye, pero nada perece"

- Séneca, filósofo -

En el transcurso de esta memoria, se han estudiado las posibilidades que ofrece la placa de evaluación TD1204 para poder monitorizar los casos de uso de interés, desembocando en el desarrollo, codificación y prueba de una aplicación que detecte, procese y muestre convenientemente dichos casos de uso. Una vez concluido ese proceso, conviene extraer una serie de **conclusiones** acerca del funcionamiento del dispositivo en cuestión.

Para ello, **compararemos** el funcionamiento del **TD1204** con un módem ya utilizado por Wellness Smart Cities para estas labores de monitorización: el **SIM900D** (datasheet disponible en [41]), que utiliza **GPRS** para la comunicación. Antes de entrar en detalles sobre esta comparación, se mencionarán varios conceptos clave de interés sobre GPRS y su relación con el Internet de las Cosas.

7.1. Breve reseña sobre GPRS

GPRS es una extensión del estándar GSM, que añade la funcionalidad de la red IP, de **conmutación de paquetes,** para el **tráfico de datos a ráfagas**, ofrecido junto al clásico tráfico de voz sobre conmutación de circuitos [42].

Además de la inclusión del tráfico de datos junto al de voz, se ofrecen **velocidades de transmisión** mayores tanto en las conexiones de bajada como de subida, alcanzándose una velocidad teórica de 171 kbps; aunque, típicamente, las velocidades se mueven en máximos de 14 kbps en sentido ascendente, y 40 kbps en sentido descendente.

Al ser una transmisión adaptada al tráfico de datos con baja velocidad (en comparación a las redes móviles de última generación), es una **alternativa** para la transmisión de información sobre el Internet de las Cosas. Bastará con equipar al dispositivo con una **tarjeta SIM** que permita su conexión a la red GPRS para el intercambio de información.

7.2. Comparación entre el TD1204 y el SIM900D

Compararemos estos dos dispositivos en los siguientes **ámbitos**; extrayendo, finalmente, una serie de **ventajas y limitaciones** del cambio al TD1204 frente al actual uso del SIM900D.

7.2.1. Consumo energético

Para el estudio del **consumo energético** este estudio, se utilizará una **hoja de cálculo** proporcionada por Wellness Smart Cities; la cual, en base al consumo del dispositivo en diferentes casos y a la batería que se utilice, permitirá obtener la **duración estimada** de la batería. Se tendrán en cuenta los siguientes **parámetros generales**:

- **Batería:** se contará con que el SIM900D requiere de 2 baterías, en paralelo, del modelo **ER26500M** [43], para satisfacer su pico de intensidad, que puede llegar a 1-2 amperios. Por lo tanto, como cada batería de este modelo proporciona 6500 mAh, 2 baterías en paralelo proporcionarán 13000 mAh. Se utilizará este valor de batería para ambos dispositivos, convertido a µAh para facilitar los cálculos posteriores (multiplicando por 1000, obteniendo 13000000 µAh).
- **Error** o **tolerancia** para el valor anterior de batería. Se admite una variación en el valor anterior de un 10%, para ambos casos.
- **Cobertura:** a mayor valor de este parámetro, peor cobertura se presentaría en el entorno real de uso. Se supone una cobertura 1 (el mejor nivel) en ambos casos; luego, no afectaría a los resultados.

Capacidad	13000000 mAh
Error	10%
Cobertura	1 (alta)

Tabla 7–1. Parámetros generales para la comparativa.

A continuación, se proporcionan los **datos de consumo** del SIM900D cuando el dispositivo está **dormido**, cuando realiza **medidas**, y cuando **transmite** información vía GPRS. Estos valores han sido proporcionados por Wellness Smart Cities, y son valores medidos en una aplicación real. Se emplean los siguientes **atributos** para calcular el consumo total:

- **Consumo por evento:** valor medio del consumo de cada uno de los eventos, individuales, considerados en el estudio, en µA.
- **Duración por evento:** tiempo empleado para que ocurra un evento individual, en segundos.
- **Período de ocurrencia:** cada cuánto tiempo ocurre el evento considerado. Medido en segundos.
- **Duración por día:** cuánto tiempo se emplea, por día, para cada uno de los eventos considerados. También se mide en segundos. Su cálculo se realiza con la fórmula 7-1:

$$duración\ por\ día\ (s) = \frac{86400\ (segundos\ en\ un\ día)}{período\ de\ ocurrencia\ (s)}\ x\ duración\ por\ evento\ (s) \qquad (7–1)$$

La duración del **estado de reposo** se puede calcular con la fórmula 7-2:

$$duración\ de\ reposo\ (s) = 86400 - \sum duración\ por\ día\ del\ resto\ de\ eventos\ (s) \qquad (7–2)$$

Con esto, se obtienen los siguientes **valores**:

Eventos	Consumo por evento (µA)	Duración por evento (s)	Período de ocurrencia (s)	Duración por día (s)
Dormido (reposo)	40	-	-	85960
Toma de medidas	30000	10	10800	80
Transmisión GPRS	100000	180	43200	360

Tabla 7–2. Medidas del SIM900D.

El **reparto de tiempo**, por día, de ocurrencia de cada evento, será el siguiente. Notar que el módem, la mayor parte del tiempo, permanece dormido.

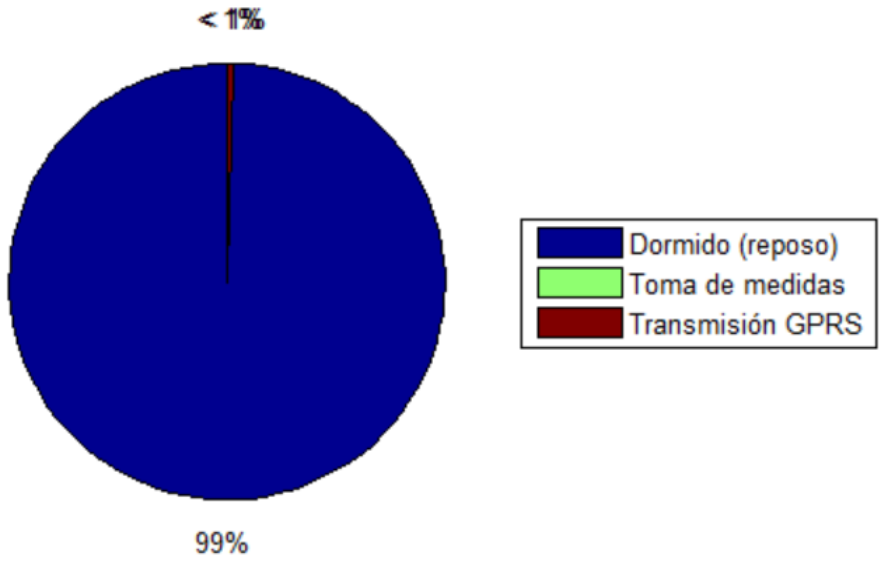


Figura 7-1. Porcentaje de tiempo de ocurrencia de cada evento, por día, para el SIM900D. *Realizado por Matlab.*

Finalmente, pasaríamos a calcular el **consumo medio** de la placa al día, en µA, con la fórmula 7-3:

$$consumo\ medio\ (\mu A) = \frac{\sum consumo\ por\ evento\ x\ duración\ por\ día}{86400} \qquad (7–3)$$

Y, a partir de este valor, se calcularía la **duración de la batería**, en horas, con la fórmula 7-4, que utiliza la capacidad de la batería y la corrección aplicada con el error considerado:

$$duración\ batería\ (\mathrm{h}) = \frac{capacidad\ batería\ (\mu Ah) x \frac{100-error}{100}}{consumo\ medio\ (\mu A)} \qquad (7–4)$$

Este último valor se puede pasar a días dividiendo entre 24 (1 día tiene 24 horas). Y este valor obtenido, a su vez, se puede convertir en años, dividiéndolo entre 365 (1 año tiene 365 días).

Finalmente, se puede estimar el **número de transmisiones** realizadas en este período de duración de la batería, con la fórmula 7-5:

$$nº\ transmisiones = \frac{86400}{período\ ocurrencia\ transmisión\ (s)} x\ duración\ batería\ (días) \qquad (7–5)$$

Aplicando estas 3 últimas fórmulas, obtenemos los siguientes **resultados**:

Consumo medio al día (µA)	484,24
Duración de la batería (h)	24161,54
Duración de la batería (días)	1006,73
Duración de la batería (años)	2,76
Número de transmisiones	2013

Tabla 7–3. Valores de consumo del SIM900D.

Pasamos, a continuación, a **comparar** estos resultados con los obtenidos al medir el consumo de la ejecución de la aplicación en la placa de evaluación. Para ello, se utilizó un **multímetro**, y se midió la corriente media producida por los eventos principales que se daban en la aplicación.

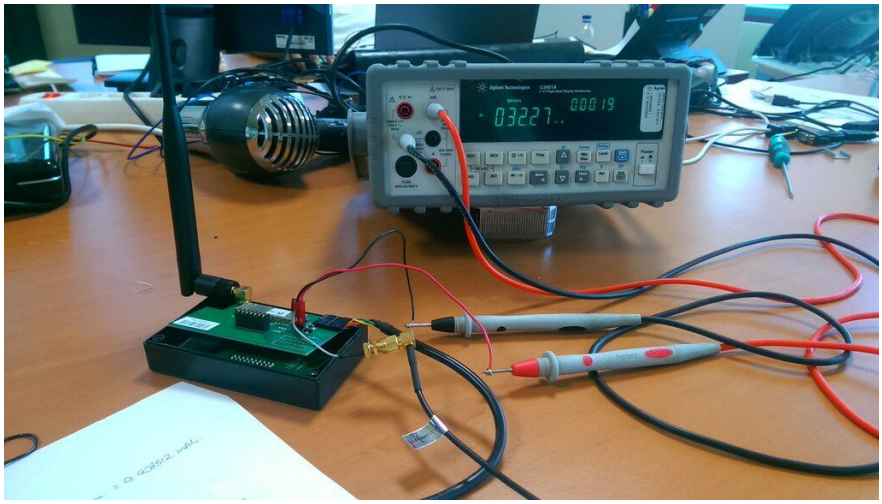


Figura 7-2. Midiendo el consumo de la placa con un multímetro.

Para realizar el conexionado, se deberán conectar los cables del multímetro al **jumper de medición de corriente**, que se indica en la siguiente imagen, y se puede estudiar con mayor profundidad en la guía de uso del TD1204, apartado 3.3.3. Hay que tener precaución con esta conexión y tenerla bien fijada, ya que provoca un reset en el dispositivo en caso de que las conexiones no queden bien unidas. También habrá que atender a la escala del multímetro, ya que puede provocar el apagado de la placa en caso de medir la intensidad con una escala que no corresponda al evento correspondiente (por ejemplo, tener la escala en amperios para medir µA**).**

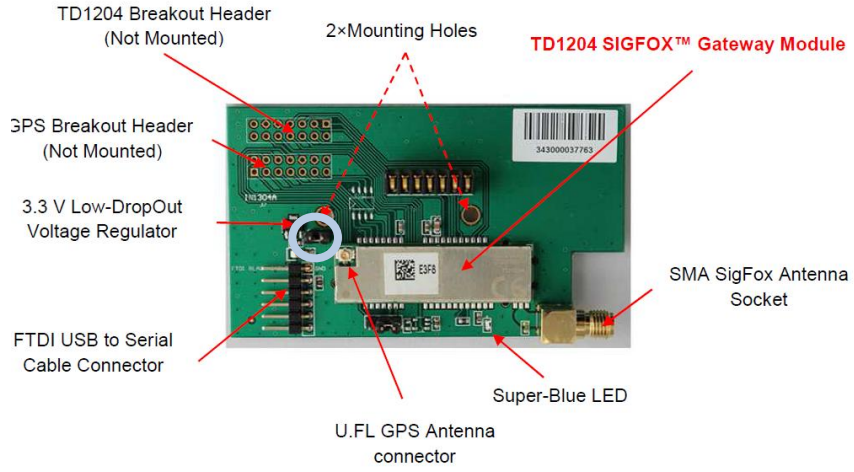


Figura 7-3. Jumper de medición de corriente, señalado en azul claro. *Guía de uso del TD1204.*

Dicho esto, comentamos los **eventos** considerados en la ejecución del TD1204, llevándolos a casos extremos de uso (detección de movimiento, cambios de posición, activación de GPS, detección de alarmas continuamente, etc.):

- **Primera transmisión, y detección de movimiento**: se consideran el mismo evento por el consumo similar que presentan (supondremos que el GPS está activo al detectar movimiento). En ellos, el GPS está activo (unos 30 mA de consumo), se enciende el LED (unos 2 mA) y se transmite información (consume unos 50 mA). Se considera un valor de **82 mA**. Su duración viene marcada por el tiempo que tarda en transmitirse el mensaje, que es de unos **7 segundos** como máximo. Se supondrá que se detecta 1 caso de movimiento al día, que sumados a la primera transmisión (que sólo se realiza una vez, pero se supondrá que, el resto de días, equivale a la detección de un movimiento), suman **2 transmisiones al día**. Es decir, su **período de ocurrencia** será, en media, cada 12 horas (**43200 segundos**).
- **Activación del GPS**: cuando el GPS está buscando una posición, consume **30 mA**. Puede ocurrir que no se esté haciendo nada más, luego hay que considerar su período de activación hasta que encuentra una posición, que puede ser de **180 segundos** (3 minutos) aproximadamente por caso. Como suponemos que el GPS se activa al detectar movimiento, y la primera transmisión también lo enciende, se darán **2 casos al día**; luego el **período de ocurrencia** será, también, de 12 horas (**43200 segundos**).
- **Transmisión de alarmas:** se considerarán las alarmas de temperatura, batería (que ocurren cada 30 minutos) y RTC (cada hora), sin que esté encendido el GPS. Se supondrá que saltará una alarma de batería o temperatura en cada ocurrencia, luego tendremos **3 transmisiones (de 7 segundos cada una) cada hora**. Se computa el consumo de transmisión (50 mA) y de encendido de LED (2 mA), resultando **52 mA. Su período de ocurrencia** será, en media, cada 20 minutos **(1200 segundos).** No se considerará el consumo provocado por el planificador, que es despreciable con respecto a estos valores (del orden del μA).
- **Primera posición del GPS y cambio de posición:** como el GPS se desactiva antes de su notificación, se considerará el mismo consumo que la transmisión de alarmas, con **52 mA** en **7 segundos** de transmisión. Se supondrá que se detecta un cambio de posición por día, que sumado a la primera posición, resultan **2 transmisiones por día** (en los días siguientes al primero, se supondrán 2 cambios de posición al día, ya que la primera posición sólo se detecta una vez). Su **período de ocurrencia** será cada 12 horas (**43200 segundos**).
- **Reposo**: el resto del tiempo, con un consumo de **2 μA** como máximo, contando con el procesamiento del acelerómetro en el bloque loop.

Con estos valores, se obtiene la siguiente **tabla de medidas,** al día, del **TD1204**:

Eventos	Consumo por evento (μA)	Duración por evento (s)	Período de ocurrencia (s)	Duración por día (s)
Dormido (reposo)	2	-	-	85508
Primera transmisión y detección de movimiento	82000	7	43200	14
Activación del GPS	30000	180	43200	360
Transmisión de alarmas	52000	7	1200	504
Primera posición del GPS y cambio de posición	52000	7	43200	14

Tabla 7–4. Medidas del TD1204.

El reparto de tiempo de eventos, por día, será el siguiente, en este caso. También vuelve a ocurrir que la placa, la mayor parte del tiempo, permanece en reposo.

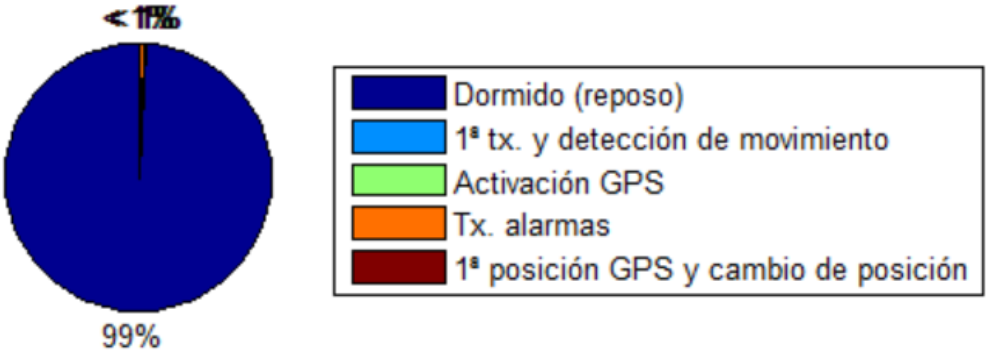


Figura 7-4. Porcentaje de tiempo de ocurrencia de cada evento, por día, para el TD1204. *Realizado por Matlab.*

Que nos da los siguientes resultados de **consumo**, aplicando las fórmulas antes mencionadas.

Consumo medio (µA)	452,03
Duración batería (h)	25883,49
Duración batería (días)	1078,48
Duración batería (años)	2,95
Número transmisiones	81964

Tabla 7–5. Valores de consumo del TD1204.

Notar, como detalle, que para el número de transmisiones, deberán considerarse los 3 casos de transmisión (eventos 2, 4 y 5), que se sumarán conforme a la fórmula 7-6:

$$nº\ transmisiones = \sum \frac{86400}{período\ ocurrencia\ transmisión\ (s)}\ x\ duración\ batería\ (días) \qquad (7–6)$$

Posteriormente, hablaremos de las conclusiones de este estudio. Pero apreciar, en primer lugar, el **menor consumo** y la **mayor duración de la batería** (más de 70 días de batería extra), como consecuencia. Recordar que estamos en casos extremos de funcionamiento, donde se están dando todos los casos de uso. De darse sólo los eventos típicos (RTC, y alguna alarma de temperatura o batería), la diferencia sería mucho mayor.

7.2.2. Ancho de banda

No se indica, en las hojas de características de cada dispositivo, información exclusiva sobre el ancho de banda concreto que utilizan para la transmisión y recepción. Pero, en cambio, sí podemos indicar el **ancho de banda de frecuencias disponibles para la transmisión y recepción** en cada caso.

Para averiguar el ancho de banda utilizado por **GPRS**, se atenderá a las **bandas de frecuencia** para telefonía móvil utilizadas en España para la red **GSM** [44]. En concreto, nos fijaremos en la **banda 900 (UN-41)**, que se ubica en torno a los 900 MHz. En el caso de **Vodafone** (compañía a la que pertenece la tarjeta SIM que reside en el SIM900D), se proporcionan **2 bloques de 10 MHz de ancho** cada uno (904.9-914.9 MHz para subida, y 949.9-959.9 MHz para bajada).

En cuanto a **SigFox** se refiere, sólo se especifica que se utiliza la tecnología UNB, y se transmite en la banda de frecuencias libre de 868 MHz en España. Tanto para transmisión como para recepción, se utiliza un mismo **bloque de 1.7 MHz de ancho** (de 868 a 869.7 MHz), si comprobamos las características del TD1204 en su datasheet. En todo caso, es un valor **menor** al empleado en GPRS.

7.2.3. Alcance

El **alcance** en GSM/GPRS dependerá del escenario en el que nos encontremos, necesitando labores de planificación, instalación y mantenimiento de las infraestructuras de telecomunicación empleadas. Típicamente, las **celdas** empleadas tienen un radio de **cientos de metros en zonas urbanas** muy pobladas, y de hasta **30 kilómetros en zonas rurales** [45].

En cuanto a SigFox, se estimó el alcance en un rango de 3**-10 kilómetros en áreas urbanas**, y de **30-50 kilómetros en áreas rurales**; valores mayores que para el caso de GSM/GPRS, aunque recordamos que dependerá de la red desplegada.

7.2.4. Costes de datos

El SIM900D viene equipado con una **tarjeta SIM** de Vodafone, aplicándose una tarifa de **2 euros al mes**, que incluye el uso de **2 MB** (también al mes) para la **transmisión de datos**. En total, cuesta **24 euros al año**.

En cuanto al TD1204, se utiliza una suscripción de **140 mensajes al día** por **14 euros al año**. Al margen de la cantidad de información disponible para transmitir, supone un **ahorro de 10 euros por dispositivo, al año**.

7.2.5. Costes de materiales

Por último, contemplaremos el **coste de los materiales**, considerando únicamente el precio del microcontrolador y de las baterías; el resto de materiales se considerarán comunes para ambos dispositivos, luego no se computarán en la suma.

El SIM900D se puede encontrar a la venta por distintos precios. La última consulta a esta página [46] indicaba un precio de **17.74 euros por unidad.** Se utilizan **2 baterías** del modelo **ER26500M** [43]. A 9.33 por unidad, tendríamos **18.66 euros en baterías**, por dispositivo. Esto hace un total de **36.4 euros** por dispositivo.

En el caso de la familia de módems TD12xx, se tienen los siguientes precios (en dólares), proporcionados por Telecom Design:

TELECOM DESIGN — AVNET Memec

Module	Description		TD P/N	packaging	<1Ku	<10Ku	<25Ku	>25Ku
TD1208	Sigfox 868Mhz module Cortex M3, 128KB Flash + 16KB RAM		PROD0756	24 units sealed ESD trays	$16.90	$15.50	$14.30	special quote
TD1204	Sigfox 868Mhz module Cortex M3, 128KB Flash + 16KB RAM GPS + Accelerometer, w/o antenna		PROD0762A	20 units sealed ESD trays	$34.00	$31.50	$29.00	special quote
TD1205	Sigfox 868Mhz module Cortex M3, 128KB Flash + 16KB RAM GPS + Accelerometer, integrated antennas		PROD0767	15 units sealed ESD trays	$52.00	$50.00	$47.00	special quote

EVB & SDK	Description		TD P/N	packaging	price
EVB TD1204	TD1204 Evaluation Board Antenna Sigfox subscription		PROD0778	1 unit	$189.00

Figura 7-5. Rango de precios para la familia de módems TD12xx. *Proporcionado por Telecom Design.*

Si nos fijamos en el precio más caro del TD1204, tenemos 34 dólares, que equivalen a **30.73 euros**.

En cuanto a la batería utilizada; podríamos recurrir a la misma que emplea el SIM900D, pero no merece la pena. El pico de intensidad detectado durante la ejecución del TD1204 está en torno a 100 mA, luego bastaría con utilizar una **batería de energía**, en vez de potencia, que proporcione más capacidad y menos intensidad de pico límite (que, recordamos, estaba en torno al amperio para el SIM900D). Un modelo posible sería el **ER26500** [47], con un valor de **6.92 euros por unidad**, y una capacidad de 9000 mAh, que soporta picos de intensidad de hasta 300-500 mA.

Si volvemos a realizar el estudio de consumo anterior utilizando esta batería, y considerando las mismas medidas realizadas anteriormente salvo las alarmas de batería y temperatura (el evento de transmisión de alarmas cambiaría a 3600 segundos de período de ocurrencia, y a 168 segundos de duración por día; ya que se enviaría 1 alarma cada hora, correspondiente a RTC), se obtendrían estos resultados:

Consumo medio (µA)	249,81
Duración batería (h)	32424,49
Duración batería (días)	1351,02
Duración batería (años)	3,70
Número transmisiones	37829

Tabla 7–6. Valores de consumo del TD1204, con cambio de batería y eliminación de alarmas de batería y temperatura.

Vemos que la duración de la batería se dispara a más de 3 años y medio, y contando que se producen casos extremos como detección de movimiento o cambio de posición. Por lo tanto, queda justificado el cambio de batería.

El precio total sería de **37.65 euros**. Una diferencia de 1.25 euros por dispositivo, siendo el TD1204 más caro que el SIM900D. Aunque, en el cómputo global, el ahorro de 10 euros al año en la tarifa de datos por dispositivo compensa esta pérdida.

7.3. Ventajas y inconvenientes del uso del TD1204 frente al SIM900D

Aunque ya se han comentado, brevemente, las **ventajas e inconvenientes del cambio de GPRS a SigFox**, con el estudio de dos módems concretos, se quiere incidir en varios **detalles** de cada punto tratado:

- El **consumo es radicalmente menor** en el TD1204. Vemos que consume 20 veces menos en reposo, y prácticamente la mitad en transmisión (si contamos que el GPS está apagado). Si omitimos el uso del GPS, que provoca un aumento del consumo de la placa, podemos comprobar que **la vida útil de la batería se alargaría considerablemente**. En el estudio de la tabla 7-6, pudimos comprobar que la eliminación de las alarmas de temperatura y batería alargaban la vida útil en casi 1 año. De omitir el uso del GPS y limitarnos a la transmisión de RTC, podríamos hablar de valores superiores a los 8 años de duración de batería.
- No se pueden extraer grandes conclusiones en cuanto al **ancho de banda** se refiere, pero sí se puede comprobar que la **banda** utilizada por **SigFox** para **transmisión y recepción es la misma**. Además; si bien su ancho es mucho menor que el utilizado por Vodafone en GPRS, pueden darse problemas de **colisión** al trabajar en una banda libre de frecuencias, donde hay multitud de dispositivos transmitiendo información.
- El **alcance** es prácticamente similar, aunque las cifras de SigFox son mayores al disponer de cifras más o menos exactas sobre la infraestructura que ya está montada. GSM/GPRS da datos más conservadores; aunque, como ya se comentó, dependerá de la instalación final de la red el valor final del alcance, que puede superar al de SigFox.

  También se quiere hablar sobre la **cobertura**, dentro de este apartado. **GSM/GPRS** tiene **cobertura prácticamente total** en España. **SigFox**, en cambio, todavía está en **proceso de expansión**. Aunque presenta buena cobertura en las grandes ciudades, todavía quedan zonas que no tienen la suficiente cobertura. Podemos ver el **mapa de cobertura** de cada opción en las siguientes figuras:

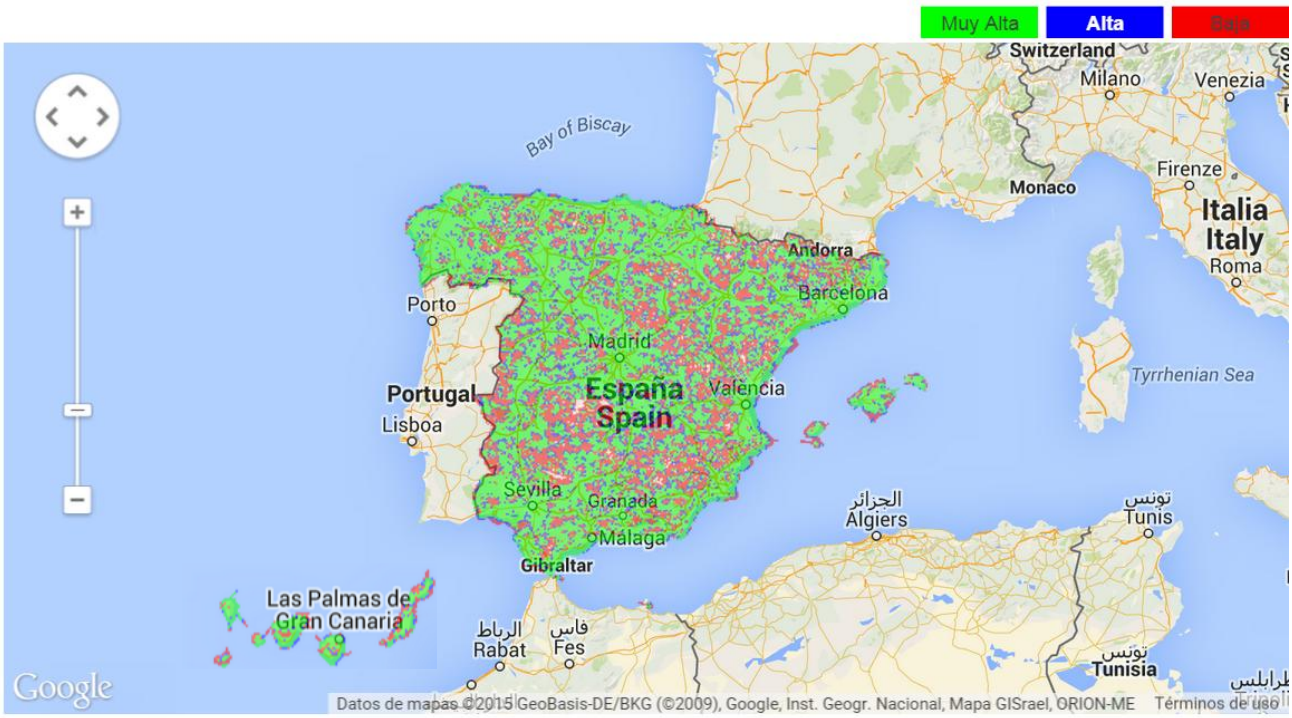


Figura 7-6. Mapa de cobertura de GSM/GPRS. *Página oficial de Vodafone España.*

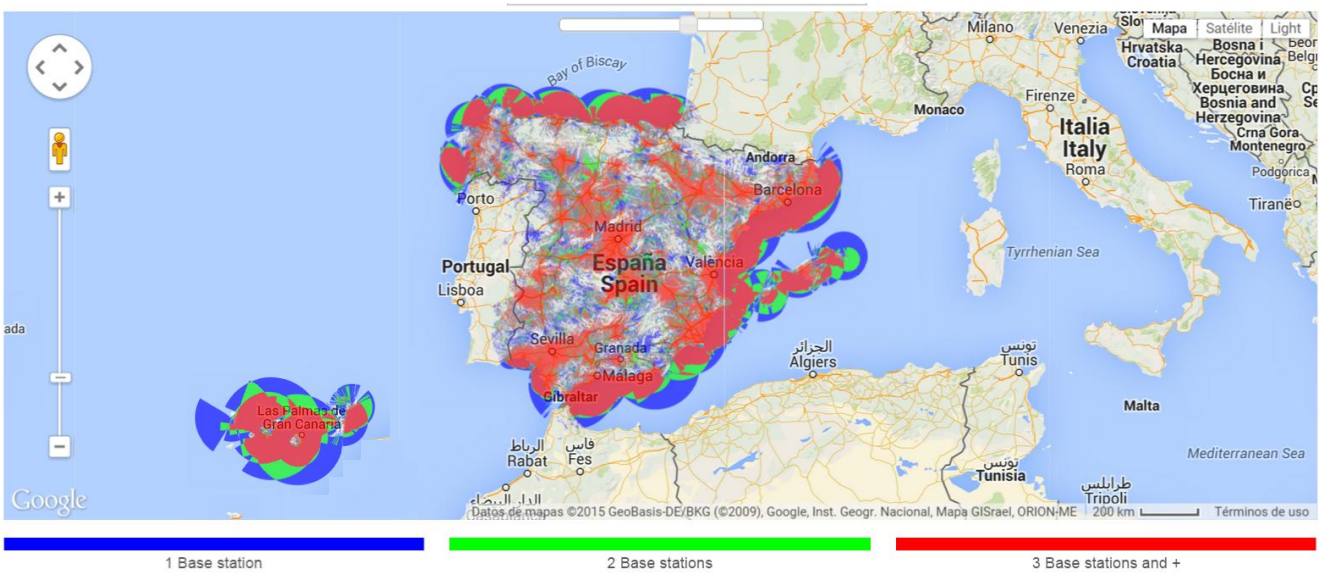


Figura 7-7. Mapa de cobertura de SigFox. *Backend de SigFox.*

- En **coste**, combinando datos y materiales, vuelve a ganar SigFox. Como **ventaja**, presenta la posibilidad de uso de **baterías más baratas**, al tener una intensidad de pico menor. Estas baterías son **de energía**, como ya se dijo en el apartado correspondiente: baja corriente de pico límite, alta capacidad, y menor precio. En cambio, los módems GPRS necesitan baterías que permitan corrientes de pico mayores, siendo baterías de potencia; con menor capacidad y mayor precio.

  En cambio, se presenta la **desventaja** de tener un **límite de mensajes al día**, y **posibilidades insuficientes**, a día de hoy, de **comunicación bidireccional**. GPRS sí que permite el flujo bidireccional de datos, y 2 MB al mes para transmitir mensajes del orden del byte de tamaño son más que suficientes. Aunque, para los casos de uso que se han presentado, se puede suponer que el límite de mensajes de SigFox no supondrá problemas a la hora del envío de información.

7.4. Conclusiones finales

Aparte de las evidentes conclusiones que se pueden extraer a partir de los resultados de **rendimiento** del funcionamiento del TD1204; principalmente, **bajo consumo, aumento de la vida útil de la batería** y **ahorro económico**, se quiere hacer un análisis de un ámbito más *filosófico* sobre el trabajo realizado.

Si se ve el contenido global del proyecto, podemos distinguir, claramente, las 3 **ramas** principales de la telecomunicación, manifestadas en los siguientes conceptos:

- **SigFox**, como ejemplo de red, sistema e infraestructura de **telecomunicación**.
- La placa de evaluación **TD1204** y su manejo, donde interviene la **electrónica**.
- El proceso de **programación** en C y configuración completa del **servidor** LAMP, que es parte de la rama **telemática**.

La combinación de todos estos elementos ha dado, como fruto, este estudio de viabilidad sobre la red SigFox y sus dispositivos certificados, con el TD1204 como caso práctico de estudio. La **migración** a un nuevo entorno compuesto por transceptores de la red SigFox parece interesante a raíz de los resultados obtenidos en las comparativas.

Se recuerda que SigFox, todavía, está en expansión, tanto en España como en otros países, y hay abiertas muchas líneas de investigación para su introducción en numerosos ámbitos de aplicación. Aunque es evidente que supone una **oportunidad de negocio** con cada vez más peso entre grandes empresas del sector tecnológico.

7.5. Líneas futuras

A lo largo del desarrollo de la memoria, se han dejado abiertos varios caminos, que ampliarían el estudio de varios de los puntos tratados en el proyecto. Se mencionan algunos de ellos, que marcarían **nuevas vías de estudio** y **líneas futuras de desarrollo**:

- La transmisión considerada para el TD1204 en el proyecto ha sido, exclusivamente, unidireccional. SigFox ya permite la **comunicación bidireccional** desde varios meses atrás, con nuevas licencias de suscripción. Aunque el sentido descendente de la comunicación está muy limitado, permitiéndose únicamente el envío de menos de 10 mensajes al día, podría resultar útil su uso para enviar **mensajes de asentimiento** al dispositivo, en caso de querer confirmar la transmisión correcta de mensajes con información sensible.
- Se han obviado, en este proyecto, los aspectos de **seguridad**. Ya se comentó en los primeros apartados que, de captar el mensaje y descubrir su codificación, podría leerse, rompiendo la confidencialidad de la comunicación. Habría que estudiar, con detalle, qué medidas de seguridad concretas se utilizan en la comunicación del TD1204, evaluando los riesgos presentes, y buscando soluciones para minimizar el daño que podrían causar esas amenazas.
- Se ha dejado de lado la **plataforma Sensor**, que ofrecía una API completa para la monitorización de dispositivos de la familia TD. Aunque de difícil comprensión, su estudio podría facilitar las labores de supervisión del dispositivo y de gestión de callbacks.
- También se cerró la vía de **Arduino**, que actuaba como intermediario entre la placa de evaluación y el ordenador, siendo el dispositivo sobre el que se programaría la aplicación. Un mejor estudio de las posibilidades de comunicación Arduino-TD1204 podrían permitir que una aplicación, similar a la codificada para el TD1204, realizase las mismas funciones. El problema de esta opción es la falta de documentación al respecto de esta comunicación, lo cual complicaría este estudio.
- Se ha considerado, únicamente, la comunicación de un dispositivo con el backend. En entornos reales, se debería plantear el **despliegue de una red de sensores** completa, compuesta por grupos de dispositivos que se comuniquen entre sí y con el backend. Para ello, se debería plantear la configuración de la **comunicación entre dispositivos en local**, y de los **grupos de dispositivos** que

interactúen con el backend de la misma forma.

- En caso de entrar en una red de sensores, el **servidor de procesamiento de datos** debería **ampliarse** para poder gestionar la información generada por todos los dispositivos. Podrían ser interesantes, por tanto, labores de monitorización por dispositivo o grupos de dispositivo, o la generación de estadísticas globales.
- Entrando en un nivel mayor de complejidad, podría estudiarse la posibilidad de utilizar esta familia de módems (el TD1204 en concreto) como **encaminador de mensajes a la red SigFox**, recibiendo los **mensajes** a enviar al backend **vía radio**.

8 PLANIFICACIÓN TEMPORAL

"El hombre que se prepara, tiene media batalla ganada"

- Miguel de Cervantes, escritor -

Como última sección de este proyecto, se concluirá con la **planificación** seguida para el desarrollo del éste, con información detallada sobre las **fechas** que han abarcado cada fase de realización del proyecto, las **horas empleadas** en cada una, **comentarios** acerca del contenido tratado en dichas fases, y el plasmado de toda esta información en un **diagrama de Gantt**.

Para ello, se ha hecho uso de la herramienta online de administración de proyectos conocida como **Smartsheet** [48], que ofrece todas las funcionalidades antes mencionadas.

8.1. Reparto de horas del trabajo

Para la realización de este proyecto, se han empleado **325 horas de trabajo** (notar que el TFG equivale a 12 créditos, que son 25*12=300 horas de trabajo), repartidas en las siguientes **fases**:

Nombre de la tarea	Fecha de Inicio	Fecha final	Duración	Predecesores	Comentarios
1. Asignación final del proyecto e ideas previas	**26/02/15**	**01/03/15**	**4**		**6 horas. Queda asignado el proyecto de manera definitiva. Se inicia el estudio del estado del arte asociado al ámbito del proyecto.**
1.1. Reunión inicial con el tutor	26/02/15	26/02/15	1		1 hora. Adjudicación definitiva del proyecto y definición de los primeros pasos a realizar.
1.2. Estudio del estado del arte	27/02/15	01/03/15	3	1.1.	5 horas. Sin tener todavía la especificación del proyecto ni las herramientas, se investiga sobre las tecnologías con las que se trabajarán (en concreto, estudio de la red SigFox, y repaso de la documentación de la placa de evaluación).
2. Definición y alcance del proyecto	**02/03/15**	**12/03/15**	**11**		**19 horas. Primer acercamiento a la dimensión del proyecto. Estudio más concreto de las tecnologías a utilizar. Primera especificación de requisitos.**

2.1. Primera reunión en Wellness Smart Cities	02/03/15	02/03/15	1		2 horas. Se comenta la dimensión del trabajo y las primeras tareas a realizar, entrando en mayor profundidad.
2.2. Estudio de IoT y SigFox	03/03/15	08/03/15	6	2.1.	10 horas. Se profundiza en los aspectos de IoT y SigFox que interesarán para este proyecto.
2.3. Estudio inicial de la placa de evaluación	09/03/15	12/03/15	4	2.2.	5 horas. Se revisa la documentación de la placa y el software disponible (aún sin contar con la placa), y se estudian sus posibilidades a priori. Registro en Bitbucket, donde estará el código del proyecto.
2.4. Especificación de requisitos del proyecto	11/03/15	12/03/15	2		2 horas. Se entrega, por correo, la especificación inicial de requisitos del proyecto. Se procede a estudiarlo y a hilar los conceptos descritos con los ya trabajados. En base a este documento, se perfila un índice provisional para el proyecto, a la espera de cerrarlo con los aspectos que todavía faltan por trabajar.
3. Introducción a la placa de evaluación	**13/03/15**	**26/03/15**	**14**		**32 horas. Trabajo con la placa de evaluación, estudiando sus características básicas de configuración.**
3.1. Segunda reunión en Wellness Smart Cities	13/03/15	13/03/15	1		2 horas. Se hace entrega de la placa, y se prueba el conexionado y el acceso a la misma desde el PC, funcionando correctamente. Carga del firmware y prueba de comandos AT.
3.2. Estudio de la placa. Comandos AT	14/03/15	18/03/15	5	3.1.	10 horas. Ya con la placa, se estudian las posibilidades de la misma en base a los comandos AT que se pueden utilizar. Se comprueba el funcionamiento de cada uno, y se anotan los que servirán de utilidad para la especificación del proyecto.
3.3. Pruebas con la placa. AT, GPS y backend	19/03/15	26/03/15	8	3.2.	20 horas. Estudiados los comandos, se prueban sobre la placa, y se trabaja con el backend de SigFox, que recibe los mensajes enviados. Estudio de posibilidades y limitaciones. No se realizan programas para la placa, todavía.
4. Integración de la placa con Arduino	**27/03/15**	**06/04/15**	**11**		**12 horas. Trabajo con una alternativa para la programación de la placa, con Arduino.**
4.1. Tercera reunión en Wellness Smart Cities	27/03/15	27/03/15	1		2 horas. Resolución de dudas. Se entrega un Arduino One y se establece el conexionado entre la placa de evaluación y el Arduino para su uso. Se prueban aplicaciones de ejemplo, con resultado correcto. Se especifica un diagrama de estado de la funcionalidad que debe implementar la placa finalmente.

4.2. Estudio de las posibilidades de Arduino	28/03/15	05/04/15	9	4.1.	10 horas. Realización de código para pasar los comandos AT a la placa vía Arduino. Investigación de otros códigos de ejemplo (librería SoftwareSerial). Anotación de dudas y problemas. No se avanza en la programación de la placa.
4.3. Código para comunicar Arduino con la placa usando el puerto serie	06/04/15	06/04/15	0	4.2.	Subida del código que permite pasar comandos AT a la placa vía Arduino en el repositorio Bitbucket, ya que no se va a avanzar más en el estudio de Arduino, por el momento.
5. Evaluación de alternativas a Arduino	**07/04/15**	**27/04/15**	**21**		**13 horas. Búsqueda de alternativas, distintas a Arduino, para la programación de la placa.**
5.1. Estudio de alternativas para programación de la placa	07/04/15	14/04/15	8		8 horas. Se buscan otras maneras de programar la placa independientes de Arduino, ya que su uso parece dificultar el uso de la placa (no se encuentran recursos para usar Arduino con la placa), y los comandos AT no consiguen un uso autónomo de la placa. Planteamiento de dudas relativas a estos problemas al tutor del TFG y de Wellness Smart Cities.
5.2. Planteamiento de dudas y búsqueda de nuevas opciones para la programación	15/04/15	27/04/15	13	5.1.	5 horas. Se plantean alternativas para programar la placa y otras opciones; como la simulación de casos de prueba, o captura de mensajes con un receptor radio. Se estudiarán a lo largo del período de feria por parte de ambos tutores y por el alumno.
6. Estudio de programación de la placa con Eclipse	**27/04/15**	**21/05/15**	**25**		**38 horas. Trabajo con Eclipse para la programación de la placa. Estudio y carga de ejemplos incorporados en el SDK.**
6.1. Estudio de documentación para la integración de Eclipse	27/04/15	03/05/15	7		15 horas. Se encuentra un repositorio en Github que permite la integración de código de ejemplo en C en Eclipse. Se siguen los pasos para programar la placa con Eclipse y se prueban programas de ejemplo, sin éxito.
6.2. Cuarta reunión en Wellness Smart Cities	04/05/15	04/05/15	1		1 hora. Revisión del ejemplo más simple y prueba en la placa, sin éxito. Se decide prescindir del Arduino y seguir trabajando con el repositorio antes comentado. Se continuará con la investigación por este nuevo camino.
6.3. Carga de programa de ejemplo en la placa	05/05/15	10/05/15	6	6.2.	6 horas. Se consigue, finalmente, cargar el programa más básico en la placa, aunque sobreescribe la memoria y no permite cargar otro. Se estudia brevemente el resto de código presente en los ejemplos.
6.4. Quinta reunión en Wellness Smart Cities	11/05/15	11/05/15	1		1 hora. Se concluye que hace falta un reset forzado de la placa para poder cargar otro programa. Se proporciona el material necesario para ello y se consigue con éxito, pudiendo cargar otros programas.

6.5. Estudio del resto de ejemplos del código para Eclipse	12/05/15	21/05/15	10	6.4.	15 horas. Trabajo con el resto del código; sobre todo en aspectos de comprensión del mismo, sin entrar en profundidad, para seleccionar los ejemplos adecuados para reutilizarlos en el código que se pretende codificar para el proyecto.
7. Trabajo de documentación	**22/05/15**	**24/05/15**	**3**		**11 horas. Inicio de la documentación, fijando índice, planificación futura y plantilla de la memoria.**
7.1. Especificación del índice definitivo del proyecto	22/05/15	22/05/15	1		2 horas. Revisando la documentación ya conseguida, y valorando los puntos trabajados en los últimos meses, se define el índice final del proyecto.
7.2. Planificación del desarrollo temporal del proyecto	23/05/15	23/05/15	1	7.1.	3 horas. En base al trabajo ya realizado, se especifica el desarrollo temporal de este proyecto en las semanas que restan hasta su entrega, con las tareas a realizar cada semana y los hitos a alcanzar.
7.3. Preparación de la memoria	24/05/15	24/05/15	1	7.2.	6 horas. Se plasma, en la memoria, el índice antes mencionado (pendiente de aprobación), y se procede a organizar toda la información ya conseguida en base a los apartados donde se incluirán. También se adecua el documento tipo a la memoria de este proyecto (lo cual llevó más tiempo, al trabajar en Mac y no Windows, y descuadrarse parte del formato que llevaba incorporado el documento).
8. Programación del código final en Eclipse	**25/05/15**	**19/06/15**	**26**		**39 horas. Trabajo de codificación de la aplicación para la placa.**
8.1. Selección de ejemplos	25/05/15	27/05/15	3		4 horas. Se eligen ejemplos ya implementados que resultan de interés para realizar el código final.
8.2. Estudio de integración de servidor web para recopilar datos	27/05/15	28/05/15	2		2 horas. Se estudia la posibilidad de utilizar un servidor web al que redirigir los mensajes que lleguen al backend mediante callbacks, que enviarían una cadena en JSON a dicho servidor, y con su respectivo tratamiento permitiría mostrar una página con estadísticas de uso.
8.3. Sexta reunión en Wellness Smart Cities	29/05/15	29/05/15	1		1 hora. Se comentan los últimos estudios (en referencia al código seleccionado, a la posibilidad de uso de estadísticas en un servidor, y a la imposibilidad de trabajar con la placa como gateway para este proyecto). Se estudia el diagrama de estados que seguirá el programa principal (que constará de un solo fichero), y los aspectos de la placa a tener en cuenta (RTC, Watchdog, interrupciones, etc.)

8.4. Adaptación de ejemplos al código final	30/05/15	18/06/15	20	8.3.	32 horas. Se complementa el diagrama de estados estudiado en la reunión con los ejemplos ya estudiados, anotando las funciones y librerías de interés para realizar los casos pedidos. Se modifica dicho diagrama en función de los problemas encontrados durante la codificación, y se termina la programación de todos los casos de uso de interés.
8.5. Código completado	19/06/15	19/06/15	0	8.4.	Código para la placa completado, en su versión 0.6. Falta revisar los detalles de implementación y probar de forma real cada caso de uso, además de medir el consumo de la placa.
9. Fin de la codificación y generación de documentación Doxygen	**19/06/15**	**24/06/15**	**6**		**20 horas. Revisión de errores en el código y corrección de los mismos. Generación de documentación Doxygen.**
9.1. Séptima reunión en Wellness Smart Cities y con el tutor	19/06/15	19/06/15	1		2 horas. En Wellness Smart Cities, se comprueba el código y se valida, a falta de retoques finales. Se prueba la redirección de mensajes del backend a una URL de prueba, funcionando también. Queda pendiente para la próxima reunión medir el consumo de la placa, y cambiar algunos detalles de implementación. Con el tutor, se comentan estos aspectos, y se deja para la semana que viene la primera entrega de la memoria y el comienzo del servidor para estadísticas.
9.2. Revisión del código y pruebas	20/06/15	20/06/15	1	9.1.	10 horas. Corrección de detalles en la implementación del código. Se reúnen las librerías base que han servido para hacer el código para añadirlas a la documentación. Revisión de comentarios y preparación de la plataforma Doxygen para la documentación. Se comprueba que todos los casos de uso se cumplen y se notifican al backend, recibiéndose los mensajes correctamente.
9.3. Documentación en Doxygen	21/06/15	21/06/15	1	9.2.	5 horas. Uso de la herramienta doxywizard (Windows) para generar la documentación Doxygen.
9.4. Octava reunión en Wellness Smart Cities y con el tutor	23/06/15	23/06/15	1		3 horas. En Wellness, se revisa el contenido actual de la memoria, aprobando su contenido y aportando pautas para los puntos restantes. Se mide el consumo de la placa con la aplicación, y se toman las primeras muestras. Con el tutor, se revisa y aprueba el contenido actual de la memoria, se proporciona acceso al servidor que proporcionará los resultados en tiempo real, y se ultiman detalles del código y la documentación Doxygen.

9.5. Código y documentación Doxygen definitivos	24/06/15	24/06/15	0	9.2., 9.3., 9.4.	Fin de la codificación con su versión 1.0, y documentación completa en Doxygen realizada, tras los detalles vistos en las reuniones. Subida de todo el proyecto al repositorio Bitbucket.
10. Procesamiento de datos con servidor LAMP	**24/06/15**	**30/06/15**	**7**		**20 horas. Trabajo de preparación y codificación de la aplicación para el servidor de procesamiento de datos.**
10.1. Puesta en funcionamiento del servidor LAMP	24/06/15	28/06/15	5		18 horas. Instalación del servidor LAMP en una máquina Debian 8.1. Verificación de funcionamiento. Inclusión de la documentación Doxygen. Inclusión y configuración de phpMyAdmin. Construcción de base de datos, usuarios, tablas y pruebas tanto aisladas como con los callbacks del backend de SigFox, con resultado final correcto.
10.2. Finalización del código del servidor	29/06/15	29/06/15	0	10.1.	Fin de la codificación del servidor, contando con el procesado y almacenamiento de datos, mostrado de estadísticas y hoja de estilo para la presentación de la aplicación web.
10.3. Novena y última reunión en Wellness Smart Cities	30/06/15	30/06/15	1		2 horas. Se completa el estudio del consumo de la placa y su comparación con GPRS. Se proporcionan plantillas para la comparativa a la hora de introducirla en la memoria.
11. Documentación final	**21/06/15**	**07/07/15**	**17**		**90 horas. Redacción de la memoria.**
11.1. Redacción de la memoria	21/06/15	06/07/15	16		90 horas. Inclusión de toda la documentación en la memoria.
11.2. Entrega de la memoria	07/07/15	07/07/15	0	11.1.	Entrega de la memoria en la secretaría de la escuela, una vez finalizada y aceptada por el tutor.
12. Preparación de la presentación	**08/07/15**	**17/07/15**	**10**		**25 horas (estimación). Trabajo de preparación para la presentación final del proyecto.**
12.1. Estudio de puntos a tratar en la presentación	08/07/15	08/07/15	1		2 horas. Revisión de todo el trabajo realizado, destacando qué puntos se tratarán con más o menos hincapié en la presentación del proyecto.
12.2. Preparación del material para la presentación	09/07/15	12/07/15	4	12.1.	15 horas. Realización de presentación Power Point para la presentación, recopilando toda la información destacada tras el trabajo del punto anterior, y utilizando capturas de pantalla para mostrar el funcionamiento de la aplicación en todas sus fases.
12.3. Práctica de la presentación	13/07/15	16/07/15	4	12.1., 12.2.	8 horas. Ejercicios de expresión oral para practicar la presentación.
12.4. Presentación del proyecto	17/07/15	17/07/15	0		Presentación del proyecto realizado. Se estima como fecha el 17 de julio por motivo de los 10 días que se dan de margen, desde la entrega del proyecto, para asignar la presentación.

Tabla 8–1. Fases de realización del proyecto. *Generado por Smartsheet.*

El reparto de horas se detalla con el siguiente **gráfico**:

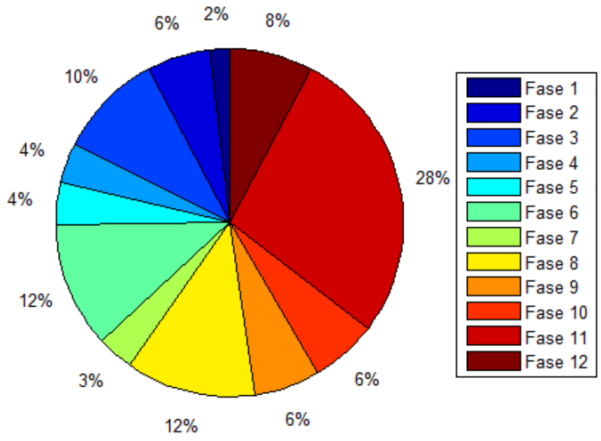
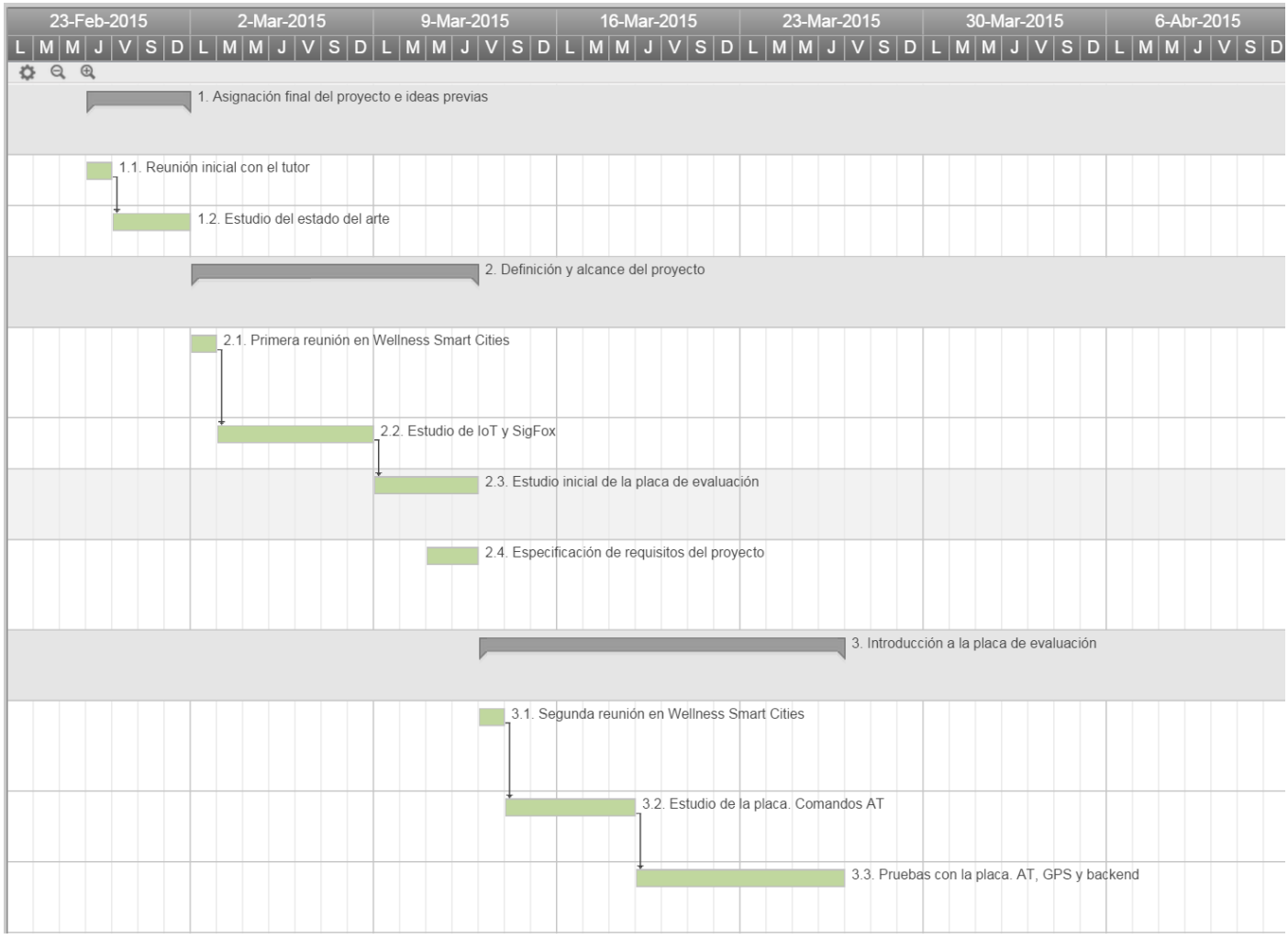
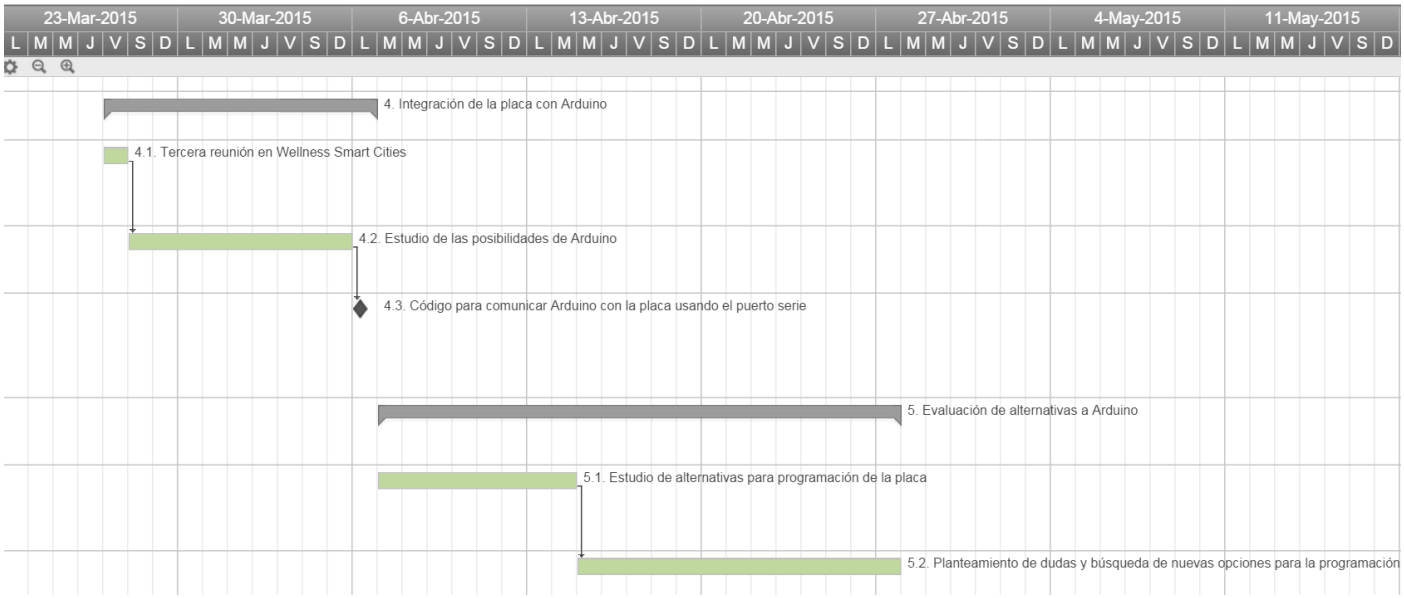


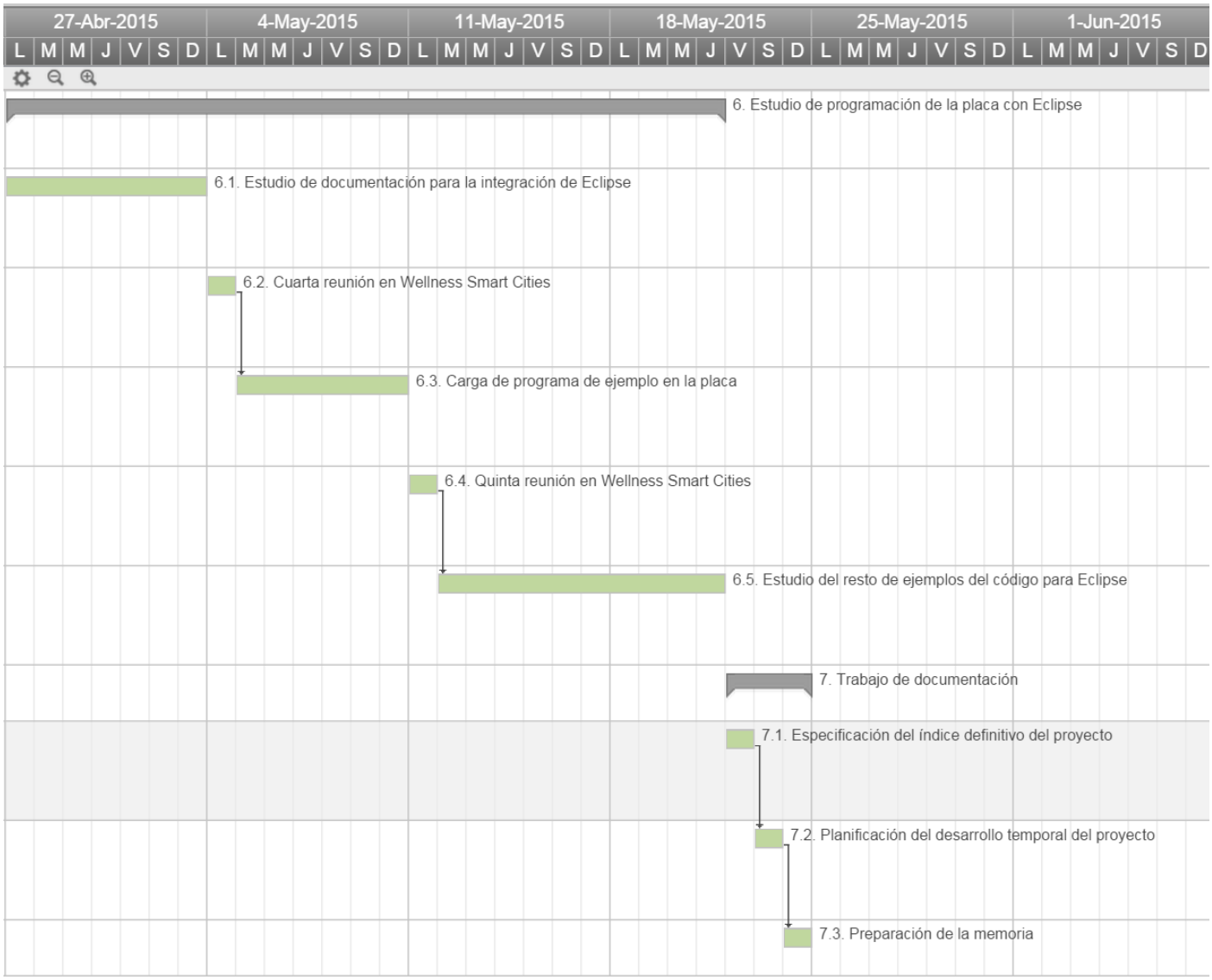
Figura 8-1. Reparto de horas del proyecto. *Generado por Matlab.*

8.2. Diagrama de Gantt

Las fases antes mencionadas se pueden representar gráficamente en el siguiente **diagrama de Gantt**.




23-Mar-2015
30-Mar-2015
6-Abr-2015
13-Abr-2015
20-Abr-2015
27-Abr-2015
4-May-2015
11-May-2015
L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D
4. Integración de la placa con Arduino
4.1. Tercera reunión en Wellness Smart Cities
4.2. Estudio de las posibilidades de Arduino
4.3. Código para comunicar Arduino con la placa usando el puerto serie
5. Evaluación de alternativas a Arduino
5.1. Estudio de alternativas para programación de la placa
5.2. Planteamiento de dudas y búsqueda de nuevas opciones para la programación


27-Abr-2015
4-May-2015
11-May-2015
18-May-2015
25-May-2015
1-Jun-2015
L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J V S D
6. Estudio de programación de la placa con Eclipse
6.1. Estudio de documentación para la integración de Eclipse
6.2. Cuarta reunión en Wellness Smart Cities
6.3. Carga de programa de ejemplo en la placa
6.4. Quinta reunión en Wellness Smart Cities
6.5. Estudio del resto de ejemplos del código para Eclipse
7. Trabajo de documentación
7.1. Especificación del índice definitivo del proyecto
7.2. Planificación del desarrollo temporal del proyecto
7.3. Preparación de la memoria

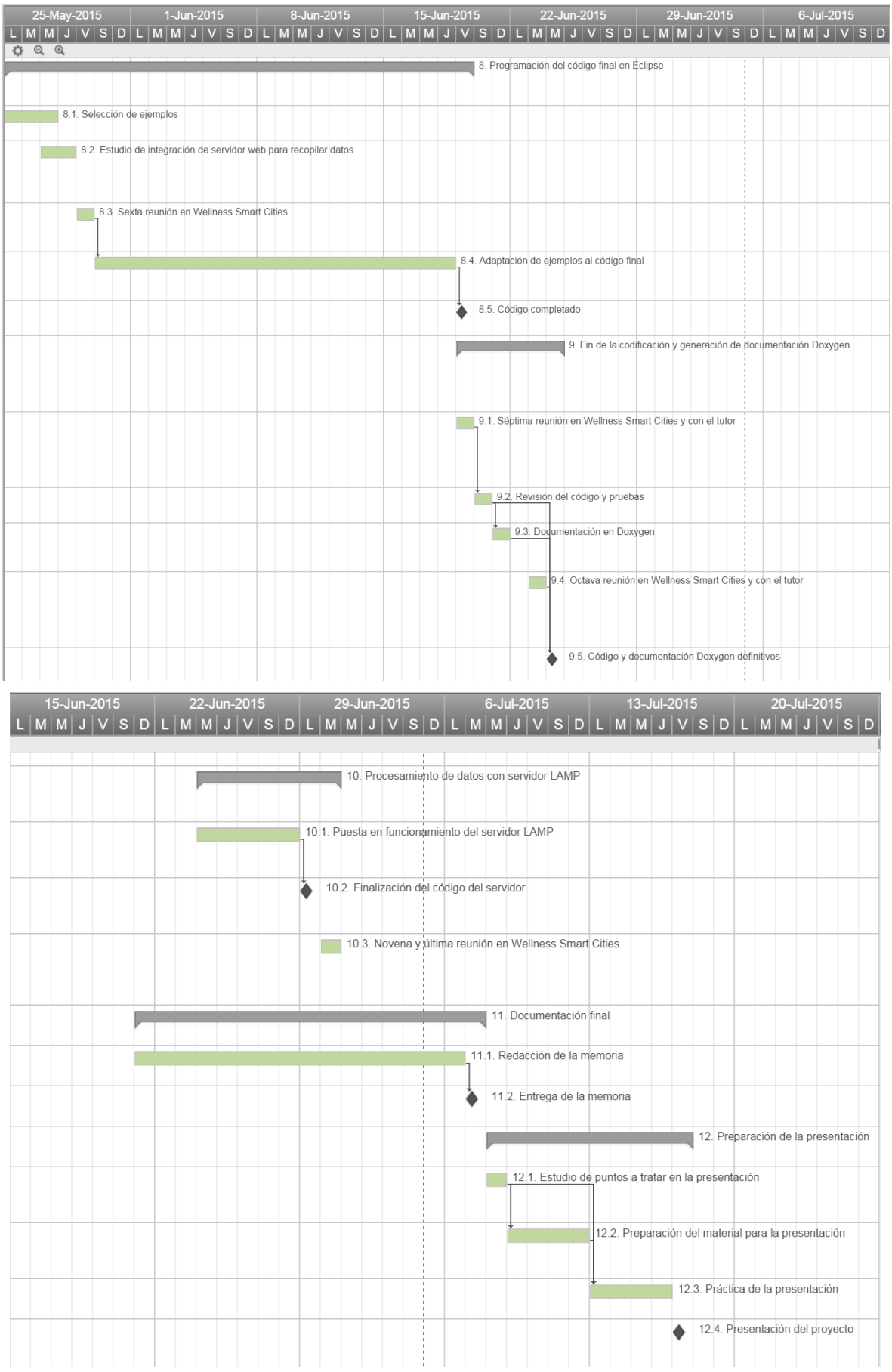


Figura 8-2. Diagrama de Gantt del proyecto. *Generado por Smartsheet.*

ANEXO A. PROCESO DE SUSCRIPCIÓN A SIGFOX

SigFox ofrece, a través de su backend [2], información relativa **al proceso de suscripción** al servicio, con los pasos a seguir para registrar los dispositivos en la red correctamente. Se enumeran estos pasos a continuación.

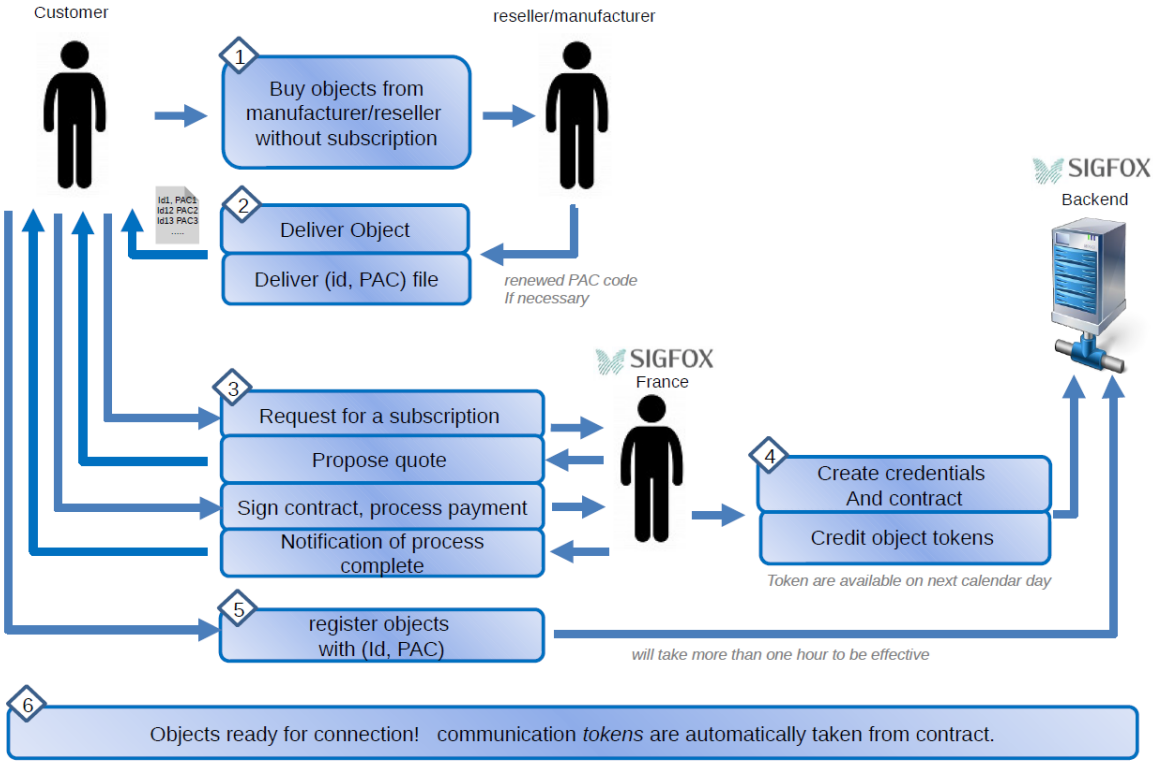


Figura A-1. Proceso de suscripción a SigFox. *Documentación del backend de SigFox.*

A.1. Obteniendo los dispositivos

En primer lugar, se deben adquirir los **dispositivos SigFox Ready** que se quieran conectar a la red. Estos equipos no llevan la suscripción a SigFox integrada, y deberán seguirse los pasos siguientes para desbloquear su acceso a la misma.

A.2. Identificando los dispositivos en SigFox

Para cada dispositivo, se necesitará un fichero que contenga una pareja de valores conocida como **ID-PAC**. Esta información es proporcionada por los proveedores de los dispositivos, teniendo cada una el siguiente significado:

- El **ID** actúa como **identificador único** del dispositivo, siendo un valor de 4 bytes representado en hexadecimal.
- El **PAC** sirve como **código de autorización** del dispositivo, y consta de un valor de 8 bytes representado también en hexadecimal. Este código actúa como un **certificado** que pertenece a cada objeto para ser validado tanto en el registro como en la transferencia de mensajes. Este código sólo se puede utilizar una vez; luego, una vez usado, se le asigna un nuevo código por parte del proveedor, si fuese necesario.

A.2.1. Obteniendo el fichero con el ID-PAC

Este fichero, entregado por el proveedor al cliente cuando adquiere los dispositivos, no se genera directamente, sino que sigue otro proceso en el que SigFox también hace acto de presencia.

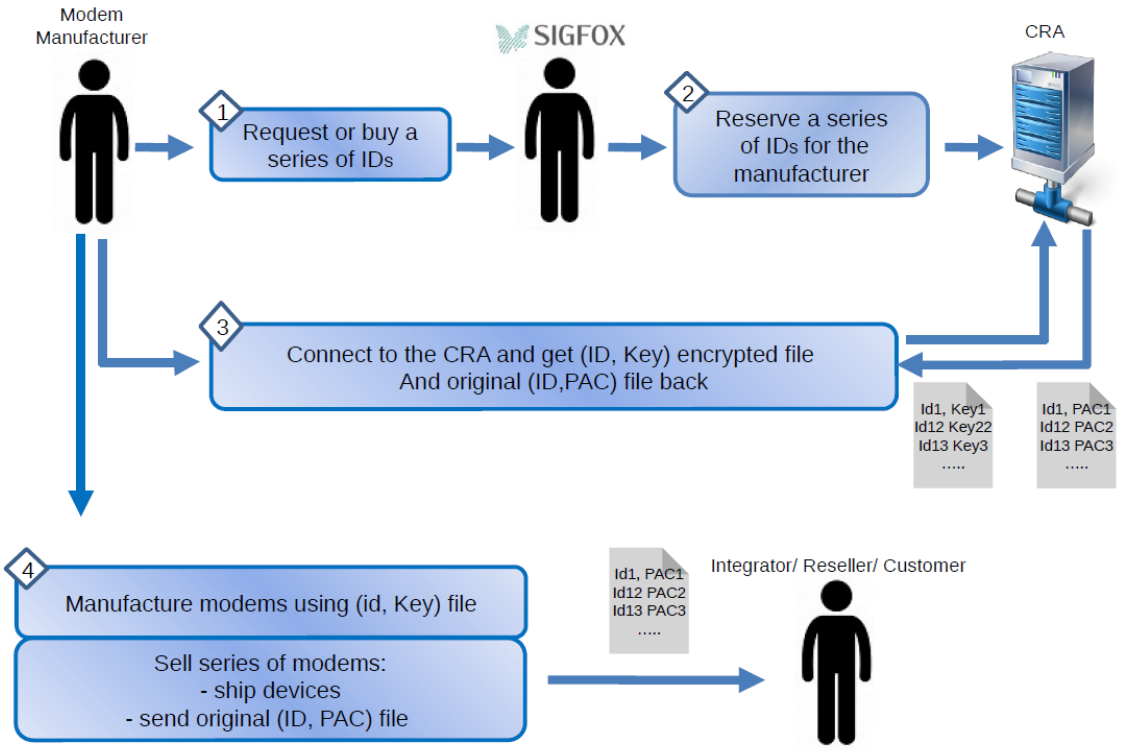


Figura A-2. Proceso de obtención del fichero con el ID-PAC. *Documentación del backend de SigFox*.

Destacar que **los ID son reservados por SigFox**, y entregados al proveedor junto al PAC. De ahí, pasan al cliente con el proceso antes comentado.

Además, el proveedor recibe una pareja ID-clave en un fichero encriptado, que puede utilizar para su propia plataforma e incluso entregarlo al consumidor de requerirse.

A.3. Firmando el contrato de suscripción

Este proceso es meramente **administrativo**, con una comunicación directa entre el cliente y SigFox, con el que se puede contactar a través del correo subscribe@sigfox.com para gestionar este proceso; que va desde la petición de suscripción hasta el contrato final, con el pago de las tasas oportunas y la notificación de proceso completado.

A.4. Accediendo al backend de SigFox

Una vez se completa el contrato de suscripción, SigFox hace entrega al cliente del **login** para el SigFox Cloud, teniendo acceso con él al backend para comenzar a hacer uso de los servicios de SigFox.

A.5. Registrando los dispositivos en SigFox

Una vez se tiene acceso al backend y se dispone del fichero con el ID-PAC de cada dispositivo, se procede a **registrarlos** en el backend para poder integrarlos en la red. Nuevamente, la misma documentación de SigFox nos describe este proceso paso a paso:

1. Accedemos a la pestaña **Device Type**, del menú superior, y pulsamos en el botón **New** de la esquina superior derecha para crear un nuevo tipo de dispositivos. Recordamos que este tipo permite identificar a un conjunto de dispositivos para ser tratados de la misma forma.

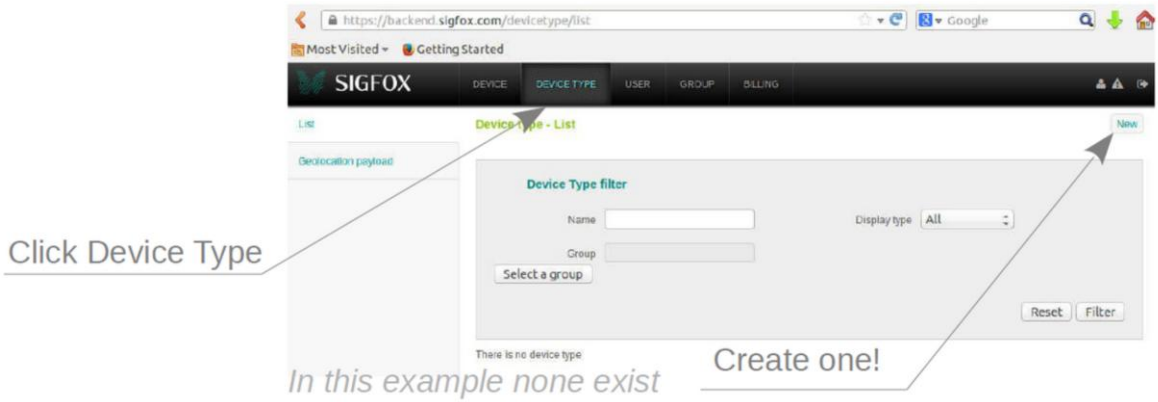


Figura A-3. Accediendo a la creación de un nuevo tipo de dispositivos. *Documentación del backend de SigFox*.

2. De ahí, **rellenamos el formulario** con los datos pedidos; obligatoriamente, el **nombre** del tipo, su **descripción** y el contrato de **suscripción** al que está asociado. El resto de parámetros podrán definirse más tarde.

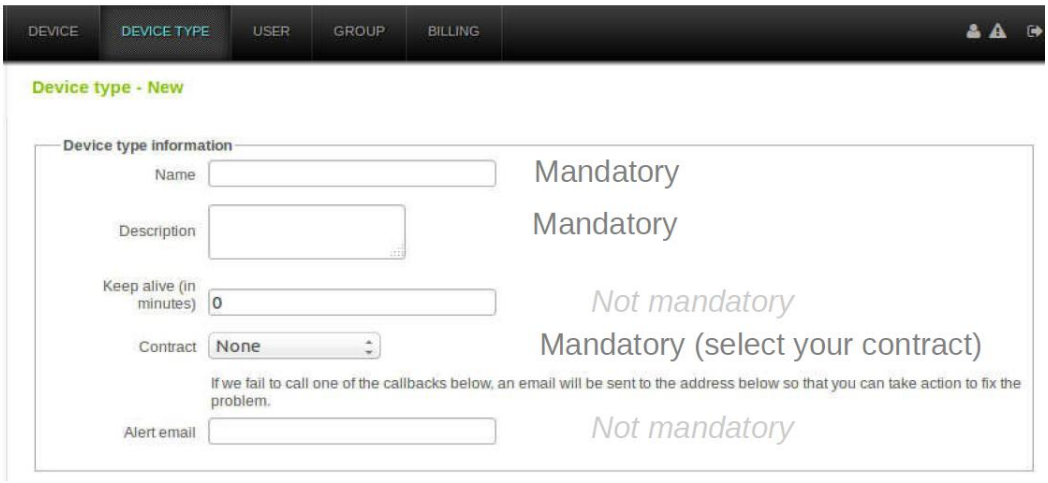


Figura A-4. Rellenando el formulario de nuevo tipo de dispositivos. *Documentación del backend de SigFox.*

3. Una vez creado el tipo de dispositivos, nos disponemos a **registrar los objetos** en el backend, accediendo a la pestaña **Device** y entrando en la opción **New series**, de la esquina superior derecha. Allí, deberemos proporcionar el **prefijo** del dispositivo y su fichero **ID-PAC** para identificarlo en el servicio.

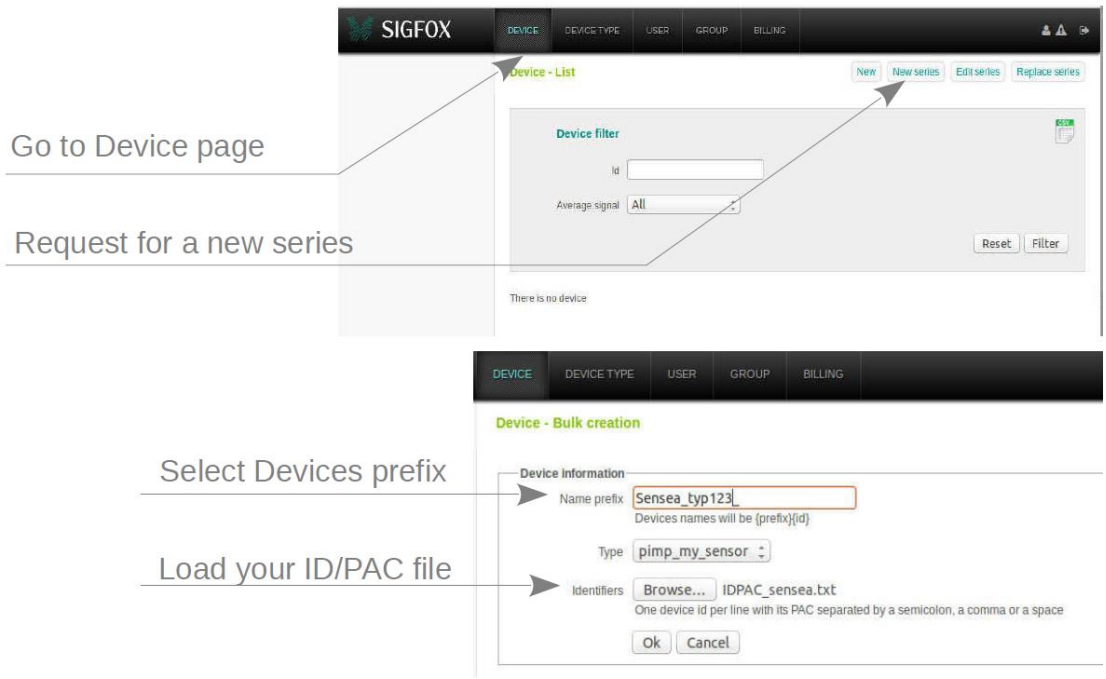


Figura A-5. Registrando un dispositivo en el backend. *Documentación del backend de SigFox.*

4. Finalmente, bastará **esperar a la respuesta del backend**, que indicará el resultado final del proceso de registro.

Figura A-6. Resultado de registro correcto de un dispositivo en SigFox. *Documentación del backend de SigFox*.

A.6. Listos para la comunicación

Una vez registrados los dispositivos, estarán correctamente identificados en la red SigFox y podrán hacer uso de la misma para la transmisión de mensajes, a la vez que pueden ser controlados y gestionados a través del backend.

Los siguientes pasos a seguir quedan a libertad del cliente para que elija la configuración que más le convenga; asociación de dispositivos al tipo de dispositivos previamente creado, generar nuevos tipos de dispositivos para distinguirlos de otros, etc.

ANEXO B. CONFIGURACIÓN DE ECLIPSE

Todos los **pasos necesarios para configurar Eclipse** correctamente y comenzar con la implementación de aplicaciones en las placas de evaluación TD12xx se encuentran en el directorio privado del repositorio e Telecom France en Github. Además de eso, también contiene el código fuente de las librerías y los ejemplos de uso [4].

Como dicho repositorio sólo puede ser accedido por desarrolladores previamente registrados, conviene mencionar explícitamente dichos pasos en este anexo, para la correcta configuración de Eclipse para compilar tanto las librerías y los ejemplos como la implementación desarrollada. Además de seguir los pasos marcados en el repositorio, se han seguido otros pasos marcados en este blog [37], siguiendo los hilos *Customize your SigFox TD1208 with SDK* y *Create a new SigFox Project*, que han facilitado la labor aquí detallada con consejos frutos de la experiencia con esta familia de módems.

B.1. Obteniendo Eclipse

Como se comentó en la sección del módem TD1204, France Telecom ofrece una versión adaptada de **Eclipse** para la familia de módems TD12xx (en concreto, para el TD1204 y el TD1208). Podemos encontrar dicho SDK en [3], en el apartado *Tools*. También podemos encontrar el SDK en [38]. En concreto, **la versión del SDK utilizada para este proyecto** ha sido **la 4.0.0.**

Una vez descargado el fichero (de unos 540 MB de peso), se debe **descomprimir obligatoriamente en el directorio C:**. De no hacerlo, la compilación del código fuente siempre fallará, y no se podrán obtener los ejecutables de las aplicaciones.

Tras esto, podemos acceder al **ejecutable de Eclipse** a través de la ruta **C:\TD\TD_RF_Module_SDK-v4.0.0\eclipse\eclipse.exe**.

Como paso previo a la configuración de Eclipse, se añadirán**variables de entorno** que incluyan la **ruta al compilador GCC**.Para ello, nos dirigimos al **Panel de control > Sistema y seguridad > Sistema > Configuración avanzada del sistema**. Allí, pulsamos en **Variables de entorno**, y añadimos tanto en las **variables de usuario** como en las **de sistema** la variable **PATH**, con valor **C:\TD\TD_RF_Module_SDK-v4.0.0\gnu\bin;C:\TD\TD_RF_Module_SDK-v4.0.0\gnu\arm-none-eabi\bin**.

Tras esto, reiniciaríamos el equipo, para que se aplique la configuración correctamente. Y, ahora sí, pasaríamos a la **configuración de Eclipse**.

B.2. Obteniendo el código fuente de las librerías

Para incluir las librerías del repositorio de Github, haremos uso de Eclipse para que las incorpore automáticamente, siguiendo estos pasos:

1. **Abrimos Eclipse**, a través de la ruta antes indicada.
2. En el menú superior, entramos en la opción **File** y pulsamos **Import…**; allí, elegimos la opción **Git > Projects from Git**. En **Select Repository Source**, elegimos la opción **URI**.
3. Allí, debemos completar el campo **URI** con el **enlace al repositorio**, que es https://github.com/Telecom-Design/TD_RF_Module_SDK.git. Los otros 2 campos se completarán automáticamente. En la parte inferior, se deberá añadir el **usuario y contraseña de Github** de la cuenta que tiene acceso al repositorio privado (el mismo que se sincronizó en el panel de control para desarrolladores en la página de desarrolladores de Telecom Design). Tras ello, pulsamos en **Next**.

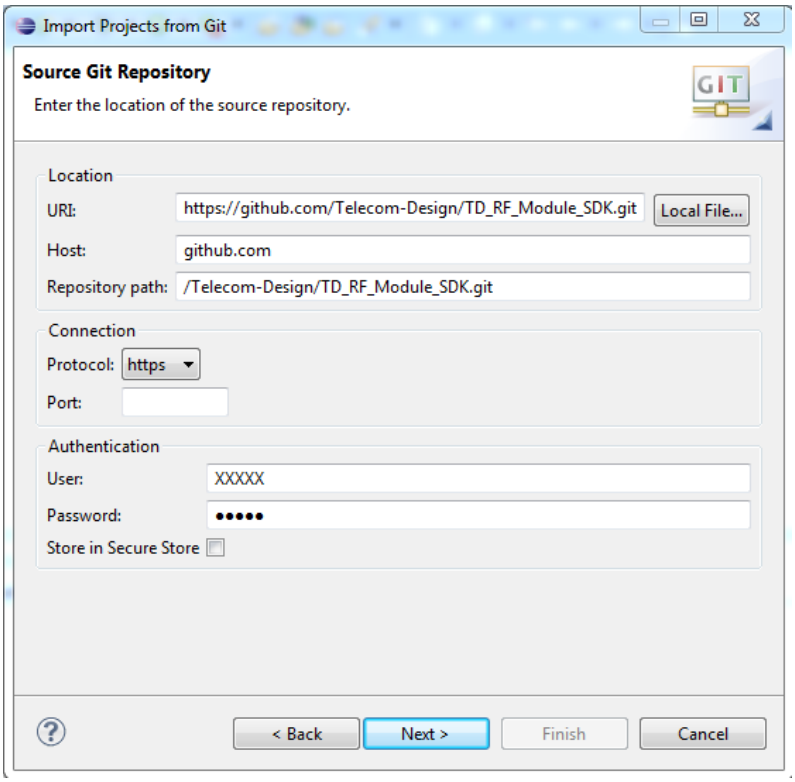


Figura B-1. Añadiendo el repositorio de Github con el código de las librerías.

4. Seleccionamos la rama **master**, y pulsamos en **Next**. En **Local Destination**, completamos **Directory** con la ruta **C:\TD\TD_RF_Module_SDK-v4.0.0\Github\TD_RF_Module_SDK**. Dejamos el resto de campos tal y como están, y pulsamos **Next**.

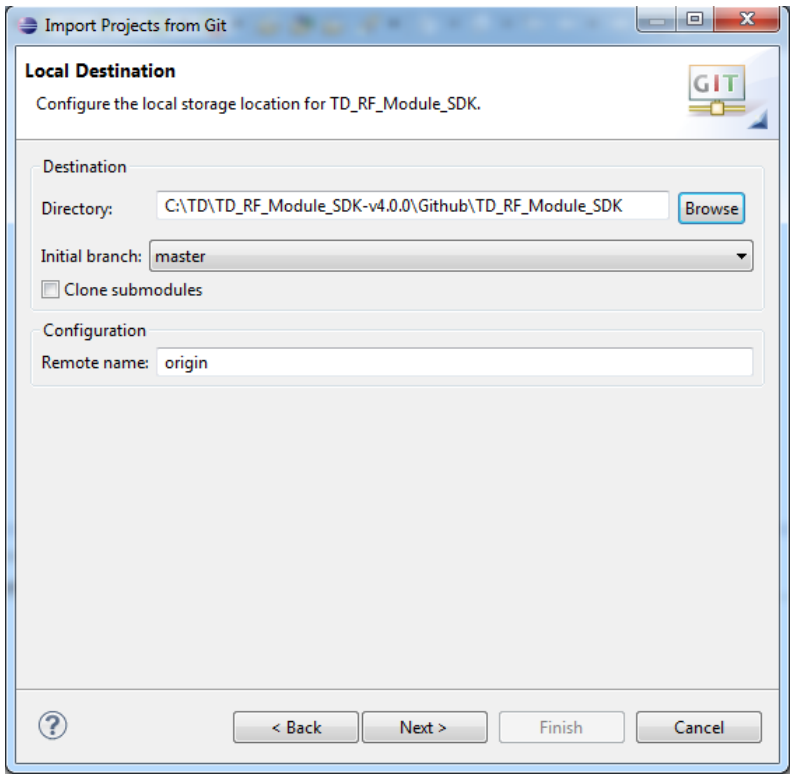


Figura B-2. Indicando el directorio destino de las librerías.

5. En la siguiente ventana, pulsamos **Import existing projects**, y pulsamos **Next**. Se nos mostrarán todas las **librerías a incluir** (seleccionadas todas por defecto). Con esto, pulsamos **Finish**, y habremos concluido la inclusión de las librerías en Eclipse.

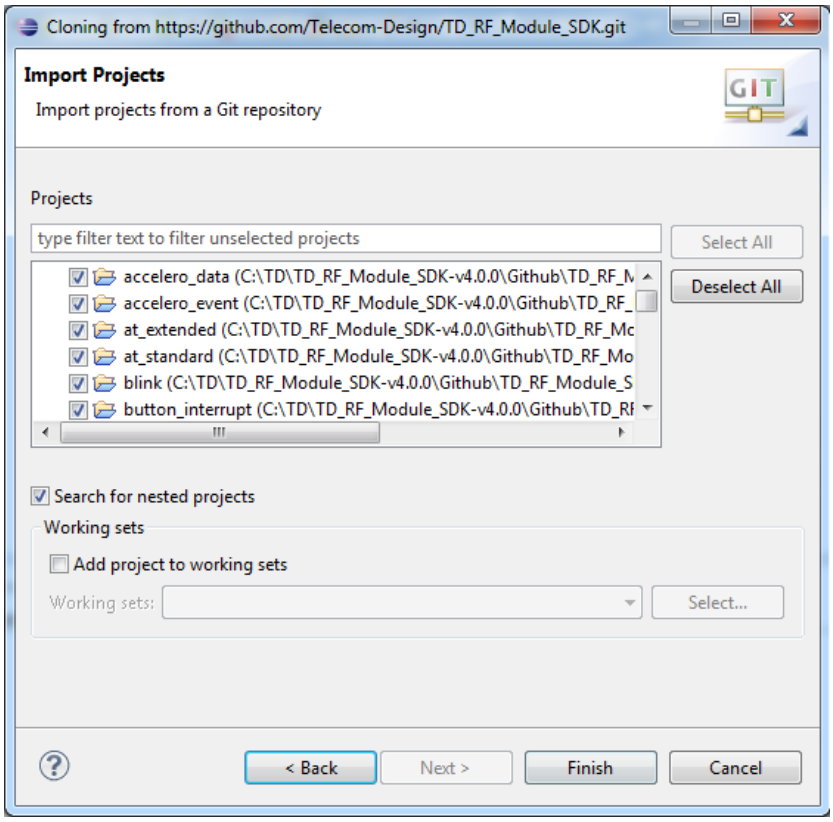


Figura B-3. Librerías a incluir en Eclipse. Se deben seleccionar todas.

B.3. Organizando el código fuente

Tras los pasos anteriores, podremos ver todas las librerías y ejemplos sin ningún tipo de organización lógica en el Project Explorer, ordenados alfabéticamente.

Para tener una **mejor organización** lógica, se debe **importar unos Working Sets** de Eclipse, que organicen todos los **proyectos por categorías**. Esto se consigue siguiendo los siguientes pasos:

1. Vamos al menú **File**, y seleccionamos la opción **Import…**; allí, elegimos **General > Working Sets**, y pulsamos **Next**.
2. En el campo **Browse…**, añadimos la ruta de los Working Sets definidos por Telecom Design: **C:\TD\TD_RF_Module_SDK-v4.0.0\Github\TD_RF_Module_SDK\TD_RF_Module_SDK.wst**. Seleccionamos todas las opciones y pulsamos **Finish**.

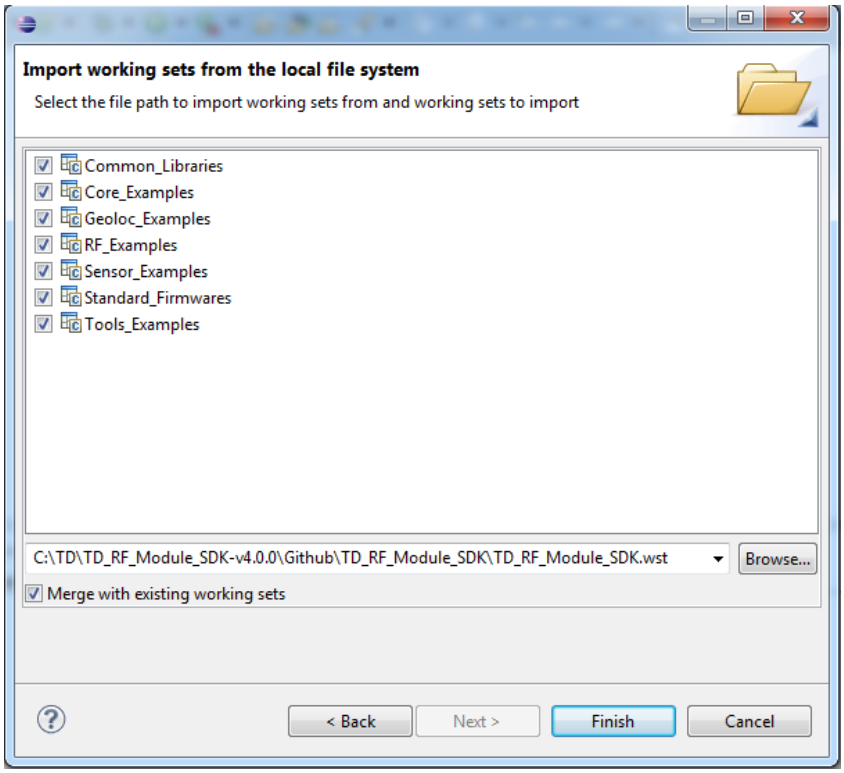


Figura B-4. Añadiendo los Working Sets al proyecto.

3. Ahora, los Working Sets está importado, pero no seleccionados. Para ello, seleccionamos la **flecha** que está al lado del botón de minimizar y maximizar en el Project Explorer, y seleccionamos **Select Working Set…**

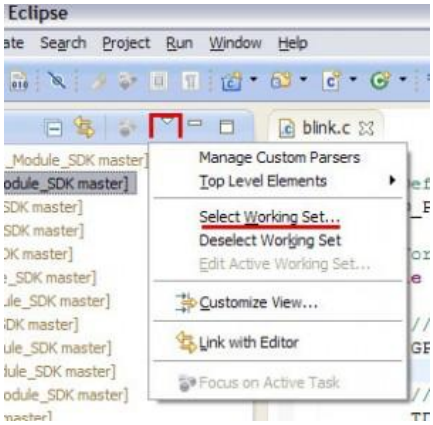


Figura B-5. Acceso a la selección del Working Set.

4. Allí, seleccionaremos **todas las opciones** presentes, con **Select Working Sets** y **Select All**, y finalmente pulsaremos en **OK**.

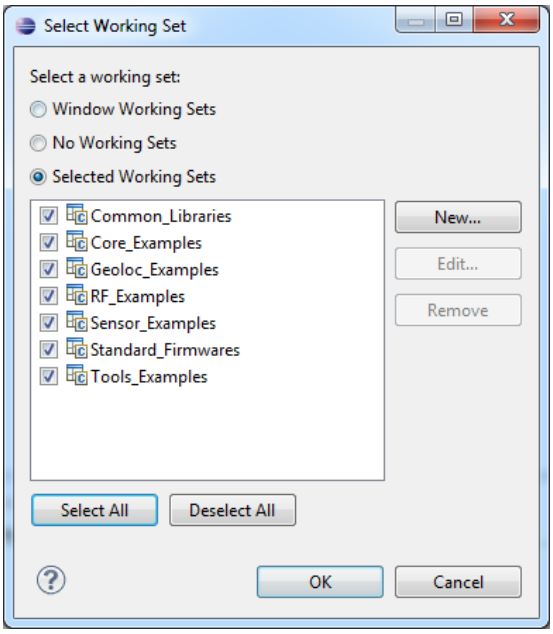


Figura B-6. Seleccionando los Working Sets.

5. Por último, activamos la visualización de los Working Sets, a través de la misma **flecha** seleccionada en el paso 3, pero pulsando en **Top Level Elements > Working Sets**en esta ocasión.

B.4. Compilando las librerías con el compilador adecuado

Ya tenemos el código fuente, pero las librerías base para los ejemplos y nuestro código no están compiladas aún. A continuación, deberemos elegir el compilador adecuado para estas librerías y compilar todo el código fuente, realizando lo siguiente:

1. Seleccionamos **todos los proyectos** del directorio **Common_Libraries**, para proceder a su compilación conjunta.

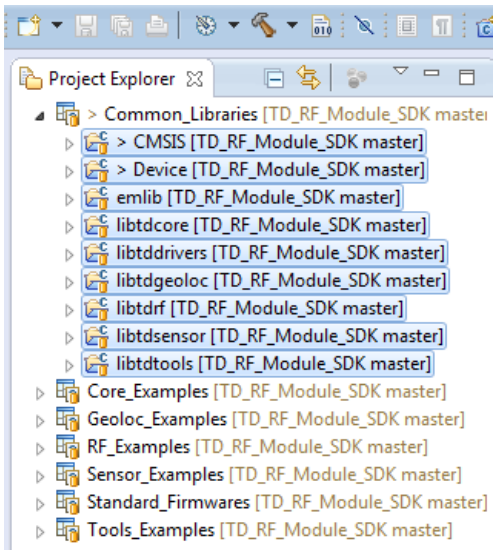


Figura B-7. Seleccionando las librerías para su compilación.

2. Ahora, seleccionaremos el **modo de compilación**. Para ello, pulsamos en **icono de la anilla** que aparece en la parte superior del Project Explorer, y seleccionaremos la opción **GCC Release**. Tras

ello, pulsaremos en el **icono del martillo** que está a su derecha (builder) y también elegiremos la opción **GCC Release**, que compilará todas las librerías.

Figura B-8. Los iconos a seleccionar en el paso 2.

3. Si el proceso se ha realizado correctamente, veremos un **mensaje** afirmativo en la **consola** de Eclipse. Esto se debe tener en cuenta no sólo con estas librerías, sino con cualquier código fuente que se compile.

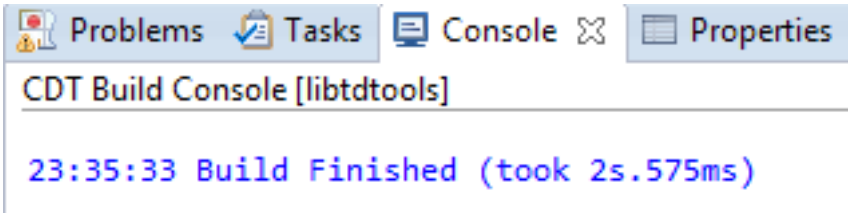


Figura B-9. Mensaje de compilación correcta.

B.5. Compilando el proyecto deseado

Una vez compiladas las librerías, se pasaría a **compilar el proyecto** que contenga la aplicación a probar en la placa de evaluación.

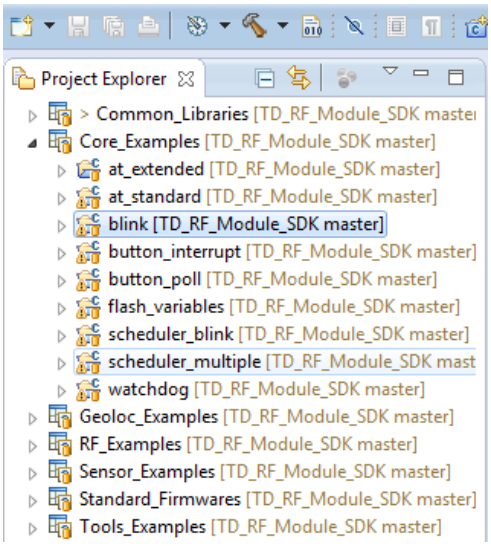


Figura B-10. Seleccionando el proyecto deseado para su compilación (blink, en este caso).

El proceso es el mismo que el seguido en el paso B.3., pero con una diferencia: la **opción** a elegir en el icono de la anilla y del martillo debe ser **TD1204 Release**, siguiendo el mismo orden de selección nuevamente.

Si la compilación se realizó correctamente, se generará un directorio llamado TD1204 Release en el proyecto compilado, que contendrá el fichero **.bin** que deberá pasarse a la placa de evaluación a través de la conexión por el puerto serie del ordenador.

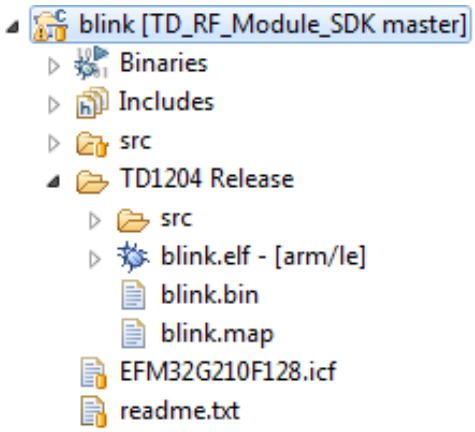


Figura B-11.Resultado de la compilación del proyecto, con el fichero .bin generado.

B.6. Ejecutando la aplicación en la placa de evaluación

Tras generar el .bin del proyecto, podremos pasárselo a la placa de evaluación para que lo ejecute. El proceso es bastante sencillo, y se realiza con la aplicación **TD Loader** (la versión usada ha sido la 1.06), que ya fue introducida en páginas anteriores de la memoria.

Una vez abierta, tendremos que seleccionar estas tres opciones correctamente, realizando el mismo proceso que el seguido para actualizar el firmware de la placa, aunque se repite por comodidad:

- El **puerto serie** por el que la placa de evaluación se conecta al ordenador. El número concreto puede verse en **Panel de control > Hardware y sonido > Administración de dispositivos > Puertos (COM y LPT)> USB Serial Port (COMx)**. Ese valor **x** deberá ponerse en este campo.
- En la opción de **producto**, elegimos la opción**TD1204 EVB**.
- En el nombre del fichero, ponemos la ruta donde se encuentre el fichero **.bin** a usar.

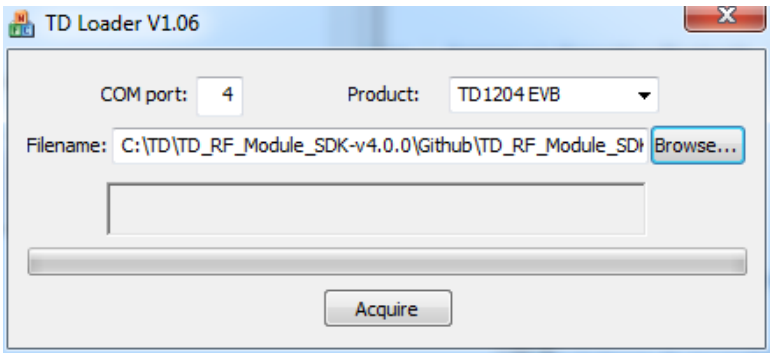


Figura B-12.Parámetros de configuración de TD Loader.

Por último; pulsando en **Acquire**, se lanzaría la aplicación a la placa, y se ejecutaría en la misma. Recordar que se puede hacer uso de aplicaciones como PuTTY para conectarnos a la placa por el puerto serie; de manera que, si la placa imprimiese información por dicho puerto, se podría ver en esa conexión.

Antes de concluir este apartado, se debe advertir al usuario que añadir una nueva aplicación a la placa no será tan fácil como la primera vez. En términos generales, la aplicación subida impide que la placa vuelva a leer el puerto serie para obtener una nueva aplicación salvo que se produzca un **reset forzado durante la fase de adquisición** (esto es, la fase de espera que resulta tras pulsar Acquire).

Dicho reset se puede realizar conectando el pin RST de la cabecera ISP a tierra, ya que es activo a nivel bajo (consultar el datasheet para más información), y en el estado normal de funcionamiento ese pin se

encuentra al aire. Basta con una conexión breve entre esos dos pines con un cable, hasta que se encienda el LED azul de la placa, que indica que se ha producido un reset.

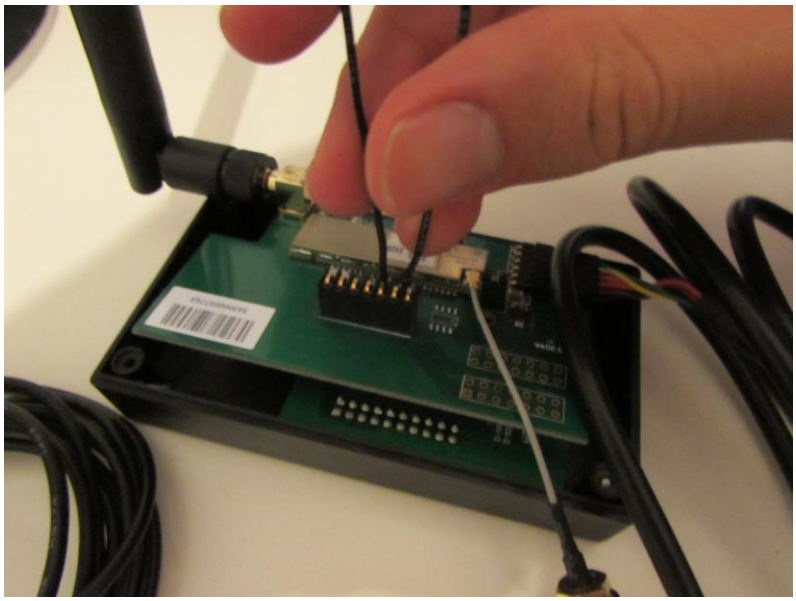


Figura B-13.Forzando un reset en la placa para subir una nueva aplicación.

B.7. Creando nuestro proyecto de aplicación

Todos los pasos comentados anteriormente nos permiten, como resultado, generar las aplicaciones de los proyectos de ejemplo del repositorio para utilizarlos en la placa. Además, se ha podido ver la forma de cambiar de una aplicación a otra, utilizando el reset forzado.

El paso restante para completar la configuración de Eclipse sería la **creación de un proyecto** para poder **programar nuestra propia aplicación**. Se deben seguir los siguientes pasos:

1. En el **Project Explorer**, abrimos **Standard_Firmwares** y nos fijamos en el proyecto llamado **td12xx_template**. Si estudiamos su contenido, descubriremos que este proyecto sirve como **esqueleto** para generar un nuevo proyecto de aplicación.

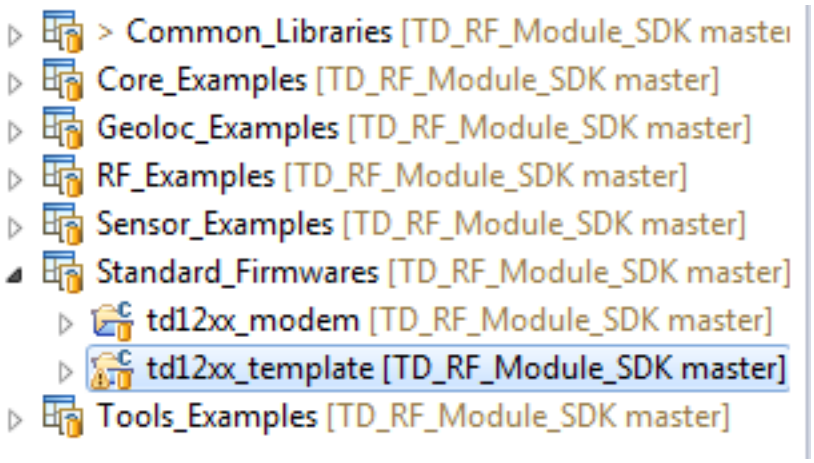


Figura B-14. Ubicación del proyecto td12xx_template en el Project Explorer.

2. Seleccionamos ese proyecto, tal y como se hace en la figura anterior, y lo **copiamos**. Después, lo **pegamos** en el Project Explorer. Entre las opciones de pegado, podremos seleccionar el nuevo

nombre del proyecto y su ubicación (por defecto, en el directorio workspace del SDK).

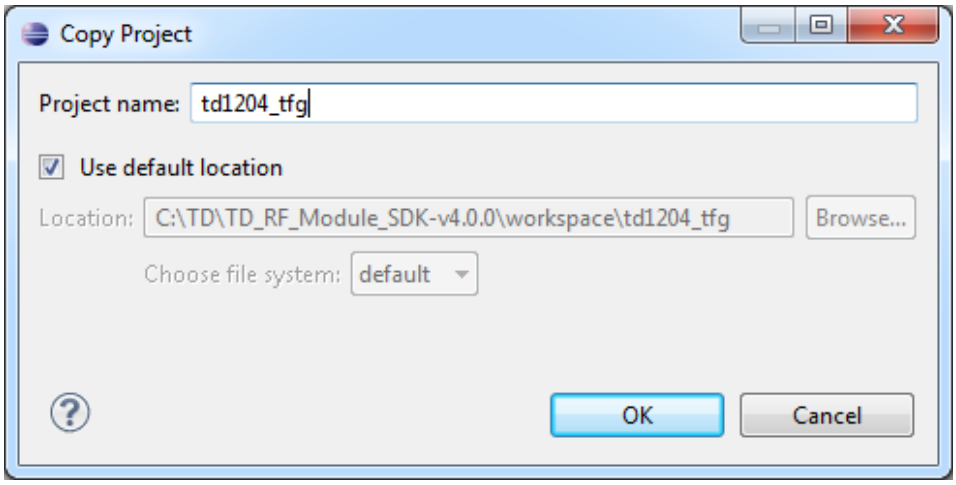


Figura B-15. Opciones de generación del nuevo proyecto.

3. Estos pasos generan el proyecto, pero todavía no aparecería en el Project Explorer. Para poder visualizarlo, tenemos que acceder a la opción **Select Working Set…**(como se mostraba en la figura B-5). Una vez allí, seleccionaremos el **directorio** en el que queremos que aparezca el proyecto (en nuestro caso, Standard_Firmwares), y pulsaremos en **Edit…**para añadirlo. Una vez marcado, pulsar en **Finish** en esa ventana y en **OK** en Select Working Set.

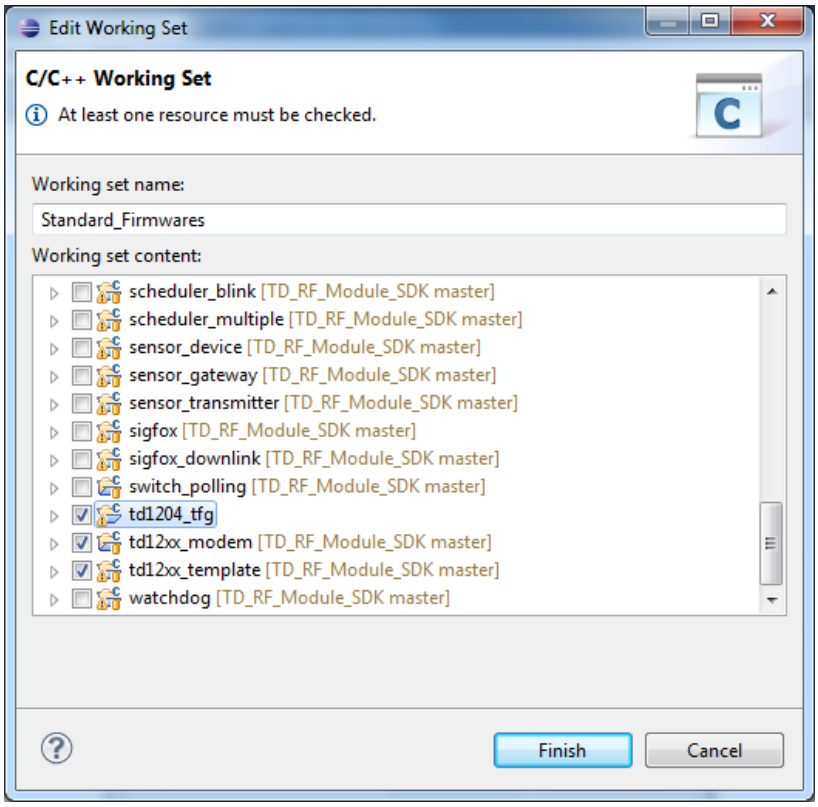


Figura B-16. Seleccionando el nuevo proyecto para su visualización.

Con esto, ya tendremos **visible** el proyecto en el directorio elegido. Faltaría por añadir las **librerías** necesarias para que la compilación de la aplicación desarrollada se realice correctamente. Para ello, se debe hacer lo siguiente:

1. Seleccionamos el nuevo proyecto en el Project Explorer, y haciendo click derecho sobre él,

accedemos a la opción **New > Folder**, para crear un directorio con nombre **inc** para incluir los ficheros de cabecera que necesite la aplicación.

2. Ahora, nos dirigimos a la barra de herramientas, y con el proyecto nuevo todavía seleccionado, elegimos la opción **Project > Properties**. Allí, nos dirigiremos a la opción **C/C++ General > Paths and Symbols**, y en la pestaña **Library Paths**, tendremos que incluir con la opción **Add…** las tres últimas **rutas** que aparecen en la siguiente figura.

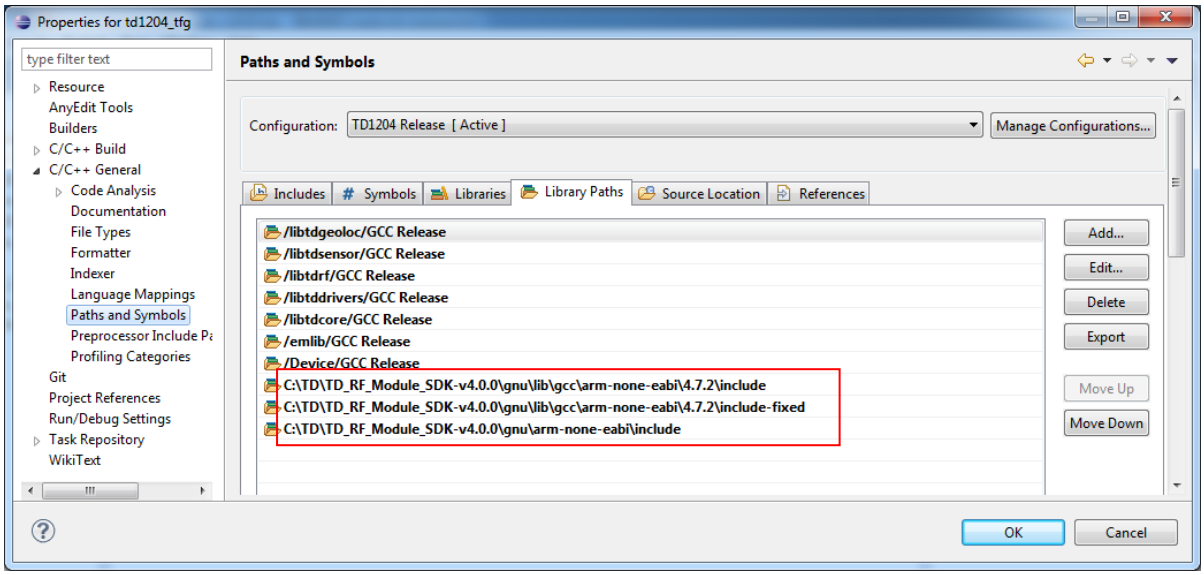


Figura B-17. Rutas a incluir en la pestaña Library Paths.

3. Por último, nos vamos a **C/C++ General > Indexer** en las propiedades, y elegimos la opción **TD1204 Release** en el campo **Build configuration for the indexer**. Con esto, pulsaríamos en **Apply** y en **OK** para aceptar los cambios realizados.

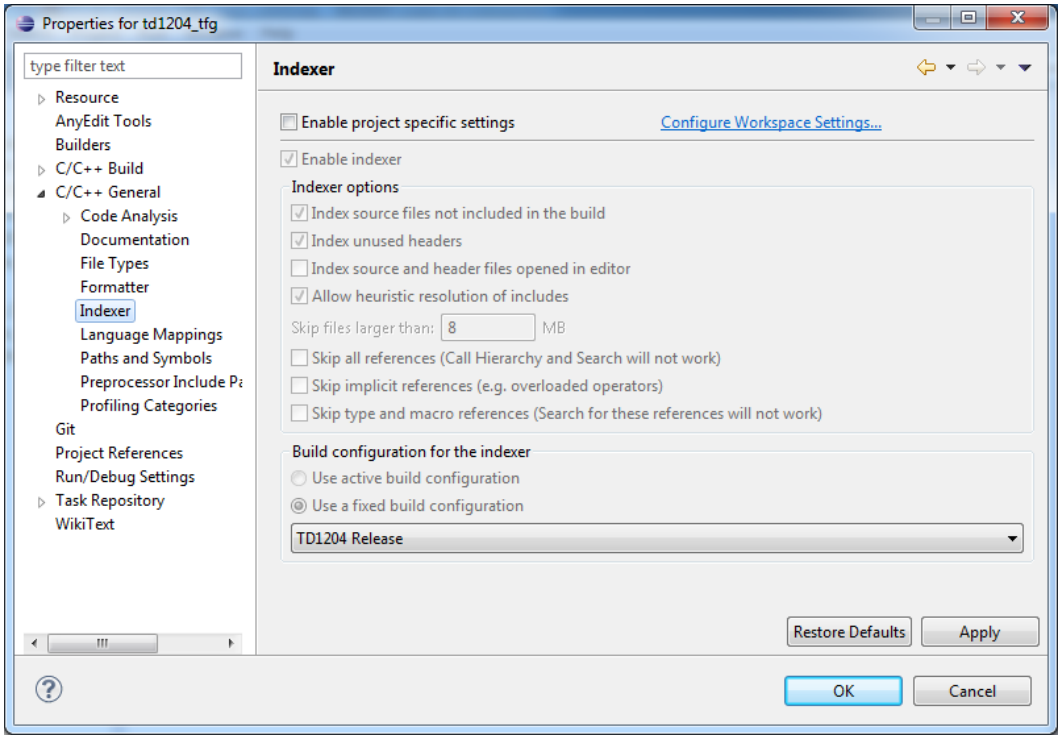


Figura B-18. Eligiendo el modo de compilación adecuado para el nuevo proyecto.

Tras todos estos pasos, nuestro proyecto quedaría perfectamente **configurado** para la generación de ejecutables para la placa de evaluación. Tras esto, faltaría la **codificación** de la aplicación en sus ficheros correspondientes, y repetir los pasos B.4. y B.5. para **compilar**la aplicación y **ejecutarla**.

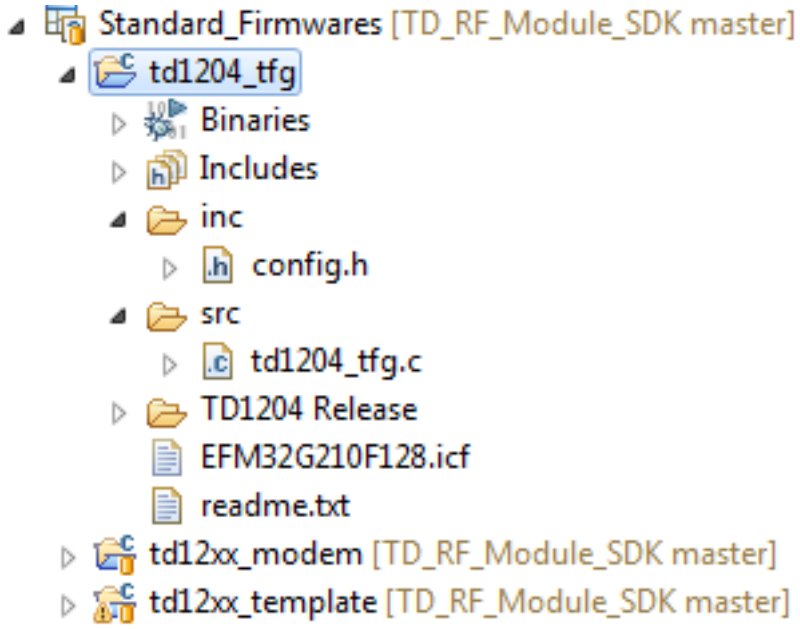


Figura B-19. Resultado final de la creación del nuevo proyecto.

ANEXO C. DOCUMENTACIÓN DEL CÓDIGO (DOXYGEN)

Se adjunta el **código** realizado para la programación del módem TD1204, en base a los casos de uso definidos y la solución final propuesta, junto a la **documentación** generada por **Doxygen**.

Notar que sólo se añadirá la documentación relativa al código programado por el alumno, sin entrar en el contenido de las librerías propias de Telecom Design.

También se habilita el siguiente enlace para la descarga de la documentación y visualización en local: https://www.dropbox.com/s/mb2bmpbkwlew82l/html.zip?dl=0. En esta documentación, sólo se han adjuntado las librerías utilizadas en la aplicación. El resto de ficheros, por comodidad, se han omitido.

C.1. Welcome to TD1204 TFG Doxygen documentation

C.1.1. Overview

Project which contains the code to manage the TD1204 EVB

Author:
Ramón Pérez Hernández

Version:
1.0.0 (24 June 2015)

C.1.2. Description

This example uses a main() function that is managed by the libtdcore library, providing the user with a **TD_USER_Setup()** function called once at startup and a **TD_USER_Loop()** function called every time the TD1204 RF modules wakes up from sleep mode.

You can find all the details into the source code, but we would like to talk about general issues here, such as:

-The main purpose of this project is to use the TD1204 EVB as a sensor, using the SigFox network to send information with less battery consumption, bandwidth or final price, among others.

-Our sensor prototype will consider some use cases, such as:

Temperature: monitor temperature value and detect low/high/ok temperature states.

Battery: same case as temperature, but considering low/ok battery states in this case.

RTC: with a fixed interval, it will send temperature and battery values. It is done because we always have these values stored and accessible. In temperature and battery monitoring, it could happen EVB always were in OK states, so it would never send a message and we would not have any information of that.

Movement: using the accelerometer integrated into the EVB, it is able to detect a movement with high precision. In our implementation, it enables GPS when a movement is detected after detecting the first well-known position. It will notify with a message if it reaches a maximum of movements, restarting the count of movements after all. Originally, we considered free fall event too, but it has been impossible to detect because accelerometer only uses one working mode, with a callback associated to that. We finally thought it would be more interesting to monitor movement instead of free fall, because it's more likely to happen, mainly.

Position: using the external GPS which is connected to the EVB, it can find its current position in case of movement event, and compare it with the first well-known position in order to know whether it has changed its position or not.

C.1.3. Packages

This Doxygen documentation includes all the libraries which have been included into **td1204_tfg.c** file (lib folder). We have to say there are more and more libraries which have not been used, but there are irrelevant to the final result, so there are not included.

C.2. inc/config.h File Reference

Configuration file for the TD1204 RF modules.

```
#include <td_module.h>
```

C.2.1. Detailed Description

Configuration file for the TD1204 RF modules.

Author:
Telecom Design S.A. (edited by Ramón Pérez Hernández)

Version:
1.0.0

Date:
16 June 2015

C.2.2. Control

1.0.0 - Initial and final commit. Nothing to do here. We will use the same **config.h** file that all example projects have.

C.2.3. License

C.3. src/td1204_tfg.c File Reference

File which contains the code to manage the TD1204 EVB.

```
#include "config.h"
#include <efm32.h>
#include <stdbool.h>
#include <stdint.h>
#include <td_core.h>
#include <td_uart.h>
#include <td_printf.h>
#include <td_rtc.h>
#include <td_gpio.h>
#include <td_scheduler.h>
#include <td_measure.h>
#include <td_watchdog.h>
#include <td_sigfox.h>
#include <td_accelero.h>
#include <td_geoloc.h>
#include <td_sensor.h>
#include <td_config.h>
```

C.3.1. Macros

- #define **WDOG_TIME** 64
- #define **LED_PORT** TIM2_PORT
- #define **LED_BIT** TIM2_BIT
- #define **FIX_TIMEOUT** 600
- #define **DIFF_LAT_LNG** 10000
- #define **BATTERY** 0
- #define **B_CHK_INTERVAL** 1800
- #define **LOW_B_LEVEL** 2600
- #define **OK_B_LEVEL** 3000
- #define **TEMPERATURE** 1
- #define **T_CHK_INTERVAL** 1800
- #define **LOW_T_LEVEL** 0
- #define **OK_T_LEVEL_DOWN** 100
- #define **OK_T_LEVEL_UP** 550
- #define **HIGH_T_LEVEL** 650
- #define **LOW_BATTERY** 0
- #define **OK_BATTERY** 1
- #define **LOW_TEMP** 0
- #define **OK_TEMP** 1
- #define **HIGH_TEMP** 2
- #define **MAX_MOV** 5
- #define **RTC_INTERVAL** 3600
- #define **BT_MSG_LENGTH** 6
- #define **WM_MSG_LENGTH** 8
- #define **AG_MSG_LENGTH** 11
- #define **MSG_LENGTH** 12

C.3.2. Functions

- static void **GPSFix** (**TD_GEOLOC_Fix_t** *fix, **bool** timeout)
  Function GPSFix. GPS Fix callback for every geoloc event.

- void **MoveDetected** (uint8_t source)
  Function MoveDetected. Called on move detection.
- static void **GPSInitialFix** (**TD_GEOLOC_Fix_t** *fix, **bool** timeout)
  Function GPSInitialFix. GPS Fix callback for the first well-known location. After that, it can start with accelerometer events.
- static void **ThresholdFunction** (uint32_t arg, uint8_t repetition)
  Function ThresholdFunction. Timer callback for monitoring battery and temperature. It will be called every 30 minutes.
- static void **TimerFunction** (uint32_t arg, uint8_t repetition)
  Function TimerFunction. Timer callback for RTC. It will be called every hour.
- void **TD_USER_Setup** (void)
  Function TD_USER_Setup. User setup function.
- void **TD_USER_Loop** (void)
  Function TD_USER_Loop. User loop function.

C.3.3. Variables

- uint32_t **now** = 0
- uint32_t **sequence_number** = 0
- uint32_t **latitude**
- uint8_t **lat_direction**
- uint32_t **longitude**
- uint8_t **long_direction**
- **bool fixing_enabled** = **true**
- uint8_t **movements** = 0
- uint32_t **temperature**
- uint32_t **battery**
- uint8_t **temp_state** = **OK_TEMP**
- uint8_t **battery_state** = **OK_BATTERY**
- uint8_t **message** [**MSG_LENGTH**]

C.3.4. Detailed Description

File which contains the code to manage the TD1204 EVB.

Author:
Ramón Pérez Hernández

Version:
1.0.0

Date:
24 June 2015

C.3.5. Control

0.0 - Initial commit. Empty project.

0.1 - The code allows us to measure temperature and voltage with a callback function, which is called every 5 seconds (it's changeable).

0.1.1 - Improving project documentation and comments at source code.

0.2 - Information is sent to SigFox backend. We have to make a necessary change in the schedule to make it possible - we don't use Irq right now, because it causes problems when it starts the transmission.

0.3 - Including the monitoring of temperature and battery, using thresholds. It has been included the sleep mode, too (but we have to test it better). Testing the application, we have found several problems: battery

event is never detected (in theory, it is checked in every SigFox transmission), and battery event is only detected once (for example, it detects a low temperature event, and it doesn't have any event until it detects a high temperature event). Moreover, it never detects temperature ok event. It might be improved by using a timer instead of Sensor functions.

0.4 - Changing the treatment of battery and temperature monitoring, using a timer with both of them. Tested, and it works.

0.5 - Including the monitoring of accelerometer and geoloc. We introduce two use cases: position change and movement detection. We don't test them. Moreover, we have found some problems: it can only use TD_ACCELERO_MonitorEvent once, and this EVB doesn't allow multithread, so it's impossible to use it to detect free fall. We consider more important to monitor movement instead of free fall.

0.6 - Correcting some issues, such as temperature threshold (in order to add hysteresis treatment), message formats, and latitude-longitude values, with their correct value (they are represented in deg-min, not in decimal). Testing process was successful.

0.7 - Correcting details, such as turning off GPS when fixing is completed in GPSFix function too, and calculate diff_lat/lon in case it has different directions in each case.

0.8 - Checking all the comments, in order to use them in Doxygen documentation. Correcting GPS condition to send a new position.

0.9 - Correcting problems with GPS. Checking was wrong. With all these changes, testing results were finally correct.

0.9.1 - Checking all the comments again to correct some details, such as syntax issues.

1.0.0 - Changing timer intervals to their original values. Source code prepared for final release.

C.3.6. License

C.3.7. Macro Definition Documentation

#define AG_MSG_LENGTH 11

Accelero/geoloc-event message length, in bytes.

#define B_CHK_INTERVAL 1800

Battery checking interval, in seconds (30 min).

#define BATTERY 0

Battery event (for ThresholdFunction).

#define BT_MSG_LENGTH 6

Battery/temperature-event message length, in bytes.

#define DIFF_LAT_LNG 10000

Maximum difference between first and new latitude/longitude (its value, 10000, is equal to 0.1').

#define FIX_TIMEOUT 600

Stop trying to fix after timeout, in seconds (10 min).

#define HIGH_T_LEVEL 650

High temperature level, in 10*degrees.

#define HIGH_TEMP 2

High temperature state.

#define LED_BIT TIM2_BIT

LED bit.

#define LED_PORT TIM2_PORT

LED port.

#define LOW_B_LEVEL 2600

Low battery level, in mV.

#define LOW_BATTERY 0

Low battery state.

#define LOW_T_LEVEL 0

Low temperature level, in 10*degrees.

#define LOW_TEMP 0

Low temperature state.

#define MAX_MOV 5

Maximum number of movements.

#define MSG_LENGTH 12

Message length (max), in bytes.

#define OK_B_LEVEL 3000

OK battery level, in mV.

#define OK_BATTERY 1

Ok battery state.

#define OK_T_LEVEL_DOWN 100

Beginning of ok temperature level, in 10*degrees.

#define OK_T_LEVEL_UP 550

End of ok temperature level, in 10*degrees.

#define OK_TEMP 1

Ok temperature state.

#define RTC_INTERVAL 3600

Interval for RTC Scheduler (1 hour).

#define T_CHK_INTERVAL 1800

Temperature checking interval, in seconds (30 min).

#define TEMPERATURE 1

Temperature event (for ThresholdFunction).

#define WDOG_TIME 64

Watchdog time, in seconds.

#define WM_MSG_LENGTH 8

Welcome/movement-event message length, in bytes.

C.3.8. Function Documentation

C.3.8.1. static void GPSFix (TD_GEOLOC_Fix_t * *fix*, bool *timeout*) [static]

Function GPSFix. GPS Fix callback for every geoloc event.

It is called when a movement is detected (accelerometer event), in order to obtain the current position and compare it with the first well-known position.

It will do anything if the fixing has enough quality (TD_GEOLOC_3D_FIX or better) or if it reaches the timeout. In both cases, it stops GPS and set fixing_enabled to true to enable GPS fixes again with another position.

-Timeout: nothing to do.

-Fixing: calculate latitude and longitude difference between the new and first position, and check if any difference is higher than 0.1 min. If it happens, it will send a message to SigFox backend with the new latitude and longitude, and their directions.

Parameters:

in	*fix*	The GPS fix data structure.

in	*timeout*	Flag that indicates whether a timeout occurred if set to true.

```
278 {
279     /* It needs a fixing type with give the latitude and longitude of the
280      * EVB with enough precision (like TD GEOLOC 3D FIX or better), or a
281      * timeout event. */
282     if (fix->type >= TD_GEOLOC_3D_FIX || timeout) {
283         tfp_printf("Stop fixing.\r\n");
284 
285         /* Stop GPS, because it could spend a long time until it has a geoloc
286          * event due to an accelerometer event. Thus, it could reduce battery use. */
287         TD GEOLOC StopFix(TD GEOLOC OFF);
288 
289         if (timeout) {
290 
291             // Timeout case.
292             tfp printf("Timeout reached.\r\n");
293 
294         }
295         else {
296 
297             /* Fixing case.
298              * It will do intermediate operations, to calculate and adjust the
299              * difference between the first and new position to positive values
300              * (firstly, with fix values, and after that, with the difference). */
301 
302             int32_t diff_lat = fix->position.latitude;
303 
304             if (diff lat < 0) {
305 
306                 // Negative latitude (S).
307                 diff_lat = -diff_lat;
308 
309                 if (lat direction == 'S') {
310 
311                     // Same direction: subtraction.
312                     diff_lat -= (int32_t) latitude;
313 
314                     /* As it does a subtraction, it has to check if the result is
315                      * negative. */
316                     if (diff_lat < 0)
317                         diff_lat = -diff_lat;
318                 }
319                 else {
320 
321                     // Opposite direction: sum.
322                     diff_lat += (int32_t) latitude;
323                 }
324             }
325             else {
326 
327                 // Positive latitude (N). Don't change the sign.
328 
329                 if (lat_direction == 'N') {
330 
331                     // Same direction: subtraction.
332                     diff lat -= (int32 t) latitude;
333 
334                     /* As it does a subtraction, it has to check if the result is
335                      * negative. */
336                     if (diff_lat < 0)
337                         diff lat = -diff lat;
338 
339                 }
340                 else {
341 
342                     // Opposite direction: sum.
343                     diff lat += (int32 t) latitude;
344                 }
345             }
346 
347             int32_t diff_lon = fix->position.longitude;
348 
349             if (diff lon < 0) {
350 
351                 // Negative longitude (W).
352                 diff_lon = -diff_lon;
```

```
  353
  354                 if (long_direction == 'W') {
  355
  356                     // Same direction: subtraction.
  357                     diff_lon -= (int32_t) longitude;
  358
  359                     /* As it does a subtraction, it has to check if the result is
  360                      * negative. */
  361                     if (diff lon < 0)
  362                         diff lon = -diff lon;
  363
  364                 }
  365                 else {
  366
  367                     // Opposite direction: sum.
  368                     diff_lon += (int32_t) longitude;
  369                 }
  370             }
  371             else {
  372
  373                 // Positive longitude (E). Don't change the sign.
  374
  375                 if (long_direction == 'E') {
  376
  377                     // Same direction: subtraction.
  378                     diff lon -= (int32 t) longitude;
  379
  380                     /* As it does a subtraction, it has to check if the result is
  381                      * negative. */
  382                     if (diff_lon < 0)
  383                         diff_lon = -diff_lon;
  384                 }
  385                 else {
  386
  387                     // Opposite direction: sum.
  388                     diff_lon += (int32_t) longitude;
  389                 }
  390             }
  391
  392             /* In order to know if it has to send the captured position to the backend,
  393              * it has to check if the difference between first and new position,
  394              * in latitude or longitude, is higher than 0.1 min (it's, typically,
  395              * the distance between two streets). */
  396             if ((diff lat > DIFF LAT LNG) || (diff lon > DIFF LAT LNG)) {
  397
  398                 // Obtain latitude, longitude and their directions (with positive
values).
  399                 uint32 t lat;
  400                 uint8 t lat dir;
  401
  402                 if (fix->position.latitude < 0) {
  403                     lat_dir = 'S';
  404                     lat = -fix->position.latitude;
  405                 }
  406                 else {
  407                     lat dir = 'N';
  408                     lat = fix->position.latitude;
  409                 }
  410
  411                 uint32 t lon;
  412                 uint8 t lon dir;
  413                 if (fix->position.longitude < 0) {
  414                     lon_dir = 'W';
  415                     lon = -fix->position.longitude;
  416                 }
  417                 else {
  418                     lon_dir = 'E';
  419                     lon = fix->position.longitude;
  420                 }
  421
  422                 tfp printf("Position change detected: %2d deg %02d.%05d min %c %3d deg
%02d.%05d min %c\r\n",
  423                             lat/10000000, (lat%10000000)/100000, lat%100000, lat dir,
  424                             lon/10000000, (lon%10000000)/100000, lon%100000, lon_dir);
  425
  426                 /* Print information about the current moment. Remember time is
relative (when
```

```
427                  * the program starts, the time is set to zero, and it increments every
428                  * second). */
429                 now = TD_SCHEDULER_GetTime() >> 15;
430                 tfp_printf("GPSFix. Time %d\r\n", now);
431
432                 /* After that, send a frame to the backend, with format:
433                  * -1B with characters 'A' (accelerometer) (1B)
434                  * -Latitude (4B) and direction N/S (1B)
435                  * -Longitude (4B) and direction W/E (1B)
436                  *  = 11B
437                  *  (It doesn't matter sequence_number here, because it helps us to
438                  *  order the messages in the server and show the correct graphic
439                  *  with statistics).
440                  * Put the variables in the correct order to make the message. */
441                 message[0] = (uint8_t) 'A';
442                 message[1] = lat >> 24;
443                 message[2] = lat >> 16;
444                 message[3] = lat >> 8;
445                 message[4] = lat;
446                 message[5] = lat_dir;
447                 message[6] = lon >> 24;
448                 message[7] = lon >> 16;
449                 message[8] = lon >> 8;
450                 message[9] = lon;
451                 message[10] = lon_dir;
452
453                 // And then, it sends the message.
454                 tfp printf("Sending message to backend with location data.\r\n");
455                 GPIO PinOutSet(LED PORT, LED BIT);
456                 TD SIGFOX Send(message, AG MSG LENGTH, 1);
457                 GPIO_PinOutClear(LED_PORT, LED_BIT);
458             }
459         }
460
461         // Finally, restart fixing right away.
462         tfp printf("Fixing on move enabled.\r\n\r\n");
463         fixing_enabled = true;
464     }
465 }
```

C.3.8.2. static void GPSInitialFix (TD_GEOLOC_Fix_t * *fix*, bool *timeout*) [static]

Function GPSInitialFix. GPS Fix callback for the first well-known location. After that, it can start with accelerometer events.

It won't do anything until it has a position with enough quality (TD_GEOLOC_3D_FIX).

When it happens, it will store that position (latitude, longitude and their directions), stop GPS, and send it to SigFox backend.

After all, it will start to monitor accelerometer events. It means that accelerometer will be disabled until that position was found.

Parameters:

in	*fix*	The GPS fix data structure.
in	*timeout*	Flag that indicates whether a timeout occurred if set to true (never reached).

```
547 {
548     /* It needs a fixing type with give us the latitude and longitude of the
549      * EVB with enough precision (like TD_GEOLOC_3D_FIX or better). */
550     if (fix->type >= TD_GEOLOC_3D_FIX) {
551
552         // Latitude.
553         if (fix->position.latitude < 0) {
554             lat direction = 'S';
555             latitude = -fix->position.latitude;
556         }
557         else {
558             lat_direction = 'N';
559             latitude = fix->position.latitude;
560         }
```

```
  561
  562         // Longitude.
  563         if (fix->position.longitude < 0) {
  564             long direction = 'W';
  565             longitude = -fix->position.longitude;
  566         }
  567         else {
  568             long_direction = 'E';
  569             longitude = fix->position.longitude;
  570         }
  571
  572         tfp_printf("First location: %2d deg %02d.%05d min %c %3d deg %02d.%05d min
%c\r\n",
  573             latitude/10000000, (latitude%10000000)/100000, latitude%100000,
lat_direction,
  574             longitude/10000000, (longitude%10000000)/100000, longitude%100000,
long_direction);
  575
  576         /* Now, it stops fixing geolocation, because it could spend a long
  577          * time until it has a geoloc event, due to an accelerometer event.
  578          * So that, it could reduce battery use. */
  579         TD_GEOLOC_StopFix(TD_GEOLOC_OFF);
  580
  581         /* After that, send a frame to the backend, with format:
  582          * -1B with characters 'G' (geoloc) (1B)
  583          * -Latitude (4B) and direction N/S (1B)
  584          * -Longitude (4B) and direction W/E (1B)
  585          *  = 11B
  586          *  (It doesn't matter sequence_number here, because it helps us to
  587          *  order the messages in the server and show the correct graphic
  588          *  with statistics). */
  589
  590         /* Print information about the repetition. Remember time is relative (when
  591          * the program starts, the time is set to zero, and it increments every
  592          * second). */
  593         now = TD_SCHEDULER_GetTime() >> 15;
  594         tfp_printf("GPSInitialFix. Time %d\r\n", now);
  595
  596         // Put the variables in the correct order to make the message.
  597         message[0] = (uint8_t) 'G';
  598         message[1] = latitude >> 24;
  599         message[2] = latitude >> 16;
  600         message[3] = latitude >> 8;
  601         message[4] = latitude;
  602         message[5] = lat_direction;
  603         message[6] = longitude >> 24;
  604         message[7] = longitude >> 16;
  605         message[8] = longitude >> 8;
  606         message[9] = longitude;
  607         message[10] = long_direction;
  608
  609         // And then, it sends the message.
  610         tfp_printf("Sending message to backend with first location data.\r\n\r\n");
  611         GPIO_PinOutSet(LED_PORT, LED_BIT);
  612         TD_SIGFOX_Send(message, AG_MSG_LENGTH, 1);
  613         GPIO_PinOutClear(LED_PORT, LED_BIT);
  614
  615         // Finally, it starts to monitor accelerometer events (movements).
  616         TD_ACCELERO_MonitorEvent(true,  // Monitoring enabled.
  617             TD_ACCELERO_10HZ,           // Sampling rate: 10 Hz.
  618             TD_ACCELERO_ALL_AXIS,       // Axis mask: all axis.
  619             TD_ACCELERO_2G,             // Scale: 2 g.
  620             TD_ACCELERO_ALL_HIGH_IRQ,   // Only monitor high IRQs, as low IRQs are
always set in
  621                                         // accelerometer registers.
  622             10,                         // Threshold in mg = 10 * 2 g / 127 = +- 160 mg
(with scale 2 g).
  623             3,                          // Duration in ms = 3 * (1 / 10 Hz) = 300 ms
(with rate 10 Hz).
  624             1,                          // High-pass filter enabled.
  625             MoveDetected);
  626     }
  627 }
```

C.3.8.3. void MoveDetected (uint8_t *source*)

Function MoveDetected. Called on move detection.

This function is called when a movement is detected by an accelerometer event. When this happens, it increments the variable movements. When it reaches its limit, it will send a message to SigFox backend (movement event), restoring its value to zero.

Moreover, it starts fixing with GPS if there aren't a fixing at the moment (because it could happen that there were a movement before and, after that, fixing was enabled as a result. Checking fixing_enable variable, it would be able to avoid a fixing in that case).

Parameters:

in	*source*	The event source mask that triggered the event.

```
488 {
489     tfp_printf("Movement detected.\r\n");
490     movements++;
491 
492     /* If it reaches MAX MOV value, send a message to the backend, and restore
493      * the value of movements. It will print the current moment of execution, too. */
494     if (movements == MAX_MOV) {
495 
496         now = TD_SCHEDULER_GetTime() >> 15;
497         tfp_printf("MoveDetected. Time %d\r\n", now);
498         tfp printf("Reached limit of movements. Sending a message right now.\r\n\r\n");
499 
500         GPIO_PinOutSet(LED_PORT, LED_BIT);
501         TD_SIGFOX_Send((uint8_t *) "Movement", WM_MSG_LENGTH, 1);
502         GPIO_PinOutClear(LED_PORT, LED_BIT);
503 
504         movements = 0;
505     }
506 
507     // If GPS is not active (so it can start a fixing process).
508     if (fixing_enabled) {
509 
510         // Disable restarting the GPS until a position is found.
511         fixing_enabled = false;
512 
513         // Obtain the current time and print it.
514         now = TD_SCHEDULER_GetTime() >> 15;
515         tfp printf("MoveDetected. Time %d\r\n", now);
516 
517         // And start processing geoloc events with GPSFix callback function.
518         tfp_printf("Starting fixing.\r\n\r\n");
519         TD_GEOLOC_TryToFix(TD_GEOLOC_POWER_SAVE_MODE, FIX_TIMEOUT, GPSFix);
520     }
521 }
```

C.3.8.4. void TD_USER_Loop (void)

Function TD_USER_Loop. User loop function.

It works like a while(true) { do something }. It is used for processing accelerometer and geoloc events. If it didn't do that, it wouldn't process those events.

```
924 {
925     // Process geoloc events.
926     TD GEOLOC Process();
927 
928     // Process accelerometer events.
929     TD_ACCELERO_Process();
930 }
```

C.3.8.5. void TD_USER_Setup (void)

Function TD_USER_Setup. User setup function.

This function is called when the EVB starts its execution. Mainly, it initializes LEUART for printf, watchdog, LED pin, Sensor mode (for GPS), geoloc and accelerometer.

After that, it looks for the first well-known position to store and use it to compare with others positions captured thanks to accelerometer events.

It uses the scheduler to invoke RTC, temperature and battery events, each one with an interval.

And finally, it sends the first message to SigFox backend (a welcome message). Remember it will take 6-7 seconds to send every message.

```
  859 {
  860     // Initialize the LEUART console to use the function tfp printf.
  861     init printf(TD UART Init(9600, true, false),
  862         TD_UART_Putc,
  863         TD_UART_Start,
  864         TD_UART_Stop);
  865
  866     // Use a 64 s automatic watchdog.
  867     TD WATCHDOG Init(WDOG TIME);
  868     TD_WATCHDOG_Enable(true, true);
  869
  870     /* Define the LED pin as an output in push-pull mode. It will use the LED
  871      * in every transmission to the SigFox backend. */
  872     GPIO PinModeSet(LED PORT, LED BIT, gpioModePushPull, 0);
  873
  874     /* Set the device as a Sensor transmitter. Even it is not using TD_SENSOR
  875      * stuff, this call is needed. Otherwise, GPS will not work. */
  876     TD_SENSOR_Init(SENSOR_TRANSMITTER, 0, 0);
  877
  878     // Geoloc and accelerometer initialization.
  879     TD_GEOLOC_Init();
  880     TD_ACCELERO_Init();
  881
  882     /* Obtain the initial position of the EVB (without timeout). After that, it will
  883      * start movement detection with the accelerometer (called in GPSInitialFix
function). */
  884     tfp_printf("Starting fixing.\r\n");
  885     TD_GEOLOC_TryToFix(TD_GEOLOC_POWER_SAVE_MODE, TD_GEOLOC_INFINITE, GPSInitialFix);
  886
  887     /* Enable battery monitoring with a timer, with B CHK INTERVAL s interval and
  888      * the levels low-ok already defined, and setup a callback on event. */
  889     TD_SCHEDULER_Append(B_CHK_INTERVAL, 0 , 0, TD_SCHEDULER_INFINITE,
ThresholdFunction,
  890             BATTERY);
  891
  892     /* Enable temperature monitoring with a timer, with T CHK INTERVAL s interval and
  893      * the levels low-high already defined, and setup a callback on event.
  894      * Notice that it uses the same callback function to monitor battery and
  895      * temperature, changing the interval and the argument (0-1) to distinguish
  896      * between every case. */
  897     TD_SCHEDULER_Append(T_CHK_INTERVAL, 0 , 0, TD_SCHEDULER_INFINITE,
ThresholdFunction,
  898             TEMPERATURE);
  899
  900     /* Initialize the timer with infinite repetitions and RTC_INTERVAL s interval (1
hour;
  901      * 3600 seconds). The timer callback is TimerFunction. */
  902     TD SCHEDULER Append(RTC INTERVAL, 0 , 0, TD SCHEDULER INFINITE, TimerFunction, 0);
  903
  904     /* Send the first message (12 bytes maximum; it uses 8 here) to SigFox backend,
with one
  905      * repetition. LED will be switched on during the transmission. */
  906     tfp printf("Sending message to backend: It works.\r\n\r\n");
  907     GPIO PinOutSet(LED PORT, LED BIT);
  908     TD_SIGFOX_Send((uint8_t *) "It works", WM_MSG_LENGTH, 1);
  909     GPIO_PinOutClear(LED_PORT, LED_BIT);
  910 }
```

C.3.8.6. static void ThresholdFunction (uint32_t *arg*, uint8_t *repetition*) [static]

Function ThresholdFunction. Timer callback for monitoring battery and temperature. It will be called every 30 minutes.

It will use the same function to monitor battery and temperature events. In both cases, it will check if it has reached any threshold which will provoke a state change. If it happens, it will send a frame to SigFox backend with the value and the new state.

Parameters:

in	*arg*	Argument passed by the Scheduler. It will be 0 if it is monitoring battery level (BATTERY), and 1 for temperature (TEMPERATURE).
in	*repetition*	Argument passed by the Scheduler (never been used).

```
650 {
651     // Value to compare with thresholds.
652     uint32 t value = 0;
653
654     // Boolean to indicate if is has to send a frame to SigFox backend.
655     bool send frame = false;
656
657     // Depending on arg value, it will check battery (0) or temperature (1) level.
658     if (arg == BATTERY) {
659
660         // It's battery.
661         value = (uint32 t) TD MEASURE VoltageTemperatureExtended(false);
662
663         // Then, it checks if it has reached any threshold.
664         if (value < LOW B LEVEL) {
665
666             /* It has detected a low level. To send a frame, it's neccesary
667              * that the current state was OK level. */
668             if (battery state == OK BATTERY) {
669
670                 tfp printf("Low battery event. Value: %d mV\r\n", value);
671                 battery_state = LOW_BATTERY;    // New state.
672                 send_frame = true;
673             }
674         }
675         else if (value > OK_B_LEVEL) {
676
677             /* The same as battery low level event. It needs to come from
678              * battery low level event to send a frame with information
679              * about OK level event.
680              * Notice that LOW/OK B LEVEL values are different. So that,
681              * even it exceeds LOW_B_LEVEL, it must reach OK_B_LEVEL value
682              * to make the change (it's such a basic hysteresis example). */
683             if (battery_state == LOW_BATTERY) {
684
685                 tfp printf("Ok battery event. Value: %d mV\r\n", value);
686                 battery state = OK BATTERY;     // New state.
687                 send frame = true;
688             }
689         }
690     }
691     else {
692         // It's temperature.
693         value = (uint32_t) TD_MEASURE_VoltageTemperatureExtended(true);
694
695         // Then, it checks if it has reached or exceeded any threshold.
696         if (value < LOW T LEVEL) {
697
698             /* To notify the event, the current state must be different
699             than LOW_TEMP. */
700             if (temp_state != LOW_TEMP) {
701
702                 tfp printf("Low temperature event. Value: %d.%d degrees\r\n",
703                         value/10, value%10);
704                 temp_state = LOW_TEMP;      // New state.
705                 send_frame = true;
706             }
707         }
708         else if (value >= OK T LEVEL DOWN && value <= OK T LEVEL UP) {
709
710             /* To notify the event, the current state must be different
711             than OK_TEMP.
712             Here there is another hysteresis example, because thresholds
```

```
  713             are different from LOW_T_LEVEL and HIGH_T_LEVEL. */
  714             if (temp_state != OK_TEMP) {
  715 
  716                 tfp_printf("Ok temperature event. Value: %d.%d degrees\r\n",
  717                         value/10, value%10);
  718                 temp_state = OK_TEMP;       // New state.
  719                 send_frame = true;
  720             }
  721         }
  722         else if (value > HIGH_T_LEVEL) {
  723 
  724             /* To notify the event, the current state must be different
  725             than HIGH_TEMP. */
  726             if (temp_state != HIGH_TEMP) {
  727 
  728                 tfp_printf("High temperature event. Value: %d.%d degrees\r\n",
  729                         value/10, value%10);
  730                 temp_state = HIGH_TEMP;     // New state.
  731                 send_frame = true;
  732             }
  733         }
  734     }
  735 
  736     /* And if it has to send a frame, it will do it, with format:
  737      * -Character T/B (1B)
  738      * -Value of battery/temp state (1B)
  739      * -Value of temperature/battery (4B)
  740      *  = 6B
  741      * (It doesn't need sequence_number here, because it's just an alarm for the
backend).
  742      */
  743     if (send_frame == true) {
  744 
  745         // Get relative time for printing.
  746         now = TD_SCHEDULER_GetTime() >> 15;
  747 
  748         /* Print information about the event.
  749          * After that, it puts the variables in the correct order to make the message.
*/
  750         if (arg == BATTERY) {
  751             tfp_printf("ThresholdFunction (battery case). Time %d\r\n", now);
  752 
  753             message[0] = (uint8_t) 'B';
  754             message[1] = battery_state;
  755         }
  756         else {
  757             tfp_printf("ThresholdFunction (temperature case). Time %d\r\n", now);
  758 
  759             message[0] = (uint8_t) 'T';
  760             message[1] = temp_state;
  761         }
  762         message[2] = value >> 24;
  763         message[3] = value >> 16;
  764         message[4] = value >> 8;
  765         message[5] = value;
  766 
  767         // And then, it sends the message.
  768         tfp_printf("Sending message to backend with monitoring data.\r\n\r\n");
  769         GPIO_PinOutSet(LED_PORT, LED_BIT);
  770         TD_SIGFOX_Send(message, BT_MSG_LENGTH, 1);
  771         GPIO_PinOutClear(LED_PORT, LED_BIT);
  772     }
  773 }
```

C.3.8.7. static void TimerFunction (uint32_t *arg*, uint8_t *repetition*) [static]

Function TimerFunction. Timer callback for RTC. It will be called every hour.

It will always send a frame to SigFox backend with temperature and battery values, using a sequence number to order every pair of values in the server to print statistics properly.

Parameters:

in	*arg*	Argument passed by the Scheduler (always 0, never been used).
in	*repetition*	Argument passed by the Scheduler (never been used).

```
794 {
795     // Firstly, it has to increment the sequence number.
796     sequence_number++;
797
798     /* Print information about the repetition. Remember time is relative (when
799      * the program starts, the time is set to zero, and it increments every
800      * second). */
801     now = TD_SCHEDULER_GetTime() >> 15;
802     tfp_printf("TimerFunction. SN %d. Time %d\r\n", sequence_number, now);
803
804     /* Print the current temperature (in degrees, with decimals) and voltage
805      * (power supply). */
806     temperature = (uint32 t) TD MEASURE VoltageTemperatureExtended(true);
807     battery = (uint32_t) TD_MEASURE_VoltageTemperatureExtended(false);
808
809     tfp printf("Temperature: %d.%d degrees\r\n", temperature/10, temperature%10);
810     tfp printf("Power supply voltage: %d mV\r\n", battery);
811
812     /* And finally, it sends all to SigFox backend in a package with format:
813      * 'R' (1B) + sequence_number (3B) + temp in 10*degrees (4B) + voltage in mV
814      * (4B) = 12B (maximum packet length).
815      * It only has to put the variables in the correct order (remember there are
816      * global variables). */
817     message[0] = 'R';
818     message[1] = sequence number >> 16;
819     message[2] = sequence_number >> 8;
820     message[3] = sequence_number;
821     message[4] = temperature >> 24;
822     message[5] = temperature >> 16;
823     message[6] = temperature >> 8;
824     message[7] = temperature;
825     message[8] = battery >> 24;
826     message[9] = battery >> 16;
827     message[10] = battery >> 8;
828     message[11] = battery;
829
830     /* After that, it sends the message to SigFox backend, with one repetition, and
831     using the LED. */
832     tfp printf("Sending message to backend with all the information.\r\n\r\n");
833     GPIO PinOutSet(LED PORT, LED BIT);
834     TD SIGFOX Send(message, MSG LENGTH, 1);
835     GPIO PinOutClear(LED PORT, LED BIT);
836 }
```

C.3.9. Variable Documentation

uint32_t battery

Current battery level, in mV.

uint8_t battery_state = OK_BATTERY

Current battery state (OK, by default).

bool fixing_enabled = true

Flag to enable (true) GPS fixes. After movement detection, it will not initialize another GPS fixing (false) until it has the position.

uint8_t lat_direction

Latitude direction (N if positive, S if negative).

uint32_t latitude

First well-known latitude, with format: XX degrees XX.XXXXX minutes.

uint8_t long_direction

Longitude direction (E if positive, W if negative).

uint32_t longitude

First well-known longitude, with format: XXX degrees XX.XXXXX minutes.

uint8_t message[MSG_LENGTH]

Message sent to SigFox backend.

uint8_t movements = 0

Number of movements captured. When it reaches MAX_MOV, it sends a message to the backend and initialize again to 0.

uint32_t now = 0

Current moment, in seconds, since the EVB starts its execution (for printf).

uint32_t sequence_number = 0

Number of messages which have been sent to backend by TimerFunction. It has uint32_t type, but it uses 24B in the message. With that figure, and 140 messages maximum every day, it would need more than 300 years to reach its limit. So that, we can suppose it will never reach that. It is used as a sequence number for the server which processes all the value to show statistics.

uint8_t temp_state = OK_TEMP

Current temperature state (OK, by default).

uint32_t temperature

Current temperature level, in 10*degrees.

ANEXO D. CÓDIGO DEL SERVIDOR

En este anexo, especificaremos la **codificación** seguida para el servidor, que se encarga de recibir los datos del backend, procesarlos, guardarlos en la base de datos, y mostrarlos en un panel de estadísticas para el usuario.

No se incluirán capturas de pantalla; puesto que las capturas que resulten de interés para el proyecto se mostrarán en la sección de pruebas del código.

Para ello, distinguimos tres **partes**; de las cuales, dos soportan toda la carga de trabajo antes mencionada (una para el **procesado e inserción** en la base de datos, y otro para la **presentación de estadísticas**), y la tercera del **formato de presentación**.

Como documentación general; citar el **manual de PHP**, accesible desde su página oficial [49]. También se ha revisado la sintaxis del código en PHP con la utilidad [50]. En cada una de las partes, se mencionará la documentación adicional consultada.

D.1. Procesado e inserción. Fichero receiveMessages.php

La siguiente aplicación web, codificada íntegramente en PHP junto a HTML con interacción con la base de datos en MySQL, se encarga de **recibir** el mensaje SigFox, procedente del backend, **procesarlo** convenientemente en función del tipo de mensaje que sea, y **almacenarlo** en la tabla de la base de datos correspondiente. Veremos este proceso paso a paso:

1. En primer lugar, recibimos los **atributos** enviados por el backend, que serán el **instante de tiempo** en el que el mensaje llegó al backend, y la **carga útil**. Ambos atributos son enviados en una respuesta HTTP POST. Posteriormente, transformaremos el instante de tiempo en un formato fecha-hora, para almacenarlo en la base de datos.

```
  001 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  002
  003 <html xmlns="http://www.w3.org/1999/xhtml">
  004
  005    <head>
  006
  007        <title>SigFox messages receiver</title>
  008        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  009        <link rel="stylesheet" type="text/css" href="estilo.css" />
  010
  011    </head>
  012
  013    <body>
  014
  015        <?php
  016
  017            // Receive message's time and data from the backend.
  018
  019            $_time = $_POST["time"];
  020            $_data = $_POST["data"];
  021
  022            // Adapt the time to a valid date format to save it in the database.
  023
  024            $date = date('Y-m-d H:i:s', $_time);
  025
```

2. Abrimos la **conexión con la base de datos**. Tras ello, procedemos a leer los 2 primeros caracteres que

vienen en la carga útil, que corresponden al **tipo de mensaje enviado**. En función de su valor, haremos un procesado u otro. Recordar que dichos caracteres siguen el formato definido en [32].

```
026             // Database connection and selection.
027
028             $link = mysql_connect("localhost", "user", "password");
029             mysql_select_db("tfg", $link);
030
031             // Obtain message's header, in order to know what to do with the message.
032
033             $header = substr($_data, 0, 2);
034
035             /* Depending on header's value, we have to do a particular action to recover
all the
036             information and save it in the database.
037             This value represents a particular letter, following the Recommendation T-50
(ITU). */
038
039             switch ($header) {
040
```

a. Si se ha producido un **cambio en la posición** del dispositivo o es la **primera posición conocida**, capturaremos la **latitud y longitud**, la transformaremos en el formato grados-minutos, y lo pasaremos a **forma decimal**. Después, comprobaremos si se debe cambiar el signo de alguno de los valores, en función del valor de orientación recibido. Con esto, guardamos los valores en la tabla **maps** de la base de datos, junto a la **fecha-hora** en la que se capturó.

```
041                 case "41":
042
043                     // Position change detected.
044
045                     $lat = hexdec(substr($ data, 2, 8));    // Obtaining the latitude.
046                     $long = hexdec(substr($_data, 12, 8));  // Obtaining the longitude.
047
048                     // Then, divide them into degrees and minutes.
049
050                     $lat_deg = floor($lat/10000000);
051                     $lat_min = ($lat%10000000)/100000.0;
052                     $long_deg = floor($long/10000000);
053                     $long_min = ($long%10000000)/100000.0;
054
055                     // After that, convert latitude and longitude to decimal values.
056
057                     $lat = $lat_deg + $lat_min/60.0;
058                     $long = $long_deg + $long_min/60.0;
059
060                     // Change latitude sign if the direction is 'S'.
061
062                     if (substr($ data, 10, 2) == "53") {
063                         $lat = -$lat;
064                     }
065
066                     // Change longitude sign if the direction is 'W'.
067
068                     if (substr($_data, 20, 2) == "57") {
069                         $long = -$long;
070                     }
071
072                     // Put the values into the table called maps.
073
074                     mysql_query("INSERT INTO maps (date, latitude, longitude) VALUES
075                       ('".$date."', '".$lat."', '".$long."')", $link);
076
077                     break;
078
079                 case "47":
080
081                     // First well-known position detected.
082
083                     $lat = hexdec(substr($ data, 2, 8));    // Obtaining the latitude.
084                     $long = hexdec(substr($ data, 12, 8));  // Obtaining the longitude.
085
086                     // Then, divide them into degrees and minutes.
087
```

```
088                     $lat_deg = floor($lat/10000000);
089                     $lat_min = ($lat%10000000)/100000.0;
090                     $long_deg = floor($long/10000000);
091                     $long_min = ($long%10000000)/100000.0;
092
093                     // After that, convert latitude and longitude to decimal values.
094
095                     $lat = $lat_deg + $lat_min/60.0;
096                     $long = $long_deg + $long_min/60.0;
097
098                     // Change latitude sign if the direction is 'S'.
099
100                     if (substr($ data, 10, 2) == "53") {
101                         $lat = -$lat;
102                     }
103
104                     // Change longitude sign if the direction is 'W'.
105
106                     if (substr($_data, 20, 2) == "57") {
107                         $long = -$long;
108                     }
109
110                     // Put the values into the table called maps.
111
112                     mysql_query("INSERT INTO maps (date, latitude, longitude) VALUES
113                       ('".$date."', '".$lat."', '".$long."')", $link);
114
115                     break;
116
```

b. Si se ha detectado **movimiento** en la placa o es el **primer mensaje** enviado por el dispositivo, formaremos un mensaje personalizado según el caso dado, y lo añadiremos a la tabla **alarms** de la base de datos, junto a la **fecha-hora** en la que se produjo.

```
117                 case "4d":
118
119                     // Movement detected.
120
121                   // Form a particular message to this event.
122
123                     $message = "Movement detected.";
124
125                     // Put the values into the table called alarms.
126
127                     mysql query("INSERT INTO alarms (date, message) VALUES
128                       ('".$date."', '".$message."')", $link);
129
130                     break;
131
132                 case "49":
133
134                     // First message sent.
135
136                     // Form a particular message to this event.
137
138                     $message = "It works.";
139
140                     // Put the values into the table called alarms.
141
142                     mysql query("INSERT INTO alarms (date, message) VALUES
143                       ('".$date."', '".$message."')", $link);
144
145                     break;
146
```

c. Si estamos en un caso de **alarma de batería o temperatura**, el mensaje personalizado incluirá el **caso** que se haya producido (batería baja/correcta y temperatura baja/correcta/alta) y el **valor** de batería/temperatura alcanzado (convertidos a voltios y grados, respectivamente). Estos datos, junto a la **fecha-hora** en que se produjeron, también se añadirán a la tabla **alarms** de la base de datos.

```
147                 case "42":
148
149                     // Battery event.
150
```

```
  151                     // Transform hexadecimal values (state and value) into decimal
values.
  152
  153                     if (hexdec(substr($ data, 2, 2)) == 0) {
  154                         $message = "Low battery event. Value: ";
  155                     }
  156                     else {
  157                         $message = "Ok battery event. Value: ";
  158                     }
  159
  160                     $value = hexdec(substr($_data, 4, 8))/1000.0;
  161
  162                     // And form the message.
  163
  164                     $message = $message . $value . " V";
  165
  166                     // Put the values into the table called alarms.
  167
  168                     mysql_query("INSERT INTO alarms (date, message) VALUES
  169                       ('".$date."', '".$message."')", $link);
  170
  171                     break;
  172
  173                 case "54":
  174
  175                     // Temperature event.
  176
  177                     // Transform hexadecimal values (state and value) into decimal
values.
  178
  179                     if (hexdec(substr($_data, 2, 2)) == 0) {
  180                         $message = "Low temperature event. Value: ";
  181                     }
  182                     else if (hexdec(substr($_data, 2, 2)) == 1) {
  183                         $message = "Ok temperature event. Value: ";
  184                     }
  185                     else {
  186                         $message = "High temperature event. Value: ";
  187                     }
  188
  189                     $value = hexdec(substr($ data, 4, 8))/10.0;
  190
  191                    // And form the message.
  192
  193                     $message = $message . $value . " °C";
  194
  195                     // Put the values into the table called alarms.
  196
 197                   mysql query("INSERT INTO alarms (date, message) VALUES
  198                       ('".$date."', '".$message."')", $link);
  199
  200                     break;
  201
```

d. Por último; si estamos ante un **evento del RTC**, capturaremos todos los datos (número de secuencia, temperatura y batería) y los almacenaremos en la tabla **graphics** de la base de datos, junto a la **fecha-hora** de ocurrencia.

```
  202                 case "52":
  203
  204                     // RTC event.
  205
  206                     /* Obtain every value, transform them into a decimal value, and save
each one
  207                     in a variable. */
  208
  209                     $sn = hexdec(substr($_data, 2, 6));
  210                     $temperature = hexdec(substr($_data, 8, 8))/10.0; // In degrees.
  211                     $battery = hexdec(substr($ data, 16, 8))/1000.0;  // In volts.
  212
  213                     // Put the values into the table called graphics.
  214
  215                     mysql_query("INSERT INTO graphics (id, date, temperature, battery)
VALUES
  216                       ('".$sn."', '".$date."', '".$temperature."', '".$battery."')",
$link);
  217
```

```
                    break;
```

e. Si la carga útil tiene una **cabecera distinta** a las anteriores, **no se procesarán** los datos.

```
                default:

                    // Nothing to do here.

                    break;
            }
```

3. Por último, **cerramos la sesión abierta** con la base de datos.

```
            // Close the database.

            mysql_close($link);
        ?>

</body>

</html>
```

Finalmente, comentar varios aspectos relacionados con la codificación antes mencionada:

- Notar que **no se presenta ningún contenido** al usuario si intenta acceder a la URL que identifica a esta aplicación web. El propósito de la aplicación se limita a recibir los datos y a añadirlos en la base de datos.
- Se da por hecho que los mensajes recibidos siguen, exclusivamente, la **codificación definida** en el código fuente descrito en el **Anexo C**. Si el mensaje saliese de ese formato; se ignoraría, o bien podría dar lugar a errores en el almacenamiento en la base de datos si los 2 primeros caracteres fueran iguales a alguno de los casos considerados.
- Conectando con la anterior cuestión, también se da por hecho que todos los mensajes recibidos proceden del **backend de SigFox**. Cualquier envío a la aplicación de respuestas HTTP POST que no tengan ese origen pueden sufrir los mismos inconvenientes que los comentados anteriormente.

D.2. Presentación de estadísticas. Fichero index.php

Con el paso anterior, hemos conseguido almacenar los datos en sus respectivas tablas de la base de datos, decodificando la carga útil que enviaba la placa para obtener los valores reales enviados.

Esos datos se **presentarán** a través de esta aplicación web. Además de PHP-MySQL y HTML, se utiliza contenido **Javascript** para generar **gráficas** con la librería de Highcharts [51] y para mostrar un **mapa** con las posiciones capturadas, a partir de la API de Google Maps [52]. Comenzamos con su análisis detallado:

1. En **head**, se añaden las **librerías Javascript** que harán uso de la API de Google Maps, de las funciones jQuery y del código de Highcharts para la generación de gráficas. Además, definimos la **hoja de estilos** a seguir (que se presentará en el siguiente apartado). Como último detalle, la segunda etiqueta **meta** define cada cuánto tiempo se **recarga automáticamente** la página, fijándolo a un valor de 10 minutos (600 segundos).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>SigFox dashboard</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="refresh" content="600" />
<link rel="stylesheet" type="text/css" href="style.css" />
<script src="http://maps.googleapis.com/maps/api/js"></script>
<script src="https://code.jquery.com/jquery.js"></script>
<script src="https://code.highcharts.com/stock/highstock.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
```

```
015
016 </head>
```

2. La **cabecera** de la aplicación web mostrará el **título de la página**.

```
017
018 <body>
019
020 <div id="header">
021
022 <h1>SigFox dashboard</h1>
023
024 </div>
025
026
```

3. En la siguiente sección, se define el **contenido** a mostrar, distinguiendo entre varias subsecciones.
    a. Primero, abrimos la conexión a la **base de datos** que contiene toda la información.

```
027 <div id="content">
028
029 <?php
030
031                 // Database connection and selection.
032
033                 $link = mysql connect("localhost", "user", "password");
034                 mysql select db("tfg", $link);
035
036             ?>
037
```

    b. Como primera subsección, tenemos la **generación de gráficas** de los niveles de **temperatura y batería** obtenidos.

```
038 <div id="graphics">
039
040 <h2>Graphics</h2>
041
```

        i. Estos valores, junto a la fecha-hora (que se transforma en marca de tiempo de nuevo, ya que es el formato empleado en la generación de las gráficas) y el número de secuencia, se **leerán de la base de datos** y se guardarán en un array para su posterior presentación. La variable empty_graphics valdrá 1 si la tabla de la base de datos está vacía, y 0 en caso contrario.

```
042 <?php
043
044                     // Look for all the values and store them in arrays.
045
046                     $result = mysql_query("SELECT * FROM graphics ORDER BY date ASC",
$link);
047
048                     $empty graphics = 1;
049
050                     $sn = array();
051                     $dates = array();
052                     $temperature = array();
053                     $battery = array();
054
055                     while($row = mysql_fetch_array($result)) {
056
057                         if ($empty_graphics == 1) {
058                             $empty graphics = 0;
059                         }
060
061                         $sn[] = $row['id'];
062
063                         // Convert the date to a timestamp value, adding 2 hours to
have the Spanish official time.
064                         $date = new DateTime($row['date']);
065                        date add($date, date interval create from date string('2
hours'));
066                        $date = $date->getTimestamp()*1000;
067
068                        $dates[] = $date;
```

```
  069                         $temperature[] = $row['temperature'];
  070                         $battery[] = $row['battery'];
  071                     }
  072
```

ii. Si la **tabla** estaba **vacía**, insertaremos en la página un **párrafo** indicando este hecho. De haber algún valor, generaremos las gráficas para cada caso. Notar que, de estar la tabla vacía, no se generaría ninguna gráfica, por cómo se definen los bloques if-else.

```
  073                     // Put a paragraph if the table is empty.
  074
  075                     if ($empty_graphics == 1) {
  076                 ?>
  077<p class="error">No graphic values.</p>
  078<?php
  079
  080                     }
  081
```

iii. A la hora de generar las gráficas, distinguiremos los casos de temperatura y batería, para generar **dos gráficas separadas**.

Primero, generaremos la **gráfica de niveles de temperatura**, usando Javascript para la gráfica y PHP para introducir los datos junto a la marca de tiempo uno por uno con un bucle for.

```
  082                     // If there are values in the table, it will show them using
graphics.
  083
  084                     else {
  085                 ?>
  086<div id="temperature">
  087<script>
  088
  089                             chartCPU = new Highcharts.StockChart({
  090                                 chart: {
  091                                     renderTo: 'temperature'
  092                                 },
  093
  094                                 rangeSelector : {
  095                                     enabled: false
  096                                 },
  097
  098                                 title: {
  099                                     text: 'Temperature'
  100                                 },
  101
  102                                 xAxis: {
  103                                     type: 'datetime'
  104                                 },
  105
  106                                 yAxis: {
  107                                     minPadding: 0.2,
  108                                     maxPadding: 0.2,
  109                                     title: {
  110                                         text: 'Values (°C)',
  111                                         margin: 10
  112                                     }
  113                                 },
  114
  115                                 series: [{
  116                                     name: 'Value',
  117                                     data: (function() {
  118                                             var data = [];
  119
  120<?php
  121                                                 // Put the values, which are in their
respective array, into the graphic.
  122                                                 for ($i = 0; $i < count($sn); $i++) {
  123
  124                                             ?>
  125
  126                                             data.push([<?php echo $dates[$i];?>,<?php
echo $temperature[$i];?>]);
  127
  128<?php
```

```
129
130                                                 }
131
132                                             ?>
133                                             return data;
134                                         })()
135                                 }],
136
137                                 credits: {
138                                         enabled: false
139                                 }
140                             });
141
142</script>
143</div>
144
```

iv. Tras la de temperatura, se genera la **gráfica de niveles de batería**. Sigue el mismo modelo que la gráfica de temperaturas, pero usando el array que contiene los valores del nivel de batería en vez del nivel de temperatura.

```
145<div id="battery">
146<script>
147
148                             chartCPU = new Highcharts.StockChart({
149                                 chart: {
150                                     renderTo: 'battery'
151                                 },
152
153                                 rangeSelector : {
154                                     enabled: false
155                                 },
156
157                                 title: {
158                                     text: 'Battery'
159                                 },
160
161                                 xAxis: {
162                                     type: 'datetime'
163                                 },
164
165                                 yAxis: {
166                                     minPadding: 0.2,
167                                     maxPadding: 0.2,
168                                     title: {
169                                         text: 'Values (V)',
170                                         margin: 10
171                                     }
172                                 },
173
174                                 series: [{
175                                     name: 'Value',
176                                     data: (function() {
177                                             var data = [];
178
179<?php
180
181                                                 // Do the same for battery values.
182
183                                                 for ($i = 0; $i < count($sn); $i++) {
184
185                                             ?>
186
187                                             data.push([<?php echo $dates[$i];?>,<?php
echo $battery[$i];?>]);
188
189<?php
190
191                                                 }
192
193                                             ?>
194                                             return data;
195                                         })()
196                                 }],
197
198                                 credits: {
199                                         enabled: false
```

```
  200                                 }
  201                             });
  202
  203</script>
  204</div>
```

v. Finalmente, se mirarán los **números de secuencia** almacenados para comprobar si ha habido un **salto** en el valor del mismo; lo que indicaría **pérdida de información**, ya que este número se incrementa en una unidad con cada envío. Por cada ocurrencia de este tipo, se mostrará un párrafo indicativo (debajo de las dos gráficas, al ser el último bloque div).

Otro caso posible sería que, si se iniciase una **nueva transmisión**, se volvería a empezar por 1, y se notificaría el salto, al romperse la secuencia.

```
  205
  206<div id="sequence">
  207
  208<?php
  209
  210                                 // After that, it will check if there were any loss of
sequence.
  211
  212                                 for ($i = 0; $i < count($sn)-1 ; $i++) {
  213                                         $before = $sn[$i];
  214                                         $after = $sn[$i+1];
  215
  216                                         // To have a loss, after-before must be
different than 1.
  217
  218                                         if ($after - $before != 1) {
  219                             ?>
  220<p class="error">Jump in sequence number: From <?php echo $before; ?> to <?php echo
$after; ?>.</p>
  221<?php
  222                                         }
  223                                 }
  224                             ?>
  225</div>
```

vi. Antes de pasar a la siguiente subsección, **liberamos los recursos** asociados a la consulta, para usar la misma variable en la siguiente consulta.

```
  226<?php
  227
  228                     }
  229
  230                     // Release resources.
  231
  232                     mysql_free_result($result);
  233                 ?>
  234
  235</div>
  236
```

c. La siguiente subsección trabaja con los **mensajes de alarma** generados por la placa; que son el primer mensaje enviado tras iniciarse, la detección de movimiento, y las alarmas de batería y temperatura por cambio de estado.

```
  237<div id="alarms">
  238
  239<h2>Alarms</h2>
  240
```

i. Listaremos todos los mensajes guardados, junto a su fecha, y mostraremos cada **alarma** en un **párrafo,** mostrando primero los mensajes más antiguos. La variable empty_alarms valdrá 1 si la tabla de la base de datos está vacía, y 0 si tiene algún valor almacenado.

```
  241<?php
  242
  243                     // Look for all alarm's messages.
  244
  245                     $result = mysql_query("SELECT * FROM alarms", $link);
  246
```

```
  247                     $empty_alarms = 1;
  248
  249                     /* Show the messages using HTML paragraphs, separated by a line.The
order is chronological, from
  250                     older to newer messages. */
  251
  252                     while($row = mysql_fetch_array($result)) {
  253
  254                         if ($empty alarms == 1) {
  255                             $empty alarms = 0;
  256                         }
  257
  258                         echo "<p><strong>".$row['date']."</strong></p>";
  259                         echo "<p>   ".$row['message']."</p>";
  260                         echo "<hr/>";
  261                     }
  262
```

ii. Si la **tabla** estuviese **vacía**, se mostrará un **párrafo** indicando este hecho. Notar que, como en el caso de las gráficas, tampoco se mostraría ningún contenido adicional en este caso.

```
  263                     // Put a paragraph if the table is empty.
  264
  265                     if ($empty alarms == 1) {
  266                 ?>
  267<p class="error">No alarm messages.</p>
  268<?php
  269
  270                     }
  271
```

iii. Por último, **liberamos los recursos** para realizar la última consulta.

```
  272                     // Release resources.
  273
  274                     mysql_free_result($result);
  275
  276                 ?>
  277
  278</div>
  279
```

d. La última subsección contempla los casos de primera posición conocida y cambio de posición. Para ello, se emplea un **mapa** generado con la API de Google Maps de Javascript.

```
  280<div id="maps">
  281
  282<h2>Maps</h2>
  283
```

i. Primero, **listamos** todas las posiciones almacenadas.

```
  284<?php
  285
  286                     // Look for latitude and longitude values.
  287
  288                     $result = mysql query("SELECT * FROM maps", $link);
  289
```

ii. A continuación, leemos la **primera fila** de la tabla de la base de datos, que nos servirá para ampliar el mapa en la zona que rodea al punto almacenado en dicha fila.

```
  290                     /* It will show the map if there is any value stored in the table.
After that, it uses initialize
  291                     Javascript function to set map's properties and create it. */
  292
  293                     if ($row = mysql_fetch_array($result)) {
  294                 ?>
  295
```

iii. **Inicializamos el mapa** usando Javascript, situando como centro el primer punto leído (con PHP).

```
  296<script>
  297
  298                         function initialize() {
```

```
  299
  300                             var mapProp = {
  301                                 center: new google.maps.LatLng(<?php echo
$row['latitude']; ?>,<?php echo $row['longitude']; ?>),
  302                                 zoom: 14,
  303                                 mapTypeId: google.maps.MapTypeId.ROADMAP
  304                             };
  305
  306                             var map = new
google.maps.Map(document.getElementById("googleMap"),mapProp);
  307
```

iv. Añadiremos un **marcador** para cada punto almacenado en la tabla de la base de datos. Como ya hemos leído la primera fila, y necesitamos que lea las demás, usaremos un bucle do-while. El **título** del marcador será un **número de secuencia**, para indicar el orden de captura de la posición, y la **fecha-hora** de la captura.

```
  308<?php
  309
  310                     // It will put as many markers as rows in this table.
  311
  312                     $seq number = 1;
  313
  314                     do {
  315
  316                ?>
  317                             var marker = new google.maps.Marker({
  318                                 position: new google.maps.LatLng(<?php echo
$row['latitude']; ?>,<?php echo $row['longitude']; ?>),
  319                                 map: map,
  320                                 title: <?php echo "\"".$seq_number.":
".$row['date']."\""; ?>
  321                             });
  322<?php
  323
  324                         $seq number++;
  325                     }
  326                     while ($row = mysql_fetch_array($result));
  327
```

v. Para terminar con el mapa, se añade un listener para **mostrarlo** cuando se cargue la página.

```
  328                     // Finally, it uses the listener to listen/bind to DOM events.
  329                ?>
  330                         }
  331
  332                         google.maps.event.addDomListener(window, 'load', initialize);
  333
  334</script>
  335
  336<?php
  337
  338                     }
  339
```

vi. En el caso de que **no hubiese una primera posición** almacenada, se mostraría un **párrafo** indicando este hecho. Y tampoco, en este caso, se mostraría otro contenido que no fuese ese párrafo.

```
  340                     else {
  341
  342                         // Put a paragraph if the table is empty.
  343                ?>
  344<p class="error">No location values.</p>
  345
  346<?php
  347
  348                     }
  349
```

vii. Por último, **liberamos los recursos** asociados a la consulta, y definimos el **bloque** que contendrá el **mapa**, con su tamaño.

```
  350                     // Release resources.
  351
```

```
352                     mysql_free_result($result);
353
354                 ?>
355
356<div id="googleMap" style="width:350px;height:350px;"></div>
357
358</div>
359
```

e. Para terminar con el bloque de contenido, se **cerrará la conexión** con la base de datos.

```
360<?php
361
362                 // Close the database.
363
364                 mysql_close($link);
365             ?>
366
367</div>
368
369
```

4. Por último, se define el **pie de página**, que muestra un mensaje final que incluye el nombre de la placa de evaluación, las siglas TFG y el nombre del alumno.

```
370<div id="bottom">
371
372<p>TD1204 TFG. Made by Ramón Pérez Hernández.</p>
373
374</div>
375
376</body>
377
378</html>
379
```

D.3. Formato de presentación. Fichero style.css

Para presentar la aplicación anterior con un formato más legible para el usuario, se adjunta la **hoja de estilos** CSS empleada.

La **maquetación** realizada divide la página en las 3 zonas que ya se han enumerado en la descripción de index.php:

- **Cabecera:** contiene el título de la aplicación web.
- **Contenido**: muestra las distintas estadísticas; de izquierda a derecha: las gráficas de temperatura y batería, los mensajes de alarma, y las posiciones capturadas en un mapa.
- **Pie de página**:contiene un párrafo final como pie de página, con el nombre del alumno.

```
01 /* General. */
02
03       body {font-family: verdana,arial,sans-serif; font-size: 10pt;}
04
05 /* Content. */
06
07       h1 {font-size: 16pt; font-weight: bold; color: #0066CC; text-align: center;}
08       h2 {font-size: 14pt; font-weight: bold; color: red; text-align: center;}
09
10 /* Layout. */
11
12       #header {border-bottom: solid thin;}
13       #content {}
14       #graphics {float: left; width: 35%; margin: 1em;}
15       #alarms {float: left; width: 30%; margin: 1em;}
16       #maps {float: left; width: 25%; margin: 1em;}
17       #bottom {clear: both; border-top: solid thin; text-align: center; font-size: 80%;}
18
19 /* Classes. */
20
21       p.error {color: red; text-align: center;}
22
```

Notar que el objetivo de esta aplicación web no es mostrar un formato de presentación sofisticado, sino centrarse en la visualización correcta y diferenciada de cada uno de los mensajes capturados. De querer mejorar dicha presentación, bastaría con codificar el fichero CSS con el contenido deseado.

REFERENCIAS

(Referencias comprobadas por última vez el 7 de julio de 2014. No se asegura que el contenido de las mismas siga siendo el mismo pasada esta fecha).

[1] Redacción de Media-tics, «Abertis y SigFox lanzarán en España la primera red de Internet de las Cosas del mundo,»*artículo deMedia-tics*, 2014:http://www.media-tics.com/noticia/4393/internet/abertis-y-sigfox-lanzaran-en-espana-la-primera-red-de-internet-de-las-cosas-del-mundo.html

[2] Enlace al backend de SigFox (necesita identificación): https://backend.sigfox.com

[3] SDK de Telecom Design para la familia de módems TD12xx: http://rfmodules.td-next.com/sdk/

[4] Repositorio Github de Telecom Design:https://github.com/Telecom-Design/

[5] Dave Evans, «Internet de las Cosas. Cómo la próxima evolución de Internet lo cambia todo,» *informe técnico deCisco Internet Business Solution Group*, p. 2-3, 2011: http://www.cisco.com/web/LA/soluciones/executive/assets/pdf/internet-of-things-iot-ibsg.pdf

[6] Kevin Ashton, «That 'Internet of Things' Thing,» *artículo de RFID Journal*, 2009: http://www.rfidjournal.com/articles/view?4986

[7] Tomàs Delclós, «El reto del 'Internet de las Cosas',»*artículo de El País*, 2007:http://elpais.com/diario/2007/05/17/ciberpais/1179368665_850215.html

[8] Infografía con la historia cronológica del Big Data, por WinShuttle:http://www.winshuttle.es/big-data-historia-cronologica/

[9] David Cuen, «Los 4.300 millones de humanos sin Internet,»*artículo de BBC*, 2014:http://www.bbc.com/mundo/blogs/2014/04/140416_blog_un_mundo_feliz_sin_internet

[10] Nicolás Rivera, «Qué es el Internet of Things y cómo cambiará nuestra vida en el futuro,»*artículo de Hipertextual*, 2015:http://hipertextual.com/2015/06/internet-of-things

[11] Aplicaciones de Internet de las Cosas, según SigFox:http://www.sigfox.com/es/#!/sectors

[12] Pablo Espeso, «Las 3 tecnologías clave para el Internet de las Cosas,»*artículo de Xakata*, 2015:http://www.xataka.com/internet-of-things/las-3-tecnologias-clave-para-el-internet-de-las-cosas

[13] Fundación de la Innovación Bankinter, «El Internet de las Cosas en un mundo conectado de objetos inteligentes,»*resumen ejecutivo*, p. 9-10, 2011:http://www.accenture.com/SiteCollectionDocuments/Local_Spain/PDF/Accenture_FTF_Internet_de_las_Cosas_2011.pdf

[14] Página oficial de ARM:http://www.arm.com/

[15] Jesús Ranz Abad, «Las Operadoras de Telecomunicaciones en Internet de las Cosas (IoT),»*artículo de ThingsCity*, 2013:http://www.thingscity.com/las-operadoras-de-telecomunicaciones-en-internet-de-las-cosas-iot/

[16] Página oficial de SigFox:http://www.sigfox.com/es/#!/

[17] Alex Davies, «On LPWANs: Why SigFox and LoRa are rather different, and the importance of the business model,» *artículo de ReTHINK Research*, 2015: http://www.rethinkresearch.biz/articles/on-lpwans-why-sigfox-and-lora-are-rather-different-and-the-importance-of-the-business-model/

[18] TD Next Modules, página oficial de la familia de módems TD12xx:http://rfmodules.td-next.com/

[19] SigFox Whitepaper: http://www.sigfox.com/static/media/Files/Documentation/SIGFOX_Whitepaper.pdf

[20] Página oficial de Cellnex Telecom:https://www.cellnextelecom.com/

[21] Página oficial de Telecom Design: http://www.telecom-design.com/

[22] Página oficial de Cloud-on-Chip: http://www.cloudonchip.com/

[23] Página de desarrolladores de soluciones IoT de Telecom Design:https://developers.insgroup.fr/

[24] Sección del TD1204 en TD Next Modules:http://rfmodules.td-next.com/modules/td1204/

[25] Sección del TD1204 en la web de desarrolladores de Telecom Design:https://developers.insgroup.fr/cloud-on-chip/td1204/evb.html#quickstart

[26] Sección del TD1204 en la página oficial de SigFox:http://www.sigfox.com/es/#!/products/td1204-30

[27] Centro de control de la placa de evaluación en la web de desarrolladores de Telecom Design:https://developers.insgroup.fr/dashboards/device.html

[28] Hilo en Taringa sobre los comandos Hayes:http://www.taringa.net/posts/info/15894960/Conjunto-de-comandos-Hayes-o-Comandos-AT.html

[29] Página de descarga de los drivers del cable USB:http://www.ftdichip.com/Drivers/VCP.htm

[30] Página de descarga de Ublox u-center:http://www.u-blox.com/en/evaluation-tools-a-software/u-center/u-center.html

[31] Página de descarga de PuTTY:http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[32] Enlace a la recomendación T-50 de la ITU para su descarga gratuita:https://www.itu.int/rec/T-REC-T.50-199209-I/es

[33] Página oficial de Arduino:https://www.arduino.cc/

[34] Esquemático del Arduino Uno R3:https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

[35] Datasheet del cable TTL-USB que incorpora el TD1204:http://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_CABLES.pdf

[36] Centro de control del desarrollador en la web de desarrolladores de Telecom Design:https://developers.insgroup.fr/dashboards/developer.html

[37] Blog Disk91.com, con aplicaciones reales de los módems TD12xx en SigFox:http://www.disk91.com/category/technology/sigfox/

[38] Otro enlace al SDK de Telecom Design, desde la página de desarrolladores de Telecom Design:https://developers.insgroup.fr/releases/2014/02/25/sdk-400-version-available/index.html

[39] Instalación básica de un servidor LAMP: https://www.atlantic.net/community/howto/installing-lamp-on-debian-8/

[40] Página de stackoverflow, para resolución de dudas: http://stackoverflow.com

[41] Datasheet del módem SIM900D: http://dl.btc.pl/kamami_wa/sim900d_hd.pdf

[42] GPRS. Página de la ETSI: http://www.etsi.org/technologies-clusters/technologies/mobile/gprs?highlight=YToxOntpOjA7czo0OiJncHJzIjt9

[43] Modelo de batería ER26500M. Página de compra: http://www.tme.eu/es/details/bat-er26500m_st-ul/pilas/ultralife/er26500mst/

[44] Bandas de frecuencia para telefonía móvil en España: http://wiki.bandaancha.st/Frecuencias_telefon%C3%ADa_m%C3%B3vil

[45] Cobertura aproximada de GSM. Del capítulo "El concepto de red celular": http://es.ccm.net/contents/681-estandar-gsm-sistema-global-de-comunicaciones-moviles

[46] Página de compra del SIM900D: http://www.tme.eu/es/details/sim900d/modulos-gsmgps/simcom/

[47] Modelo de batería ER26500. Página de compra: http://www.tme.eu/es/details/bat-er26500_st-

ul/pilas/ultralife/er26500st/

[48] Página oficial de Smartsheet: http://es.smartsheet.com/

[49] Manual de PHP:https://secure.php.net/manual/es/index.php

[50] Comprobación de sintaxis de código PHP: http://es.piliapp.com/php-syntax-check/

[51] Tutorial para generar gráficas dinámicas en función del tiempo:https://geekytheory.com/php-mysql-highcharts-mostrar-grafica-dinamica-en-funcion-del-tiempo/

[52] API de Google Maps en Javascript, en w3schools.com:http://www.w3schools.com/googleapi/

Glosario

- **Siglas**

ADC: Analog-to-Digital Converter.

API: Application Programming Interface.

ARM: Advanced RISC Machine.

ASCII: American Standard Code for Information Interchange.

CBS: Columbia Broadcasting System.

CEPT: Conférence Européenne des administrations des Postes et des Télécommunications.

CPU: Central Processing Unit.

CRA: Customer Response Applications.

CSS:Cascading Style Sheets.

DAC: Digital-to-Analog Converter.

DOM: Document Object Model.

ETSI: European Telecommunications Standards Institute.

EVB: EValuation Board.

FCC: Federal Communications Commission.

GCC: GNU Compiler Collection.

GNU: GNU is Not Unix.

GPIO: General Purpose Input/Output.

GPRS: General Packet Radio Service.

GPS: Global Positioning System.

GSM: Global System for Mobile communications.

HTML: HyperText Markup Language.

HTTP: HyperText Transfer Protocol.

IBSG: Internet Business Solutions Group.

IDE: Integrated Development Environment.

IoT: Internet of Things.

IP: Internet Protocol.

ISM: Industrial, Scientific and Medical.

ISP: In-System Programming.

ITU: International Telecommunication Union.

I^2C: Inter-Integrated Circuit.

JSON: JavaScript Object Notation.

LAMP: Linux, Apache, MySQL, and PHP.

LAN: Local Area Network.

LED: Light-Emitting Diode.

LPT: Line Print Terminal.

MIT: Massachusetts Institute of Technology.

M2M: Machine To Machine.

NFC: Near Field Communication.

NMEA: National Marine Electronics Association.

PAC: Porting Authorization Code.

PHP: PHP Hypertext Preprocessor.

RAE: Real Academia Española.

RAM: Random-Access Memory.

REST: REpresentational State Transfer.

RF: Radio Frequency.

RFID: Radio Frequency IDentification.

RISC: Reduced Instruction Set Computer.

RTC: Real-Time Clock.

SDK: Software Development Kit.

SIM: Subscriber Identify Module.

SNO: SigFox Network Operator.

SQL: Structured Query Language.

SSH: Secure SHell.

TD: Telecom Design.

TFG: Trabajo Fin de Grado.

TIC: Tecnologías de la Información y la Comunicación.

TTL: Transistor-Transistor Logic.

UART: Universal Asynchronous Receiver-Transmitter.

UML: Unified Modeling Language.

UNB: Ultra Narrow Band.

URL: Uniform Resource Locator.

USB: Universal Serial Bus.

- **Abreviaturas**

AT: Attention.

bin: Binary file.

bps: Bitpor segundo.

COM: Communications.

dB: Decibelio.

dBm: Decibelio-milivatio.

exe: Executable.

G: Fuerza G.

h: Horas.

ID Identificador.

KB: Kilobyte.

kbps: Kilobit por segundo.

KHz: Kilohercio.

MB: Megabyte.

MHz: Megahercio.

mV: Milivoltio.

mW: Milivatio.

mA: Miliamperio.

mAh: Miliamperio hora.

µA: Microamperio.

µAh: Microamperio hora.

s: Segundos.