ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Despertador inteligente basado en Arduino y aplicación Android

Arduino based smart alarm clock and Android application

Realizado por
Javier Luengo Romero

Tutorizado por
Daniel Garrido Márquez

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2019

Fecha defensa: de julio de 2017

Fdo. El/la Secretario/a del Tribunal

Resumen

Mediante el uso de la plataforma *Arduino* y el sistema operativo *Android*, el trabajo consiste en el desarrollo de software para un dispositivo que funciona a modo de despertador y para un móvil conectado a este. El despertador contará principalmente con capacidades de reloj y mostrará las notificaciones del teléfono. La aplicación *Android* realizará una conexión al despertador mediante *Bluetooth LE.*

El núcleo principal del trabajo consiste en la sucesión repetitiva de fases de desarrollo denominadas iteraciones, en las cuáles se van analizando las funcionalidades que se van a implementar, se diseñan las soluciones y se llevan a cabo, arreglando los problemas que se puedan ocasionar.

Debido a la metodología de desarrollo utilizada, las funcionalidades pueden cambiar respecto a las establecidas en un principio. Esto es debido a que en cada iteración aparecen complicaciones que ralentizan el desarrollo. También surgen Ideas alternativas que no se habían contemplado previamente y que son preferibles a las anteriores.

Palabras clave:

Arduino, *Android*, Bluetooth LE

Abstract

By using Arduino platform and Android operating system, the project consists in the development of software for a device that acts as an alarm clock and for a mobile connected to it. The alarm clock will mainly have clock capabilities and will show phone notifications. The Android application will make a connection to the alarm clock via Bluetooth LE.

The Main core of the work consists in the repetitive succession of development phases called iterations, in which the functionalities that are going to be implemented are analysed. The solutions are designed and carried out, fixing the problems that may arise.

Due to the development methodology used, the functionalities may change with respect to those initially established. This is due to complications that slow down the development in each iteration. Alternative ideas also arise that had not been previously contemplated and that are preferable to the previous ones.

Keywords:

Arduino, *Android*, Bluetooth LE

Índice

Índice de figuras

1

Introducción

1.1. Motivación

En los tiempos que corren nos encontramos rodeados de una infinidad de dispositivos electrónicos que nos permiten establecer un vínculo entre personas, empresas u otras máquinas. El número de estos dispositivos con los cuales interactuamos y las posibilidades que nos permiten no para de crecer. La llegada de las conexiones 5G y el Internet de las Cosas abre las puertas a controlar o monitorizar un sinfín de propiedades, espacios, electrodomésticos o incluso sistemas de salud [1]. Es decir, una amplia variedad de *cosas*. Esto se podrá hacer en cualquier momento y lugar donde nos encontremos.

Si un dispositivo destaca a día de hoy entre el resto por acompañar a la mayoría de personas y ser el orquestante de sus vidas digitales, este es el teléfono inteligente. Es por ello que surge la motivación de realizar un dispositivo que reduzca la carga que tienen estos teléfonos en las tareas relativas a obtener información, como pueden ser mensajes de texto, las últimas noticias, la información meteorológica o un gesto tan sencillo como mirar la hora.

1.2. Objetivos

El objetivo principal es el desarrollo de un sistema que sea de utilidad a usuarios que quieran evitar el uso directo de sus dispositivos móviles recibiendo información del teléfono en él, además de permitir un uso autónomo como reloj para ser colocado en un salón o dormitorio.

Para ello, en este trabajo se va a desarrollar *software* para un dispositivo *Arduino* y una aplicación para *Android*. La placa *Arduino* irá montada en una caja junto con un circuito electrónico conformando un prototipo de despertador inteligente sobre el que se desarrollará su software. Este se conectará a un teléfono *Android* por medio de una conexión Bluetooth LE con una aplicación desarrollada a tal efecto.

El móvil será capaz de enviar información al despertador; enviando por ejemplo notificaciones para que puedan ser leídas en una pantalla. El despertador debe ser capaz de responder ante esa entrada de información, por ejemplo, con una acción que permita descartar las notificaciones en el teléfono. En la Figura 1 se puede observar un modelo de dominio del sistema.

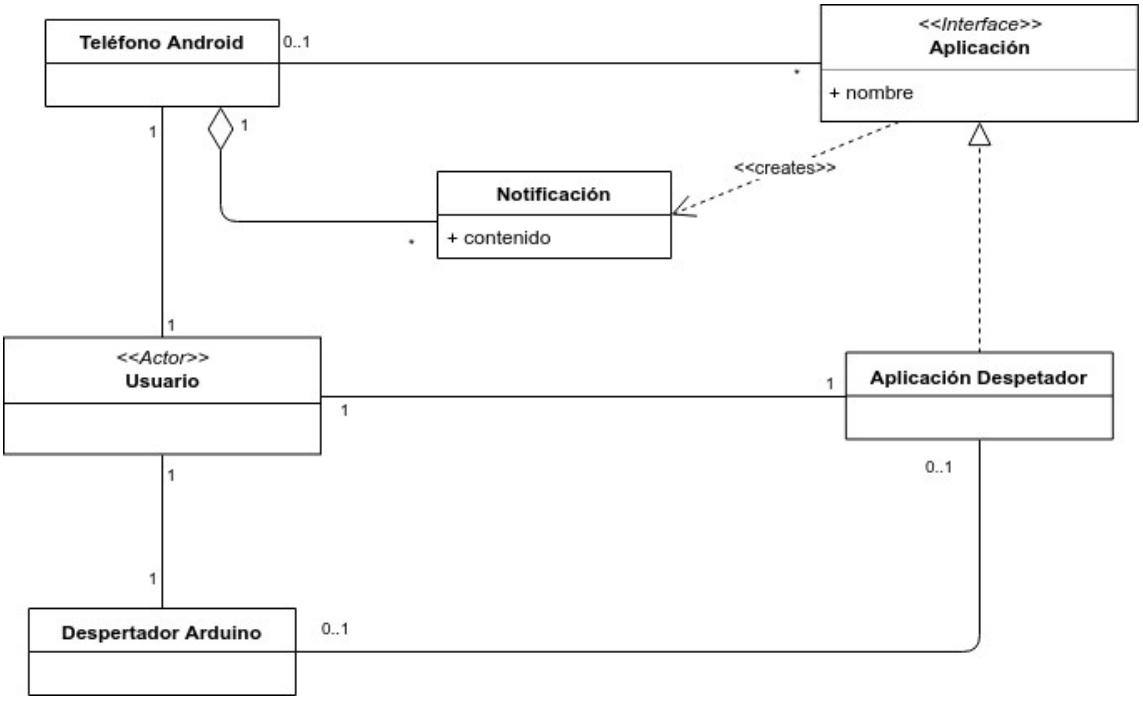

Figura 1: Modelo de dominio del sistema.

1.3. Estado del arte

Relativo a este trabajo es cierto que también han surgido otros conceptos similares conocidos como asistentes inteligentes [2] que también pretenden ocupar un hueco en los hogares y se encuentran conectados a Internet. Sin embargo un punto diferenciador es que estos carecen de la funcionalidad de mostrar información textual debido a que no tienen pantalla; su interfaz de control es

mediante comandos de voz. Además la forma de interactuar con ellos suele ser activa: es el usuario el que tiene que iniciar la interacción por medio de algún gesto o comando de voz. Por tanto, estos dispositivos no ofrecen información ni leen recordatorios o notificaciones a menos que interactúes previamente con ellos. Realmente son dispositivos que realizan acciones diferentes, aunque en ambos se persigue la finalidad de descargar al usuario del uso del teléfono móvil u otros aparatos similares.

1.4. Estructura de la memoria

En los primeros capítulos se explican las fases previas al propio desarrollo del Software: En el capítulo 2 se hace una pequeña introducción a la metodología de desarrollo de software que se ha llevado a cabo y su justificación debido a las características del proyecto. En el capítulo 3 se presentan los componentes hardware necesarios para la producción del prototipo del despertador así como una breve explicación proceso de creación del circuito. Por último, en el capítulo 4 se explican las diferentes tecnologías y plataformas informáticas que han sido necesarias para la realización del proyecto así como consideraciones específicas tomadas para este trabajo (Como pueden ser por ejemplo las limitaciones técnicas de *Arduino* o de su entorno de desarrollo oficial).

Tras estos apartados, en el capítulo 5 es donde sucede todo el grueso del trabajo. Se detalla el proceso de desarrollo de software divido en sus ciclos o iteraciones. En cada iteración se comentan cuáles son las tareas que se deben desarrollar acompañadas de diagramas conceptuales, se realiza una descripción de cómo ha sido el proceso de diseño y se describe la implementación de software realizada junto con diagramas, se prueba el funcionamiento y se realiza una evaluación del cumplimiento de las tareas para esa iteración.

Tras este capítulo se realiza una conclusión del trabajo realizado y se incluyen las referencias bibliográficas. Por último se incluyen dos anexos: el primero detalla el proceso de creación de un prototipo de despertador y el segundo es un manual de usuario del sistema.

2

Organización

2.1. Características principales del proyecto.

La mayor parte del trabajo consistirá en el diseño y desarrollo del *software*, el cuál se irá decidiendo en función de unos objetivos que se vayan actualizando conforme avanza el desarrollo. Este proceso de desarrollo conllevará el surgimiento de errores y problemas que deberán ser resueltos de la mejor forma posible. Es importante destacar la complejidad que conlleva un proceso de depuración y obtención de errores debido a que se desarrolla software conjunto para dos plataformas, siendo *Arduino* la que más complicaciones conlleva debido a la falta de una herramienta que asista la depuración.

2.2. Metodología de desarrollo utilizada

Tanto el desarrollo de *software* de *Arduino* como el de *Android* se han basado en la metodología de desarrollo iterativo incremental [3]. Esta metodología consiste principalmente en la sucesión de iteraciones, que son etapas de tiempo en las cuáles se presenta todo el ciclo de un desarrollo habitual. En la Figura 2 podemos ver como en cada una de estas iteraciones se priorizan las funcionalidades que debe tener el sistema y se establecen unos requisitos que entren en los plazos de la iteración, se diseña la solución y esta es implementada. Después se comprueba que la solución funcione correctamente y por último que la solución se adecue a lo que se esperaba según el proyecto (Evaluación). Es en esta etapa cuando se analizan las experiencias que han surgido durante desarrollo y pueden influir en los requisitos de las futuras

iteraciones. Si en esta etapa ha surgido algún inconveniente técnico se podrían descartar los requisitos derivados de él para evitar sobrecostes. Si por el contrario han surgido nuevas ideas que pueden resultar útiles, se propondrán para desarrollarse en iteraciones posteriores.

Figura 2: Fases del desarrollo iterativo. Recuperado de http://aprendiendocositasdelsoftware.blogspot.com/p/desarrollo-iterativo-y-creciente.html

En contraposición con otras metodologías, como la tradicional de desarrollo en cascada, los requisitos del desarrollo iterativo incremental pueden cambiar a lo largo del proceso, mientras que los requisitos del desarrollo en cascada permanecen inamovibles una vez se han establecido.

Cuando surge algún problema en alguna de las fases, se intenta mantener el desarrollo hasta el final de la iteración, pero se debe tener en cuenta para resolverse en iteraciones posteriores. De esta manera no se retrasa el avance del proyecto pero tampoco se siguen manteniendo diseños incorrectos o errores durante todo el proceso de desarrollo.

Esta metodología conlleva que los requisitos, que son los que dirigen el rumbo del trabajo, sean cambiantes. Durante las iteraciones del proceso de desarrollo

se podrán reemplazar funcionalidades que en un principio parecían adecuadas por otras que requieran menos tiempo y recursos y que mejoren las prestaciones del sistema. Los requisitos se van a ir definiendo e implementando gradualmente según vayan sucediendo las iteraciones.

2.3. El papel de la metodología en el proyecto

El principal motivo por el cuál se ha decidido el uso de esta metodología es la falta de experiencia en las tecnologías involucradas en el desarrollo, lo cuál conlleva el riesgo de que se haya hecho un análisis impreciso de las horas de trabajo para la realización de unos requisitos determinados. Mediante esta metodología se podrá finalizar el trabajo cuando se haya alcanzado el límite de tiempo y aún así tener un resultado completamente funcional. Además, de esta manera se puede invertir mejor el tiempo en añadir funcionalidades más útiles a las contempladas en un principio gracias a la flexibilidad que aporta poder realizar modificaciones de los requisitos en cada iteración. Además, debido a que el desarrollo del trabajo está realizado por una sola persona, es más idónea por permitir refinar la funcionalidad según se avanza en el desarrollo sin ningún problema de contraposición de ideas.
Como punto negativo de haber elegido esta metodología, debido al desconocimiento del tiempo de cumplimiento de los requisitos, la duración de las iteraciones es difícil de estimar.

Aunque esta metodología permite realizar cambios en los requisitos del sistema según se desarrolla, lo que no debe cambiar es la idea principal con la que se ha concebido el trabajo. Esto es, que el despertador reciba información del teléfono que pueda ser de utilidad para un usuario final, así como permitir el uso del despertador de forma autónoma como un reloj de mesita de noche.

3

Hardware Utilizado

3.1. Justificaciones del *hardware* elegido

Se ha utilizado una placa similar a *Arduino* UNO comprada a través de Internet. El *Arduino* UNO es una placa muy conocida que integra un microcontrolador con fines de creación de prototipos electrónicos que equipa un procesador ATmega328p y un controlador USB para ser conectarlo a un ordenador. El auge y popularidad de estas placas de desarrollo se sitúa en un contexto en el que la filosofía "*Hágalo usted mismo"*, o *DIY [4]* se extiende por Internet entre usuarios interesados por poder reparar y fabricar sus propios aparatos. Siguiendo esta filosofía, las placas *Arduino* pretenden simplificar la tarea de programación y configuración de placas programables a usuarios que con pocos o nulos conocimientos de electrónica y de informática quieran aprender a utilizarlos. Para ello también se provee de un entorno de desarrollo propio enfocado en la facilidad de uso así como de mucha documentación en la página web oficial de *Arduino* [5].

Los diseños de las placas de *Arduino* son *hardware* libre distribuido bajo la licencia *Creative Commons Attribution Share-Alike 2.5*. Cualquiera puede fabricar y distribuir un *hardware* similar con la condición de que no se use el nombre de *Arduino* para comercializarse, ya que este nombre es una marca registrada [6].

3.2. Limitaciones del *hardware*

Arduino UNO y otras placas similares tienen una memoria limitada: tanto la memoria flash donde se almacena el programa como la memoria utilizada para variables y datos en tiempo de ejecución. Es por ello que hay que tener en cuenta buenas prácticas a la hora de programar, así como realizar optimizaciones para aprovechar al máximo el espacio disponible. Además la velocidad de procesamiento es muy baja en comparación con otros dispositivos modernos. Por ejemplo, no existe aceleración por *hardware* para operaciones aritméticas en punto flotante.

El número de pines de entrada y salida disponibles y la intensidad de corriente máxima soportada por la placa también serán inconvenientes que deberán tenerse en cuenta.

3.3. Componentes del circuito

Para el diseño del circuito del despertador, se ha utilizado la herramienta *Fritzing*[1]. Esta aplicación permite diseñar circuitos electrónicos visuales que permiten la construcción a personas con pocos conocimientos de electrónica.

Los principales componentes de los que se compone el despertador son:

- **Módulo Bluetooth MLT-BT05 4.0**

  Es imitación de otros módulos similares como el *hm-10*, pero con una implementación *software* diferente. Es por ello que ciertos comandos que reconoce pueden ser diferentes. Es fácil y económico de obtener a través de Internet.

  Este módulo se comunica con *Arduino* mediante el puerto serie con los conocidos como 'comandos AT'.

- **Pantalla de 5 módulos con matrices de led 8x8**

  Cada módulo contiene microcontrolador *MAX7219* en una placa que tiene conectada una matriz de leds. Este microcontrolador es frecuentemente utilizado para controlar simultáneamente cuatro *displays* de 7

1 Esta herramienta será explicada en el capítulo siguiente.

segmentos mediante la técnica de multiplexación. Permite encender de forma individual cada fila en la matriz o uno de los cuatro dígitos de 7 segmentos y también regular el brillo mediante *PWM*[1]. Otra característica de este microcontrolador es que se puede conectar en cascada a más módulos. Para obtener la pantalla de 5 módulos se han obtenido 4 módulos unidos de fábrica a los que se ha añadido una matriz extra soldando las conexiones de la placa de igual manera que el resto que vienen ya unidos [7]. Estos componentes se conectan con el *Arduino* mediante el puerto SPI y le permite ahorrar tiempo de procesamiento y conexiones respecto a manejar las matrices directamente.

- **RTC DS3231 AT24C32 CII**

  Es un reloj de tiempo real también muy económico y que dispone de una memoria flash de 32 *KB* para almacenar registros, así como un sensor de temperatura que puede ser consultado pero que se usa internamente para que la medición del tiempo sea más precisa. Utiliza una pila recargable de botón para mantener la hora cuando el despertador está desconectado.

- **Sensor de humedad y temperatura DHT11**

  Este sensor no tiene una gran precisión (un error de dos grados a temperaturas normales), pero es barato y suficiente para los objetivos de este trabajo. Se podría reemplazar en un futuro por un módulo más preciso (DHT22) sin necesidad de modificar el software.

  Este módulo se conecta por medio de un interfaz propio a través de un solo pin.

1 Técnica que permite simular la regulación de voltaje en sistemas digitales mediante el uso de ondas cuadradas. Sirve para regular el voltaje de media, pero no el instantáneo, que solo podrá ser el nivel alto o bajo.

- **Led *RGB***

  Componente compuesto internamente por tres led de colores rojo, verde y azul. Un led es un componente electrónico semiconductor que deja pasar la corriente en un determinado sentido y que cuando lo hace emite luz. El led *RGB* será utilizado para mostrar notificaciones y otros avisos mediante una luz de color. Mediante la variación de la intensidad de estos tres colores colores podremos generar toda la gama de colores visibles para el ser humano [8].

- **Zumbador pasivo**

  Componente electrónico simple formado por un material piezoeléctrico, el cuál se deforma cuando pasa corriente a través de él. Dejando pasar e interrumpiendo el paso de corriente a través de él a una determinada frecuencia conseguimos mover el aire a la misma frecuencia, lo que se conoce como sonido.

  Los zumbadores activos, en cambio, disponen de un elemento que produce impulsos a una determinada frecuencia. Por tanto solo es necesario que pase corriente ininterrumpida a través de él, pero como inconveniente solo producen un sonido.

  Un zumbador tiene peor calidad de sonido que un altavoz, pero tiene la ventaja de un consumo energético mucho más reducido.

  Este componente es utilizado para avisar de notificaciones, una retroalimentación del uso de los botones y otros eventos.

- **Cinco pulsadores**

  Utilizados para controlar el despertador. Uno de ellos contiene el led *RGB* pensado para realizar acciones sobre las notificaciones y otras acciones sobre el teléfono.

3.4. Diseño del circuito

Durante el diseño del circuito se ha pretendido que fuese lo más sencillo posible para simplificar el proceso de fabricación de prototipos. Una solución que ha permitido ahorrar muchas resistencias en el circuito es el uso de las resistencias *pull up* incluidas dentro de la propia placa *Arduino*. Esto invierte la lógica de las pulsaciones (una tensión baja en el pin de entrada indica una pulsación del botón y una tensión alta indica que el botón no está pulsado).

El puerto serie del módulo de *Bluetooth LE* trabaja a 3.3V. Sin embargo las entradas y salidas de *Arduino* son de 5V. Que la entrada hacia el *Arduino* sea de 3.3V no es problema, puesto que aún al ser más baja se detecta correctamente. Sin embargo una salida de 5V hacia el módulo podría quemarlo. Es por ello que se ha tenido que incluir un divisor de voltaje en el circuito.

La explicación más detallada de los circuitos utilizados así como un diagrama con las conexiones de los componentes se encuentra disponible en el anexo A.

3.5. Fabricación del prototipo

Para el montaje del despertador se ha utilizado una caja de plástico para montajes electrónicos a la cuál se ha fijado internamente la placa *Arduino* junto con una pequeña placa de prototipado y el resto de componentes (Figura 4). Sobre la placa *Arduino* se ha añadido además una placa conocida como *Protoshield,* que permite desacoplar la placa sin tener que modificar el circuito.

Dentro de la caja también se encuentra una pequeña placa de prototipado donde está montado el circuito.

Como se aprecia en la Figura 3, en la tapa superior se encuentran cuatro pequeños botones que permiten interactuar con el despertador. También se ha colocado un botón luminoso más grande cuya función principal es la de mostrar notificaciones y descartarlas. En la parte trasera hay una ranura para insertar un cable *USB* tipo B para la alimentación del dispositivo.

3.6. Coste de fabricación de un prototipo

El cálculo del coste que supone la fabricación de un prototipo viene detallado en el anexo A. Los precios utilizados para el cálculo son los que han costado en el momento de su compra[1], a excepción de algunos que se han obtenido mediante un kit con otros componentes que no se han utilizado. Para estos últimos componentes se muestran sus precios por separado. Tampoco se han tenido en cuenta elementos embellecedores o los costes de las herramientas empleadas.

Según estos criterios, el coste de fabricación de un prototipo supondría unos 42 €. Hay que tener en cuenta que el precio por utilizar componentes de mayor calidad y las diferentes tiendas y condiciones de envío podría aumentar el precio.

Figura 3: Vista frontal del despertador.

1 Los precios tienen el I.V.A. incluido así como los gastos de envío.

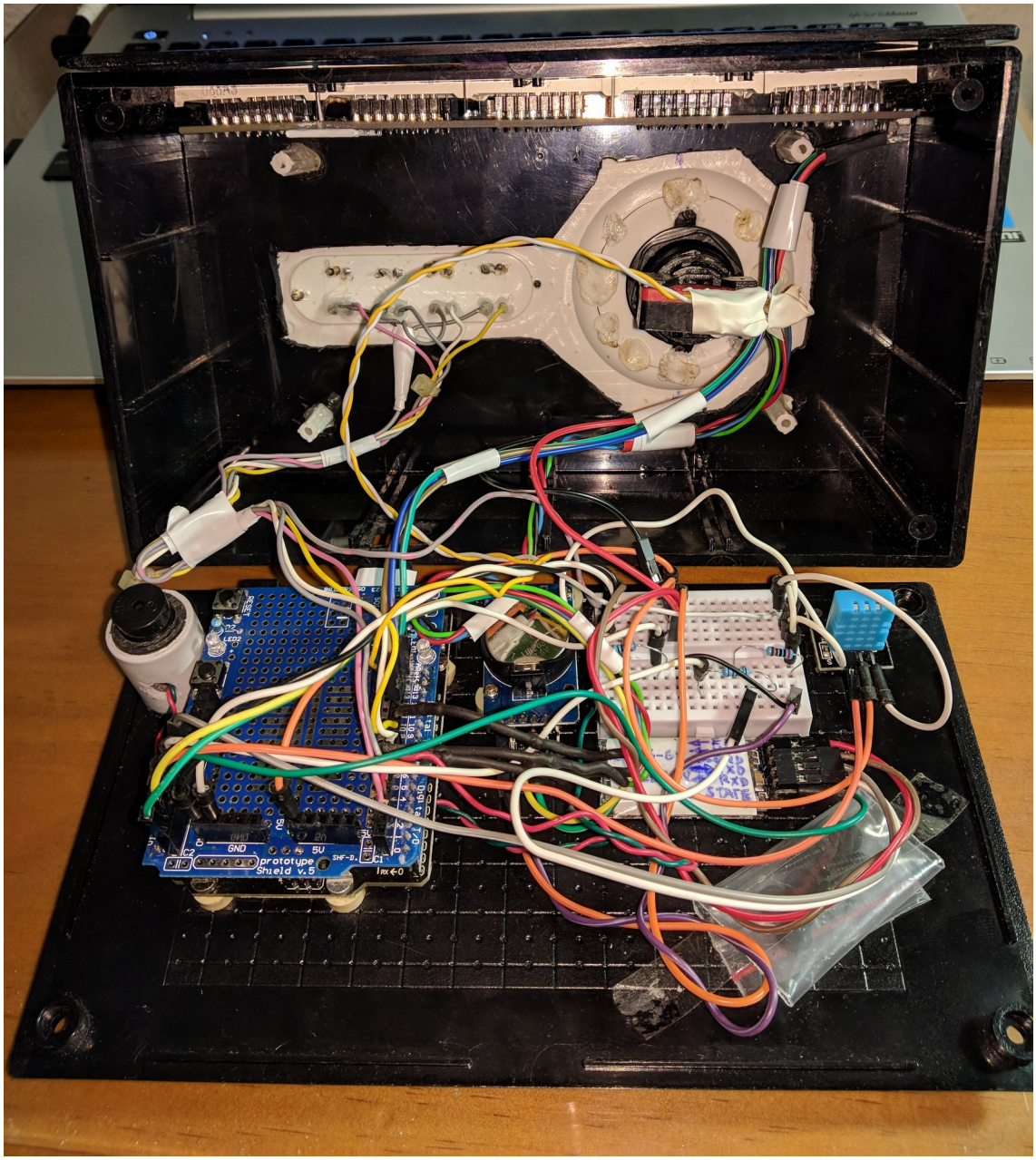

Figura 4: Prototipo del despertador con el circuito montado y abierto.

4

Tecnologías utilizadas

4.1. *Fritzing*

Es una aplicación de ordenador que permite diseñar circuitos electrónicos con elementos programables. La complejidad de los circuitos puede ir desde prototipos que se fabrican manualmente a placas impresas que pueden fabricarse de forma industrial. El proceso se realiza de forma sencilla y visual. Su estilo es acorde con la filosofía de *Arduino* de acercar la fabricación y programación de aparatos electrónicos de forma sencilla a estudiantes o aficionados. Es una herramienta de código abierto bajo la licencia *GPLv3* [9].

4.2. Draw.io

Plataforma *online* que funciona sobre tecnologías abiertas y que sirve para realizar diagramas de diversa índole. Incluye elementos que facilitan la creación de diagramas *UML*. Todo se realiza en una aplicación web a través del navegador y ofrece soporte para *Google Drive*.

4.2. C / C++

Lenguajes de programación muy populares y utilizados en *Arduino* para la programación de las placas. La implementación utilizada por *Arduino* ha sido

modificada para simplificar el desarrollo (por ejemplo mediante el uso de archivos .”ino” [10] que encapsulan el funcionamiento o la definición de funciones globales). No obstante también se encuentran limitaciones en el código C++, faltando algunas implementaciones de métodos estándar o estando limitados por temas de optimización. Por ejemplo no están disponibles los vectores y las cadenas de texto utilizadas suelen ser las propias de C.

Se ha aprovechado la orientación de objetos de C++ para diseñar el software del despertador *Arduino*. Aunque en un principio se había considerado el uso exclusivo del lenguaje C por ser utilizado en otros proyectos en la Universidad, al final se ha aprovechado la potencia de la orientación a objetos para simplificar el proceso de desarrollo.

4.3. *Java*

Es un lenguaje de programación y entorno de ejecución que permite escribir programas que funcionan en diferentes plataformas que tengan implementada la máquina virtual de *Java* (*JVM*). Fue originalmente desarrollado por *Sun Microsystems* y actualmente pertenece a *Oracle* [11]. Todo el código base usado para máquina virtual de Java se encuentra ahora disponible en la implementación *OpenJDK* bajo la licencia *GPLv2* .

Esta tecnología permite crear programas que pueden funcionar en la mayoría de sistemas operativos de ordenador y además es la base del sistema operativo *Android*. También es utilizado en todos los entornos de desarrollo mencionados en este capítulo.

4.4. *Arduino*

Es una compañía de *hardware* y *software* libres encargada de diseñar y fabricar, entre otras cosas, las placas programables del mismo nombre. Los diseños electrónicos de las placas se distribuyen bajo la licencia *Creative Commons Attribution Share-Alike 2.5,* lo que permite a cualquiera fabricar y comercializar sus propias placas compatibles pero sin el uso del nombre *Arduino* [6].

Concretamente la placa utilizada en el trabajo es similar al *Arduino UNO*; utiliza un procesador *ATmega328P U* el cuál se basa en la arquitectura *RISC* para ejecutar sus instrucciones. Existen en Internet infinidad de bibliotecas de software que permiten realizar muchas funciones con esta placa con poco esfuerzo.

4.5. *Arduino IDE*

Es el entorno de desarrollo oficial de *Arduino*. Se encuentra desarrollado en Java y está enfocado principalmente a usuarios aficionados o estudiantes. El programa permite desarrollar programas utilizando archivos de extensión ".ino" o simples archivos de código fuente de C o C++. Desde el propio programa se pueden compilar y grabar los programas en una placa *Arduino* conectada a un ordenador o de forma inalámbrica en los dispositivos soportados.

4.6. Biblioteca *Parola*

Biblioteca para *Arduino* utilizada para manejar a alto nivel matrices de led de 8x8 píxeles controladas por chips de la familia *MAX72XX* (por ejemplo el *MAX7219*). Ha sido desarrollada por Marco Colli y se encuentra disponible públicamente en *GitHub* bajo la licencia *LGPL*. Esta biblioteca utiliza a su vez la biblioteca *MD_MAX72xx* desarrollada también por él mismo y que permite controlar los chips de estas matrices a más bajo nivel. Estos chips suelen ser también utilizados para controlar cuatro *displays* de 7 segmentos y su funcionamiento es similar. Reciben a través del puerto *SPI* la información a mostrar y se encargan por medio de una multiplexación de encender cada dígito o fila durante un breve periodo de tiempo. La técnica de multiplexación consiste en mostrar diferentes secciones de una pantalla en periodos de tiempo muy breves para ahorrar conexiones. Este método es eficaz ya que el ojo humano tiene la sensación de que todos los componentes se encuentran encendidos al mismo tiempo gracias al fenómeno de la persistencia de la visión [12].

La biblioteca *Parola* permite disponer de muchos efectos de texto, mostrar iconos y otras animaciones diversas. El inconveniente es el gran uso de memo-

ria que conlleva. En el trabajo se ha optimizado el uso de memoria de la biblioteca desactivando efectos que no se pretendían utilizar en tiempo de compilación [13].

4.7. *Eclipse IDE*

Entorno de desarrollo escrito en Java que simplifica el proceso de desarrollo de muchos lenguajes de programación y tecnologías. Fue desarrollado originalmente por IBM, pero ahora es mantenido por la Fundación Eclipse [14]. Algunos lenguajes de programación que se pueden desarrollar con él son Java, C, C++, Lua o Python.

Eclipse es más que un solo entorno de desarrollo y sirve de núcleo para el desarrollo de otros programas, como por ejemplo *Apache Directory Studio, u*na herramienta para gestionar servidores que utilizan el protocolo LDAP.

4.8. *Plugin Sloeber*

Complemento para *Eclipse IDE* que simplifica el desarrollo de s*oftware* para *Arduino* en este entorno de desarrollo. Es tan sencillo como descargarlo desde la tienda de aplicaciones de Eclipse, crear un nuevo proyecto de forma similar al *IDE* de *Arduino* y pulsar un botón para subir a la placa. El complemento está desarrollado por Jantje y es de código abierto [15]. La alternativa principal a este complemento es el *plugin AVR-Eclipse*, el cuál es más complejo de utilizar.

4.9. Android

Es un sistema operativo muy popular para dispositivos móviles desarrollado por Google. Su primera versión se lanzó al mercado el 23 de septiembre de 2008 [16]. Las aplicaciones desarrolladas para este sistema son escritas principalmente en Java dentro de un *framework*[1] aunque también se permiten trozos de

1 Un *framework* o marco de trabajo es una estructura predefinida de un software que sirve como base para desarrollar dentro de él.

código escritos en C o C++. Las aplicaciones actualmente se ejecutan por medio del *runtime ART*, siendo Dalvik el *runtime* de versiones anteriores.

Una característica muy interesante desde el punto de vista del desarrollo son las herramientas *ADB [17].* Estas son herramientas de línea de comandos para dispositivos *Android* que permiten manipular el sistema con fines de depuración.

Las aplicaciones que se pueden desarrollar en *Android* están compuestas por lo que se conoce como actividades, que se corresponden con pantallas con las que el usuario puede interactuar [18]. Estas actividades están gobernadas por un ciclo de vida al cuál nosotros podemos establecer las acciones que se realizarán en determinados momentos del ciclo de vida por medio de las herramientas que nos ofrece el *framework*. En la Figura 5 podemos ver un diagrama del funcionamiento de este ciclo de vida de una actividad.

4.10. *Android Room*

Es una herramienta para la persistencia de información en una aplicación *Android*. Permite abstraer el uso de la base de datos *SQLite*[1] que viene integrada en *Android*. Se puede considerar como una técnica de Mapeo objeto-relacional [19] u *ORM* por sus siglas en inglés. Mediante esta herramienta se pueden hacer operaciones en una base de datos mediante la manipulación de objetos, evitando así redundancias en las transformaciones de contenidos a la hora de persistir la información de un programa.

1 Base de datos relacional que utiliza un solo fichero para almacenar la información.

4.11. *Android Studio*

Es la herramienta oficial para desarrollar aplicaciones de *Android*. Está basado en el *IDE intelliJ IDEA* de la empresa JetBrains. Reemplazó a *Eclipse IDE* como herramienta oficial para el desarrollo de aplicaciones en esta plataforma [20]. Mediante asistentes se puede empezar a crear una aplicación *Android* de forma inmediata tras su instalación. También dispone de muchas herramientas de depuración utilizando el ya mencionado *ADB* y un emulador integrado.

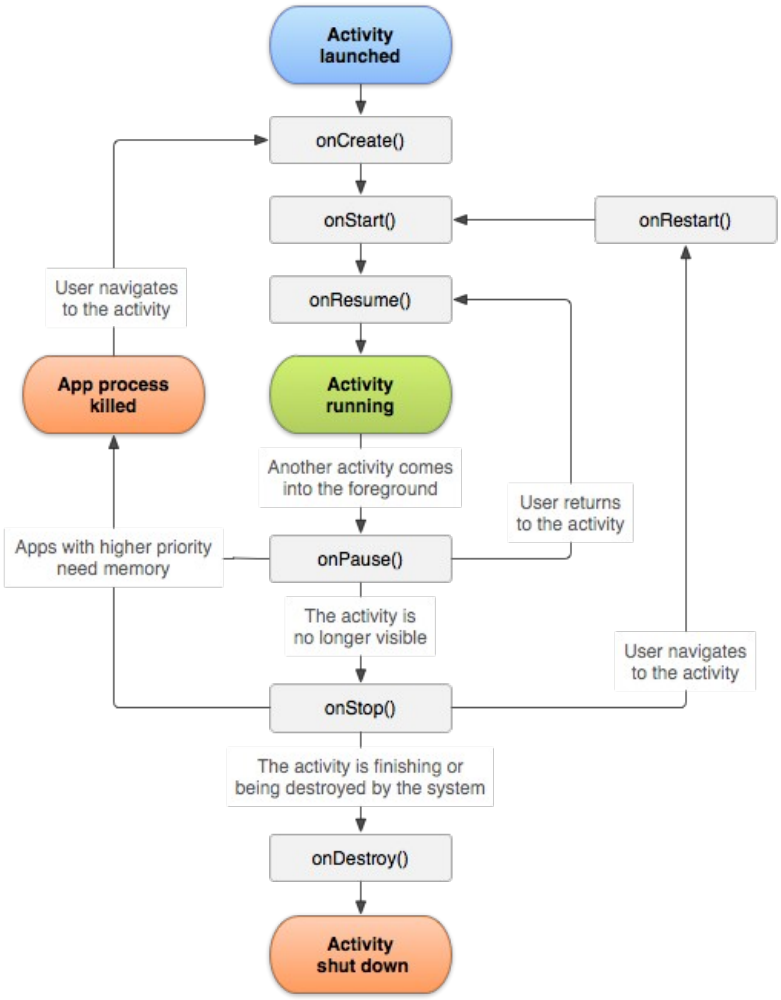

Figura 5: Ciclo de vida de una actividad en Android. Recuperado de https://developer.Android.com/guide/components/activities.html?hl=ES

4.12. Bluetooth LE

Bluetooth LE o Bluetooth Low Energy, es una tecnología de comunicación de dispositivos en una r*ed de área personal* (PAN por sus siglas en inglés) que se mueve en el mismo rango de frecuencias que su versión anterior[1] (*Bluetooth* clásico). *Pese a compartir la marca Bluetooth,* es una tecnología totalmente diferente al *Bluetooth* clásico. *Bluetooth LE* se centra principalmente en el ahorro de energía y su implantación en dispositivos de recursos reducidos [21] y para ello ha adoptado un protocolo completamente diferente.

Antiguas especificaciones *Bluetooth* permitían dispositivos de tipo emulador de puerto serie. En la especificación *Bluetooth LE,* sin embargo, esta categoría ya no existe y módulos como el utilizado en este trabajo implementan esta funcionalidad de otra forma no estándar.

La especificación Bluetooth LE es un remplazo completo del protocolo anterior en el cuál se utiliza un esquema de conexión completamente diferente. El objetivo principal de esta renovación es principalmente el ahorro de energía, enviando para ello la menor cantidad de información posible.

Los términos clave [22] necesarios para conocer el funcionamiento de las conexiones Bluetooth LE son:

- **Protocolo GATT:** Provee el descubrimiento de dispositivos, sus tipos y cuáles deben de ser sus identificadores. Define la existencia de Características y Servicios.
- **Característica:** La parte mínima de la que se compone una comunicación Bluetooth LE. Una característica es capaz de comunicar un único valor, el cuál no debe superar los 20-22 bits. Una posible característica serían las pulsaciones de una persona comunicadas por un reloj inteligente.
- **Servicio:** Agrupación de características con alguna finalidad común.
- **Atributo:** Son unidades de información definidas en el protocolo GATT que pueden designar tanto características como servicios.
- **Dispositivo Maestro/Central:** Se encarga de buscar dispositivos esclavos y mandar peticiones de conexión hacia ellos para establecer cone-

1 Entre 2402 y 2480 MHz.

xiones. Este dispositivo puede conectarse a más de un dispositivo. En nuestro caso concreto este rol lo toma el teléfono móvil.

- **Dispositivo Esclavo/Periférico:** Recibe y acepta peticiones de conexión de un Maestro. Solo pueden estar conectados a un dispositivo maestro. Este rol será el que deba tomar el despertador inteligente.
- **Cliente:** Es el dispositivo que se comunica con un servidor para obtener o publicar información. Aunque suela ser propio de los dispositivos con el rol de maestro, también podría serlo un dispositivo de rol esclavo. Este será el papel del dispositivo móvil conectado al despertador.
- **Servidor:** Es el dispositivo que responde a las comunicaciones bien devolviendo algún valor requerido o almacenándolo.
- ***Write* y *Read*:** Acciones realizadas por el cliente para comunicarse con un servidor para enviar o solicitar información respectivamente. El módulo *Bluetooth* utilizado en este trabajo emula una conexión de puerto serie y solo permite recibir la acción *Write*.
- ***Notify* e *Indicate*:** Ambas son acciones utilizadas para comunicarse con un cliente que previamente ha debido suscribirse a la característica. La diferencia radica en que una acción *Notify* no requiere una confirmación por parte del cliente, mientras que el uso de *Indicate* sí.

4.13. Bluetooth LE en Android

La documentación en Internet acerca del uso del *Bluetooth LE* en *Android* es algo escasa en comparación con la del *Bluetooth* clásico. Es necesario tener claros los conceptos propios de esta tecnología para poder comprender las guías disponibles. La disponibilidad en Internet del código fuente de bastantes proyectos que hacen uso de *Bluetooth LE* facilita la comprensión del funcionamiento de esta tecnología en *Android*.

En términos generales para establecer una conexión básica en *Android* con el rol maestro y cliente hacia un dispositivo *Bluetooth LE*, primero necesitamos obtener la instancia *BluetoothAdapter,* que representa el componente físico del teléfono. Este se obtiene por medio de la instancia *BluetoothManager* del

sistema *Android* que sigue el patrón s*ingleton*[1]. Llamando al método *start-LeScan(callback)* de *BluetoothAdapter* obtenemos los dispositivos encontrados, que irán siendo notificados al objeto *callback*[2] que hemos debido de añadir como parámetro.

Cuando encontremos un dispositivo con el que queramos establecer conexión, llamamos al método *connectGatt(context, autoconnect, callback)* del mismo introduciendo como parámetro *callback*, un objeto que recibirá los diferentes eventos que el Servidor o la conexión pudieran desencadenar para ser tratados en el momento de su aparición [21] .

1 Patrón de diseño en *ingeniería de software* que sirve para que solo pueda existir un único objeto de una clase y este sea obtenido desde cualquier parte del código.

2 Método u objeto que se se pasa como referencia a otra parte del código para que sea llamado cuando se ha recibido o completado alguna acción. Normalmente se suele utilizar en sistemas asíncronos [23].

5

Proceso de desarrollo

5.1. Iteración Primera

5.1.1. Planteamiento y requisitos previstos

Los objetivos marcados para esta iteración son crear un sistema modular de aplicaciones, detectar las pulsaciones de los botones, mostrar y editar la hora y mostrar mensajes procedentes de una aplicación móvil mediante la conexión *Bluetooth LE*. En la Figura 6 podemos ver un diagrama de casos de uso de un usuario sobre sistema .

5.1.2. Diseño y desarrollo en *Arduino*

Se ha comenzado desarrollando el software de la placa *Arduino* para comprobar el correcto funcionamiento de los componentes y del circuito. Lo primero que se hizo fue un código de prueba para mostrar los colores del led, escribir un texto de ejemplo en la pantalla y reproducir un sonido con el zumbador. Para escribir el texto en la pantalla se ha utilizado la biblioteca Parola. Después se ha desarrollado otro código para mostrar la hora, la temperatura y la humedad.

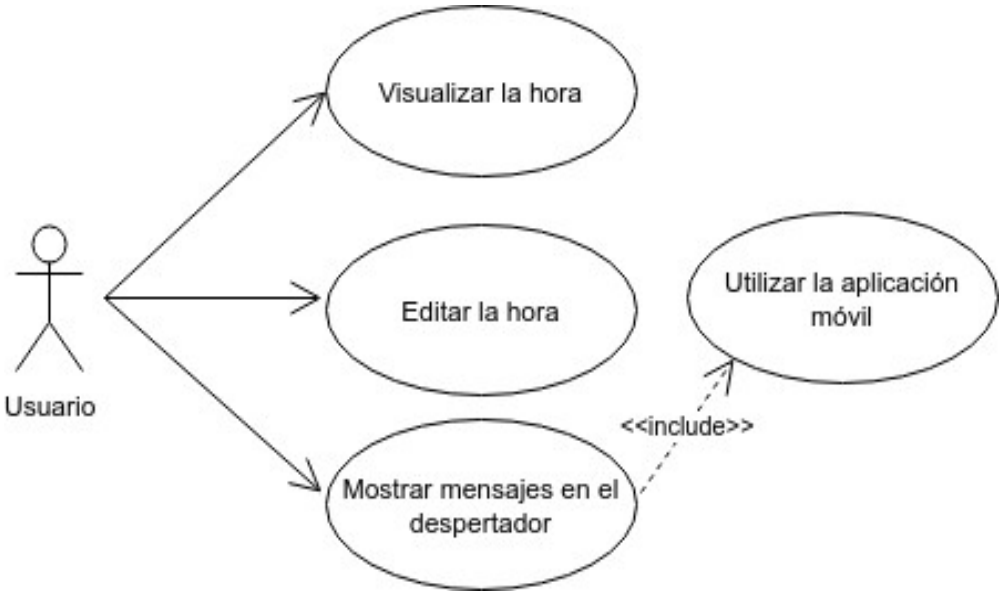

Figura 6: Diagrama de casos de uso de la iteración primera.

Tras los códigos de prueba del *hardware* fue necesaria una investigación más a fondo de la biblioteca Parola y de cómo se podría aprovechar para mostrar el reloj, las notificaciones y otros menús en pantalla. Tras un tiempo en el que no se conseguía mostrar texto correctamente por la pantalla, al final esta empezó a funcionar cambiando el tipo de *hardware* predeterminado que utiliza la biblioteca por el tipo *FC16_HW*.

En un principio la idea era desarrollar el programa de *Arduino* utilizando solo código de C, pero al final se optó por la programación orientada a objetos que permite C++ para estructurar todo el sistema. Sin embargo debido a los escasos conocimientos de este lenguaje, se tuvieron que ir aprendiendo sobre la marcha muchos conceptos relativos a cómo funciona su orientación a objetos, el uso de la pila y de la memoria dinámica para almacenar objetos.
Debido a las limitaciones de la placa *Arduino*, el uso de la memoria es un elemento crítico y por tanto es necesaria la escritura de código optimizado que permita aprovechar al máximo sus capacidades. El contenido de las variables de las funciones que se llaman de forma anidada aumenta el tamaño de la pila, que consume memoria. La creación dinámica de objetos y otras estructuras en

la memoria sin utilizar el anidamiento en la pila provoca también un aumento de memoria. La diferencia es que cuando se vacían los contenidos de la pila, lo que se realiza de forma ordenada, esta disminuye sin dejar espacios. Sin embargo cuando se eliminan elementos creados dinámicamente sin un orden establecido se producen huecos en la memoria que pueden conllevar eventualmente al límite de memoria del *Arduino* y por tanto a un fallo en el programa. Para minimizar al máximo este inconveniente, la pila y la memoria dinámica parten de valores opuestos de la memoria. De esta forma se puede compensar un uso elevado de objetos en memoria dinámica con un anidamiento bajo de funciones o viceversa. Esto se puede observar en la Figura 7.

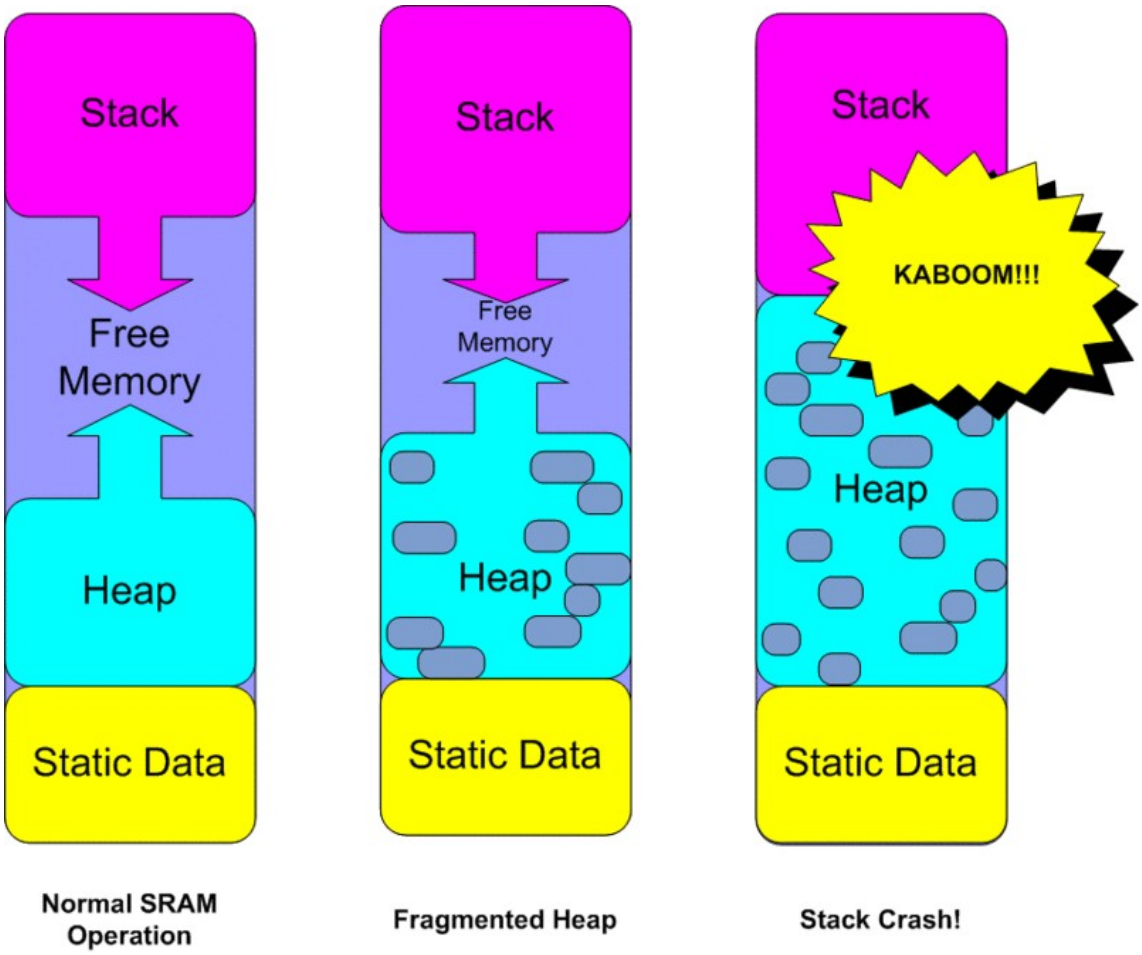

Figura 7: Situación de sobrecarga de memoria de datos variables de programas en Arduino. Recuperado de "https://learn.adafruit.com/memories-of-an-Arduino/optimizing-sram"

Hay que tener en cuenta que C y C++ son compatibles entre sí en el mismo programa. Durante la compilación se pueden enlazar los binarios objetos resultantes siempre que se separen en ficheros diferentes con la extensión “.c” o “.cpp” dependiendo de si están en código C o C++ respectivamente. El compilador lo reconoce y compila todo adecuadamente.
Para organizar el código se creó un archivo separado con constantes con todas las configuraciones importantes de pines llamado *pinout.h*. También se creó otro archivo llamado *hardware.h* con importaciones de las bibliotecas necesarias para controlar el *hardware* y con clases que controlan los dispositivos y otras constantes globales que les afectan.

Con la ayuda de la orientación a objetos, se empezó a diseñar una forma de crear diferentes programas que se ejecutasen de forma independiente en el despertador (Figura 8). A estos programas se le dieron el nombre de actividades. Una actividad se constituye en base a un objeto de una clase que hereda de la clase Actividad. Las actividades poseen un método ejecutar, que es llamado continuamente por el bucle principal del *Arduino*. Esta estructura de control se conoce en sistemas de tiempo real como ejecutivo cíclico. Asimismo las actividades también tienen un método set_instante() que permite actualizar un contador interno creciente que sirve para que las actividades midan el tiempo. La obtención de este valor por un argumento en lugar de ser obtenido internamente mediante la función de *Arduino millis()* o *micros()* permite realizar pruebas utilizando valores personalizados (por ejemplo comprobar qué sucede ante un desbordamiento del contador) o incluso congelar las actividades de forma que no pase el tiempo para estas.
Se ha diseñado la actividad *Reloj*, que será la primera que se cargue cuando arranque el sistema.

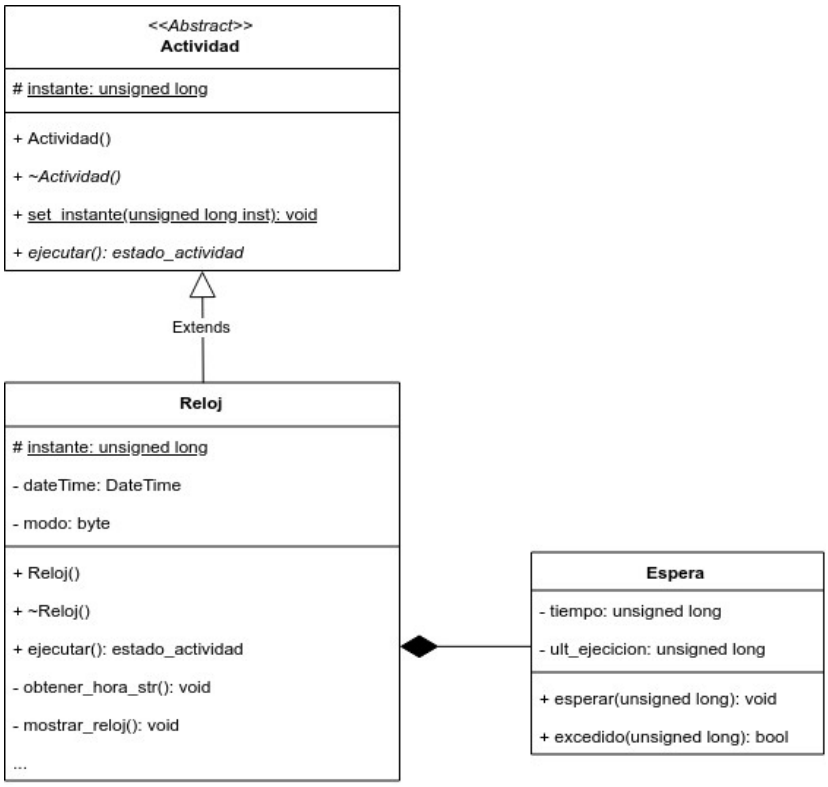

Figura 8: Diagrama de clases de la actividad Reloj *en el despertador.*

Debido a que una vez que se empieza una tarea, esta continúa un tiempo hasta que la misma lo considera conveniente, pudiendo haber varias tareas ejecutándose una detrás de otra, también se podría considerar como un sistema de multitarea cooperativa. También se podía haber optado por utilizar interrupciones del sistema, pero dado que las capacidades del *Arduino* son limitadas y las aplicaciones son desarrolladas en conjunto con el sistema, se ha decidido no aplicar este enfoque. De esta manera no es necesario un mecanismo de cambio de contexto y se ahorra espacio ya que cada vez que una aplicación empieza una ejecución, reserva un espacio en la pila para después liberarlo. Al no retener la posición del contador de programa ni las variables de la pila en cada ejecución de una actividad, es necesario que las actividades se desarrollen siguiendo un funcionamiento de máquinas de estado y almacenen dentro del objeto los datos necesarios para continuar su ejecución.

Se empezó realizando el desarrollo en el *IDE* oficial de *Arduino*, pero tras el aumento de la complejidad del desarrollo, se investigó la existencia de otros entornos de desarrollo que simplificasen el proceso y se adaptaran mejor a proyectos avanzados. En el *IDE* de *Arduino* no se pueden elegir las opciones de compilación y no se dispone de autocompletado ni verificación de código durante su escritura. Todos los archivos que formen parte del proyecto permanecen abiertos como pestañas y llenan toda la barra sin poder desplazarla. Es por todo ello que se decidió utilizar otra alternativa: El *plugin Sloeber* para *Eclipse IDE.* Este *plugin* tiene todas las funcionalidades del *IDE* de *Arduino* pero permite editar los ficheros desde Eclipse, lo cuál simplifica mucho la tarea de desarrollo.

5.1.3. Diseño y desarrollo en *Android*

Se empezó creando una aplicación desde cero desde el asistente de *Android Studio*. Tras haber creado la aplicación se modificó el archivo *manifest* de la aplicación. Este archivo se empaqueta dentro de la aplicación e informa al sistema operativo de los permisos que requiere o de las funcionalidades que implementa o necesita.

Los permisos que se tuvieron que incorporar a la aplicación aparecen descritos en la Guía oficial de desarrolladores de *Android*, en la sección de visión general de Bluetooth LE y son los siguientes:

```
<uses-permission Android:name="Android.permission.BLUETOOTH"/>
<uses-permission
          Android:name="Android.permission.BLUETOOTH_ADMIN"/>
```

Además del siguiente para *manifest*ar que la aplicación solo puede funcionar si el dispositivo tiene conectividad Bluetooth LE:

```
<uses-feature Android:name="Android.hardware.bluetooth_le"
             Android:required="true"/>
```

A la hora de mostrar el texto en el panel del despertador recibido por Bluetooth desde el teléfono *Android* hay un límite en el tamaño del *buffer* que se comu-

nica a la biblioteca parola. Es por ello que la aplicación móvil conoce cuál es el tamaño del *buffer* y manda un máximo de información sin llegar a saturarlo. En un principio el tamaño del *buffer* en todas las actividades será de 10 bytes. Este tamaño es fijo, pero en futuras iteraciones podría cambiar de forma dinámica y negociarse entre el teléfono y el despertador.

5.1.4. Pruebas de funcionamiento

A la hora de comprobar el funcionamiento de los botones se comprobó que producían una salida errónea. Para solucionar esto se decidió resolverlo antes de la siguiente iteración implementando lo que se conoce como *debouncing*. Esto consiste en realizar una pequeña espera cada vez que se presiona o se deja de presionar un botón para evitar reaccionar ante las pequeñas fluctuaciones eléctricas fruto de la oscilación del pulsador [24]. La implementación utiliza la clase Espera para medir el paso del tiempo.

Cuando se envían mensajes cuya longitud coincide exactamente con el tamaño de *buffer* o múltiplos de él, el texto no termina de desaparecer. Este error se puede reproducir con un tamaño de *buffer* de 10 bytes y la secuencia “Hola mundo”. Este fallo será resuelto en la próxima iteración.

5.1.5. Evaluación del resultado

Solo se pueden mandar mensajes de un tamaño determinado, debido a que si el mensaje sobrepasa el buffer no cabe y por tanto no puede ser comunicado a la biblioteca que lo muestre. Este inconveniente indica la necesidad de hacer un rediseño en la próxima iteración.

Un apartado a considerar es el de las representaciones de información y codificaciones del texto. En este punto del desarrollo ya se encuentran los dos sistemas conectados y por tanto cada uno representa los datos de diferentes maneras. Un entero normal para *Android* lo compondrían 32 *bits*, sin embargo para el *Arduino* solo serían 16. La codificación de caracteres en *Parola* utiliza una tabla de 256 caracteres, la cuál por defecto es la que viene en la librería MD_MAXX72XX y es una codificación *ASCII* extendida (Los 128 primeros so

caracteres de la tabla *ASCII* y el resto de caracteres son utilizados para símbolos y caracteres latinos como la eñe o las tildes).

5.2. Iteración Segunda

5.2.1. Planteamiento y requisitos previstos

Los objetivos marcados para esta iteración son:

- Mostrar en el despertador mensajes de cualquier longitud enviados desde el teléfono, arreglando el fallo de la iteración anterior.
- Dotar a estos mensajes de una melodía definida por el usuario.
- Iluminar el led *RGB* del despertador tras una orden del teléfono a una frecuencia y color determinados y responder a los botones para apagarlo.
- Mostrar la humedad y temperatura locales del despertador.

Para cumplir con el primer objetivo de esta iteración es necesario solucionar el problema de la iteración anterior con el límite del tamaño del texto en los mensajes.

5.2.2. Diseño y desarrollo en *Arduino*

Se pretende que todos los objetivos descritos de esta iteración sean abordados mediante actividades individuales que permitan el reparto de las funcionalidades y den la sensación de multitarea en el dispositivo.

Para evitar tener que modificar el desarrollo posterior, primero se solucionó el problema del límite del *buffer* de la anterior iteración para así poder enviar un texto de cualquier longitud.

No fue una solución sencilla debido a que la biblioteca no informa del progreso del texto en la pantalla, solo se puede conocer si la animación ha terminado cuando el método *bool MD_Parola::displayAnimate(void)* devuelve *true*.

La solución en última instancia conllevaría implementar la impresión de texto en la pantalla utilizando la biblioteca MD_MAX72XX, de más bajo nivel y que utiliza parola internamente. Para intentar evitar lo que sería una solución complicada y que podría tener problemas de compatibilidad posteriores por

usar la biblioteca Parola al mismo tiempo en otras partes, se decidió investigar alguna otra forma de conseguir el objetivo con las herramientas que ofrece la biblioteca Parola.

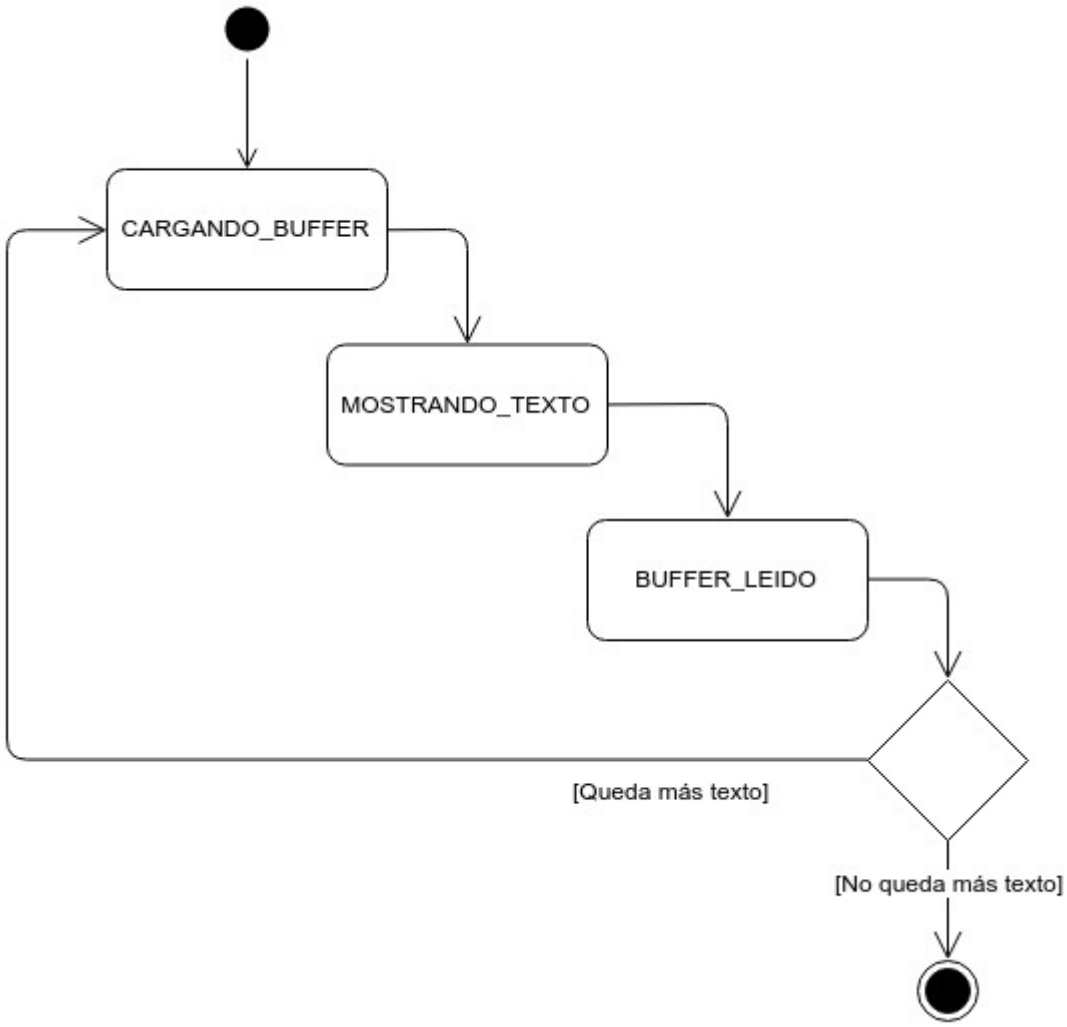

Figura 9: Diagrama de estados de la actividad Mensaje.

Tras un tiempo buscando soluciones se consiguió implementar lo que se conoce como *workaround*. Esto es una solución por una vía alternativa a la adecuada de un fallo software o una carencia por diseño [25]. Dependiendo del caso solución puede tomarse durante el desarrollo de un software o en el momento de su operación. En este caso consistió en aprovechar que se puede conocer en cualquier momento si la animación de entrada del texto ha terminado de mostrarse. Si el mensaje es más largo que lo que el *Arduino* es capaz de mostrar, entonces el móvil enviará un primer segmento de él y el

despertador desactivará el efecto de desaparición del texto. Entonces en el instante en que eso ocurra, el despertador avisa al teléfono móvil de que envíe más texto hasta volver a llenar el *buffer*. El teléfono enviará otro segmento y después el despertador volverá a llamar a la biblioteca Parola para imprimir el nuevo texto contenido en el *buffer* otra vez, realizando la misma acción anteriormente descrita. Cuando al final quede un texto más pequeño que el tamaño del *buffer* se llamará a la biblioteca, pero esta vez indicando que debe haber una animación para hacer desaparecer el texto. Todo este proceso es llevado a cabo por la actividad Mensaje, la cuál se rige por el diagrama de estados de la Figura 9.

El siguiente paso fue idear una forma de reproducir sonidos en el *Arduino*. La idea es utilizar una especie de partituras que se encuentren en la memoria y que por medio de una Actividad se vaya reproduciendo. Comienzo realizando pequeñas pruebas con el zumbador para ver la mejor forma de reproducir sonidos. Pruebo a utilizar la misma función de espera que he desarrollado para evitar el ruido en los botones o la espera de actividades y funciona correctamente. El problema es que la escala de milisegundos se queda muy corta para las frecuencias audibles. Por ejemplo una nota musical La de 440hz: El período sería de unos 2,27 ms, por lo cuál el uso de milisegundos no es adecuado. Se podrían utilizar nanosegundos para las esperas, pero igualmente podría darse la situación que tareas de otras actividades tardasen demasiado tiempo y hubiera problemas de sonido. Es por ello que me decidí por la función tone() de *Arduino*. Esta recibe el pin, una frecuencia y una duración y mediante el uso de interrupciones hardware se produce un sonido con la frecuencia deseada en cualquier momento de la ejecución del programa.
Utilizando esto último se crea la actividad abstracta “Musica”, con un método *unsigned long reproducir_nota(byte nota),* que recibe una figura musical, reproduce su sonido durante su duración y devuelve cuánto tiempo debe pasar para reproducir la siguiente. Esta actividad también contiene un método virtual *byte siguiente_nota()* el cuál es abstracto y será concretado por otras implementaciones en función de por qué medios obtengan la melodía.

La actividad abstracta Música se completa por medio de la actividad *Musica-Struct*, que recibe una melodía a través de un *struct* en el momento de ser construida. Cada vez que se ejecuta la actividad se comprueba si debe reproducirse la melodía siguiente.

La manera de escribir melodías es creando un cadena de bytes en el código. Cada byte representa una figura musical que se va reproduciendo hasta que se llega a una nota especial que es donde se termina la melodía. Las figuras musicales contienen tanto una nota musical como una duración. En la actividad se utilizan los primeros 5 bits para definir la nota musical. Para ahorrar espacio de almacenamiento solo utilizo un array con las frecuencias empezando por el Fa_3, ya que multiplicar por dos sucesivamente una frecuencia permite obtener las de octavas superiores. Para ahorrar aún más espacio, resto a las frecuencias del array un número que me permite que cada una quepa en un byte. La gama de notas posibles va desde Fa sostenido en tercera a Do en quinta. El sexto bit se utiliza para indicar si se quiere hacer una ligadura, que consiste en tocar dos figuras musicales consecutivas sin interrupción. Por defecto entre cada figura musical se reproducirá una pequeña pausa. Si este bit se encuentra activo, la pausa se suprimirá, lo cuál amplia muchas posibilidades.

Los dos últimos bits definen la duración de la nota. La duración de cada nota musical será de $\frac{b}{2^d}$, donde b es la duración de una blanca y d el valor de los dos últimos bits de la nota, tomando valores entre 0 y 3. Por tanto las duraciones posibles de una nota son blanca, negra, corchea y semicorchea. Hay una forma de obtener duraciones más largas y es juntando dos figuras por medio de una ligadura. El puntillo se puede obtener de una forma similar, ya que este indica un alargamiento de la mitad de una duración.

La melodía también contiene un tempo, esto es en lenguaje musical, el número de negras o blancas por minuto.

Para simplificar la tarea de la escritura de melodías se escribe el archivo de cabecera *notas.h*, el cuál contiene macros de precompilador con el valor de cada nota musical, las duraciones, la ligadura, el silencio y el fin de melodía.

Se han escrito unas melodías de prueba las cuáles podrán ser utilizadas en futuras iteraciones en sonidos de alarmas o recordatorios, o simplemente con fines demostrativos.

Se procede a diseñar la reproducir sonidos provenientes del teléfono en el despertador. Debido a que ya se envían mensajes utilizando el mismo canal, se debe diseñar un mecanismo que permita que diferentes segmentos de retransmisiones sean recibidos por sus actividades correspondientes así como evitar las interrupciones de flujo entre ellas.
Es por ello que se crea la actividad *Protocolo,* que recibirá el *Stream* del puerto serie de la conexión *Bluetooth* y será la encargada de identificar el destino de cada segmento (identificado por un carácter inicial) y de crear las actividades o ejecutar actividades en segundo plano creadas.
Después de ser creadas las actividades, serán éstas las encargadas de leer los segmentos y comprobar si les corresponden. Habrá actividades que se creen en segundo plano como la de reproducir sonido, las cuáles serán ejecutadas desde dentro de la actividad *Protocolo*.
Se escribe la actividad *MusicaStream,* la cuál se basa en la actividad "*Musica"*, pero irá reproduciendo una melodía según lleguen segmentos por el *Stream Bluetooth*, de una forma similar a los mensajes. Para evitar que haya interrupciones en la reproducción de música, esta actividad contiene un *buffer* circular que irá pidiendo y almacenando segmentos nuevos de la melodía conforme tenga espacio. Los mensajes dirigidos hacia esta actividad empezarán por "S" para empezar a reproducir nuevas melodías y por "s" para continuar con la reproducción de una melodía en curso.

Se ha modificado la actividad Mensaje para que se adapte al protocolo y no se produzcan conflictos con la reproducción de música. Es por tanto que la activi-

dad se ha modificado para que solo lea segmentos dirigidos a ella, los cuáles se precedidos por la letra “M” en nuevas creaciones y la “m” para continuaciones del mensaje. Sin embargo, debido a la falta de notificación de progreso de la biblioteca *Parola*, no se puede implementar un *buffer* circular debido a la falta de anticipación. No obstante la animación del texto es suficientemente fluida.
En la Figura 10 se puede ver un diagrama de secuencia de ejemplo de cómo se envía un mensaje del móvil al despertador

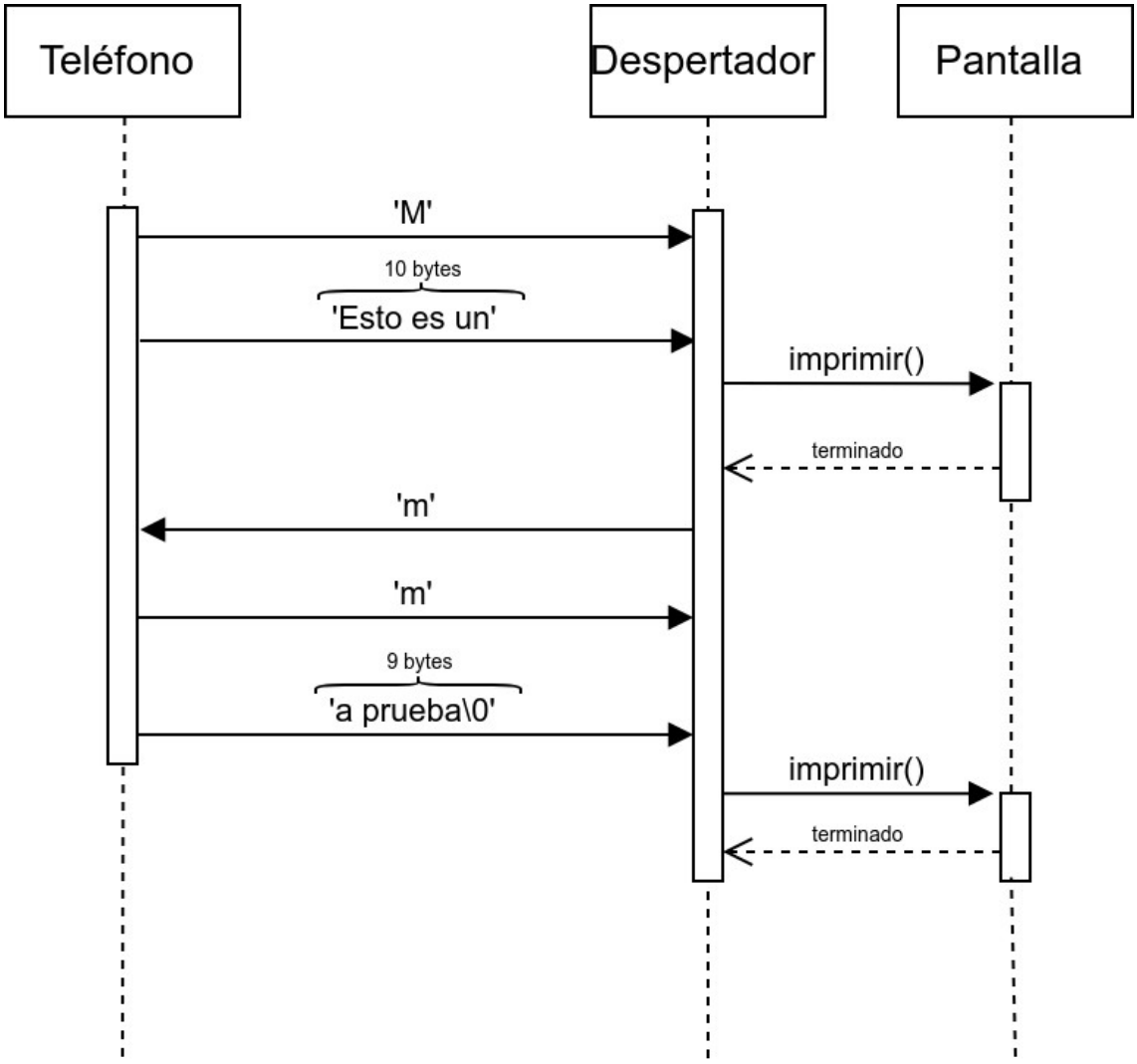

Figura 10: Diagrama de secuencia de un mensaje siguiendo el protocolo de comunicación implementado.

Se escribe la clase *LED*, de la cuál se crea una instancia en una variable global acompañando al resto de componentes hardware. Esto permite un acceso centralizado y homogéneo de las funcionalidades de este componente.

Se escribe la actividad *LedStream*, que se inicializará con un número que indica la frecuencia del parpadeo del led, así como el color a utilizar. Esta actividad hará uso de la variable de instancia de *LED* creada anteriormente. Durante su ejecución también estará a la escucha el botón grande para saber cuándo se ha pulsado. Esto en futuras iteraciones servirá para descartar las notificaciones del teléfono.

Para mostrar la humedad y temperatura se ha modificado la actividad *Reloj* añadiendo más estados que representan las diferentes pantallas que puede tomar. Mediante el uso de los botones centrales se podrá desplazar entre los modos de reloj, temperatura y humedad.

5.2.3. Diseño y desarrollo en Android

Para simplificar el envío de segmentos de diferentes tipos de datos se ha creado la clase *PacketIterator,* que implementa una interfaz de iterador, ofreciendo de forma secuencial los segmentos que deben ir transmitiéndose. De esta clase más general, a su vez se basan las clases *Message* y *Sound*. Estas clases se encuentran recogidas en el paquete communication, dentro del paquete *model*.

El funcionamiento anterior del envío de mensajes se ha reescrito completamente en la clase anteriormente comentada.

También se ha creado un *HashMap* en la clase Despertador, que almacena objetos *PacketIterator* con segmentos pendientes de enviar que serán enviados cuando el despertador avise al móvil. Las claves de este *HashMap* son los caracteres de continuación de cada uno de los tipos de segmentos, de forma que se identifique cuál es el siguiente segmento que se debe enviar hacia el despertador.

A la hora de reproducir melodías la idea es que estas sean leídas a través de una cadena de texto, lo cuál facilita que nuevas melodías puedan ser escritas tanto por usuarios dentro de la aplicación como en ficheros de texto dentro del código fuente. Para convertir el texto con la melodía al *array* de *bytes* que interpreta el despertador se han desarrollado métodos en la clase auxiliar *SoundUtils*, los cuáles son capaces de interpretar todas las notas musicales con cualquier número de sostenidos o bemoles, los cuáles se representan por # y ~ respectivamente.
Se han creado nuevos elementos de interfaz para poder escribir melodías y probar su funcionamiento.

5.2.4. Pruebas de funcionamiento

Se han tenido que hacer cambios en ambas plataformas a la hora de comunicarse debido a errores que producían que los sonidos no se enviaran o recibieran correctamente. Los segmentos se escribían en posiciones incorrectas del *buffer* debido a la jerarquía de operaciones en una expresión. Esta expresión contenía una macro del precompilador que en lugar de ser un número era un producto (Figura 11). Esto hizo que la operación módulo se realizase solo con el primer factor y luego el resultado final se multiplicase por dos. Este error se descubrió tras varias depuraciones conjuntamente entre los dos sistemas para ver si todo funcionaba correctamente.

```
//Si recibimos otro fragmento de melodía y nos cabe, lo guardamos en el buffer circular
if(stream->available() && stream->peek() == SONIDO_CONT && TAM_BUFFER_REAL-available >= TAM_BUFFER_CON ){
    Serial.println("Guardamos notas");

    stream->read(); //Leemos la s

    unsigned first_empty = (next+available)%TAM_BUFFER_REAL;
    size_t part1 = (TAM_BUFFER_REAL-first_em
                    Macro Expansion
                    10*2
                    Press "F2" for macro expansion steps
    size_t part2 = 0;
    if(part1 > TAM_BUFFER_CON)
        part1 = TAM_BUFFER_CON;
    else
        part2 = TAM_BUFFER_CON-part1;
```

Figura 11: Error difícil de encontrar que involucra al precompilador.

Se solucionó otro problema en la aplicación *Android* que provocaba que un mensaje o sonido no terminara de mostrarse o reproducirse. La forma de resolverlo fue que en el constructor de *PacketIterator* existían las variables *data*

y *this.data que referencian un parámetro del constructor y una variable de objeto respectivamente. S*e estaba modificando *data* para añadirle el símbolo de terminación, por lo cuál no se guardaban los cambios en el objeto y este símbolo no era enviado hacia el despertador.

Se comprobó que el funcionamiento de la reproducción de sonido era errática en melodías muy rápidas. Se consiguió solucionar este problema de forma sencilla, simplemente añadiendo la función *noTone()* antes de llamar a la función *tone()*, para así evitar que siguiera sonando un sonido anterior e impedir que sonara correctamente el siguiente.

Los caracteres especiales fuera de la tabla *ASCII* como las eñes no se visualizan correctamente. Esto es debido a que *Android* envía el texto codificado como *UTF-8*, que además utiliza más de un byte para este tipo de caracteres. Este problema será solucionado en otra iteración por medio de la conversión a otra codificación.

La funcionalidad que añadí de prolongar la duración de una nota al doble añadiendo varias ligaduras sin notas a continuación no funcionaba correctamente. Esto es debido a que el método *split()* de la clase *String* utilizado no muestra las separaciones entre caracteres del final si lo que separan son cadenas vacías. Teniendo en cuenta esto se solucionó el funcionamiento.

El funcionamiento del sonido y de los mensajes enviados al mismo tiempo es correcto, lo cuál garantiza la viabilidad de este diseño para implementar las notificaciones en futuras iteraciones. Sin embargo cuando se envían notificaciones solo funcionan los mensajes y el sonido. El led de notificaciones no se activa la mayoría de las veces. Es necesario arreglar este funcionamiento en otra iteración posterior.

5.2.5. Evaluación del resultado

Las funcionalidades llevadas a cabo en esta iteración allanan el camino para que en las próximas iteraciones ya se puedan mostrar las notificaciones recibidas desde el teléfono. Por lo que recibir las notificaciones del teléfono será uno de los requisitos de la próxima iteración.

5.3. Iteración tercera

5.3.1. Planteamiento y requisitos previstos

Esta iteración se enfoca principalmente en el funcionamiento interno de la aplicación *Android* y en el arreglo de fallos.

Primero se debe investigar el problema que hace que a veces se pierdan segmentos en la conexión.

Se deben arreglar los fallos que provocan que la aplicación no retenga la conexión en los cambios de pantallas e implementar un sistema para que las notificaciones que lleguen al teléfono se reflejen en el despertador. En la parte de *Arduino* deben investigarse los problemas en la reproducción de melodías largas.

Por último se debe implementar un mecanismo para leer las notificaciones del teléfono y reflejarlas en el despertador por medio de un mensaje con sonido y luz de notificación.

5.3.2. Diseño y desarrollo en Arduino

Se han resuelto los problemas de reproducción de melodías. Estos problemas surgían porque algunas melodías de prueba contenían la nota silencio de blanca con ligadura. Esta nota está reservada para la terminación de una melodía.

5.3.3. Diseño y desarrollo en Android

Se ha comenzado solucionando la pérdida de conexión con el despertador moviendo las llamadas de búsqueda y creación de conexión del despertador del método *onResume* al método *onCreate* para evitar que se realicen cada vez que se cambia de aplicación.

Debido a la falta de información, se ha realizado una investigación acerca del problema de la pérdida de segmentos en la conexión. Se ha estimado que este fallo podría provenir por funcionamiento del método *WriteCharacteristic.* El funcionamiento de este método es asíncrono y parece que en una misma aplicación no se implementa un mecanismo de colas para los mensajes [26].

Debido a esto cuando se envían varios segmentos al mismo tiempo como es el caso de las notificaciones, algunos datos se sobrescriben antes de enviarse, y por tanto se pierden. Para solucionar este problema se ha recurrido al uso de una cola con métodos sincronizados, la cuál irá recibiendo los mensajes y los irá procesando según terminen de enviarse los anteriores. Aprovechando la inclusión de la cola se ha refactorizado el mecanismo de envío de mensajes por puerto serie a la clase *SerialPort* para desacoplarlo de la clase *Despertador*.

Para leer las notificaciones en *Android*, se debe añadir a la aplicación una entrada en el archivo de manifiesto del proyecto, indicando un nuevo servicio así como crearlo en el código fuente [27]. Para leer las notificaciones en un móvil con la aplicación instalada, es necesario que esta obtenga un permiso especial del usuario desde la aplicación de ajustes del teléfono.
Para utilizar este servicio la mejor forma de obtener información de él es mediante una suscripción o *binding [28] [29] [30]*. La actividad Despertador se suscribirá al servicio mediante un método de *callback que será el que llame el servicio receptor de notificaciones.* Al mismo tiempo la actividad Despertador se modificará para que ésta también sea un servicio. De esta forma se garantizará que cada uno de los componentes de la aplicación funcionen aunque la aplicación no se encuentre en primer plano, asegurando la persistencia de la conexión aunque la aplicación no se esté ejecutando. Gracias a que los servicios se ejecutan en la misma aplicación es posible el uso de métodos de *callback*, no siendo necesario utilizar otros mecanismos de paso de mensajes en *Android* diseñados para lo que se conoce como IPC (*Inter-Process Communication*).

5.3.4. Pruebas de funcionamiento

Para agilizar la comprobación el correcto funcionamiento de la recepción de notificaciones y su posterior envío y aparición en el despertador, se ha utilizado una aplicación de envío de notificaciones personalizadas disponible en la *Play Store [31]*. Las notificaciones aparecen en el teléfono y justo después se mues-

tran en el despertador con el nombre de la aplicación, el contenido principal, una melodía de notificación y el led de notificaciones.

5.3.5. Evaluación del resultado

En esta iteración se han resuelto las principales dificultades técnicas involucradas en los requisitos principales del trabajo. Sin embargo las notificaciones en el despertador disponen de una melodía y un aviso luminoso del LED que se pueden personalizar, pero que deben ser configurables por el usuario. Además se debe establecer el funcionamiento del botón luminoso para que realice la acción de descartar la notificación recibida. También se espera recibir mensajes con caracteres latinos como la eñe o las tildes, que actualmente se visualizan de forma incorrecta. Esto debe ser arreglado en la próxima iteración.

5.4. Iteración Cuarta

Esta iteración se centra principalmente la aplicación móvil, concretamente en la gestión de las notificaciones.

Los objetivos de esta iteración son los siguientes:

- Permitir al usuario personalizar la melodía, el color del led y su frecuencia de parpadeo para las aplicaciones especificadas. Las aplicaciones no personalizadas no deberían aparecer en el despertador para evitar molestias innecesarias.
- Permitir descartar la notificación del teléfono por medio del botón luminoso del despertador cuando se muestre una notificación en él.
- Arreglar el fallo de visualización de notificaciones con eñe o tildes.

5.4.1. Planteamiento y requisitos previstos

5.4.2. Diseño y desarrollo en Arduino

La actividad Led se ha modificado para que reciba las pulsaciones del botón y envíe un mensaje “l”, avisando al teléfono de que descarte la notificación que se está retransmitiendo actualmente.

Se ha comprobado que la eñe y las tildes de tabla de caracteres que viene por defecto en la biblioteca MD_MAX72XX coinciden con la llamada Página de códigos 850 de IBM-PC, también conocida como “CP850”, por lo que la aplicación *Android* debe enviar los mensajes en esa codificación.

5.4.3. Diseño y desarrollo en Android

Para poder personalizar la forma de aviso de las notificaciones se debe utilizar una nueva actividad en la cuál se permita elegir la aplicación de la cuál se vayan a personalizar sus notificaciones para posteriormente aparecer en una lista. Esta lista de aplicaciones debe almacenarse de forma permanente.

Se ha creado la actividad *MelodyActivity* en la cuál se pueden añadir melodías a una lista escribiéndolas en un campo de texto acompañado del tempo que indica la velocidad de la melodía. También se ofrece la opción de reproducir las melodías en el despertador para comprobar si suenan correctamente. En el Anexo se explica cómo escribir melodías para el despertador.
Se ha creado la actividad *NotificationConfigActivity* en la cuál se puede añadir una aplicación de la lista de aplicaciones del teléfono. Una vez seleccionada se añade a una lista de notificaciones activadas junto con un botón que permite modificar la melodía, el color del led de notificaciones y su tiempo de parpadeo. Las notificaciones pertenecientes a las aplicaciones de esta lista serán las mostradas por el despertador.

Para el almacenamiento persistente de configuraciones se ha utilizado la *API* de persistencia *Room* [32]. Esto permite una abstracción del mecanismo de persistencia *SQLite* integrado en *Android*. Para almacenar cada notificación personalizada se han creado las clases de entidad *NotificationAlert* y *Melody* [33].

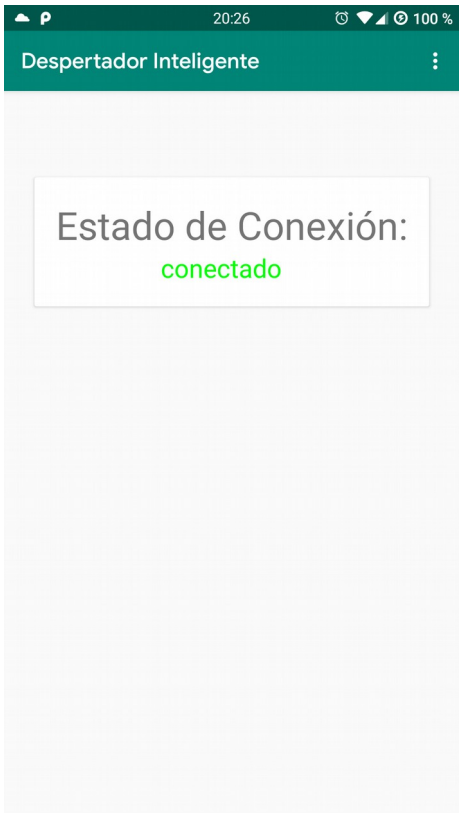

Figura 12: MainActivity

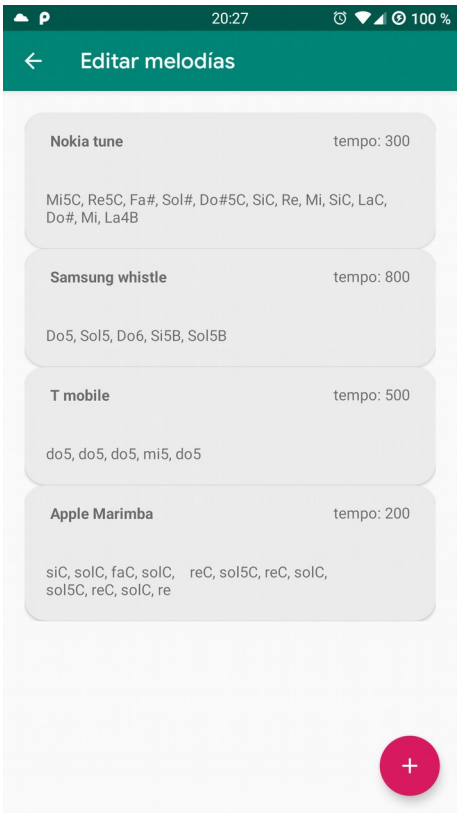

Figura 13: MelodyActivity.

Se ha modificado la actividad *MainActivity* sustituyendo los elementos de prueba por una interfaz pensada para el usuario final con información sobre la conexión como aparece en la Figura 12. En el menú desplegable de opciones se han añadido opciones para añadir melodías y crear notificaciones personalizadas, las cuales dirigen a las actividades *MelodyActivity (Figura 13)* y *NotificationConfigActivity (Figura 14)* respectivamente. Para el selector de color del editor de alertas de notificaciones se ha utilizado la biblioteca *Color Picker* de *QuadFlask* [34] como se puede ver en la Figura 15.

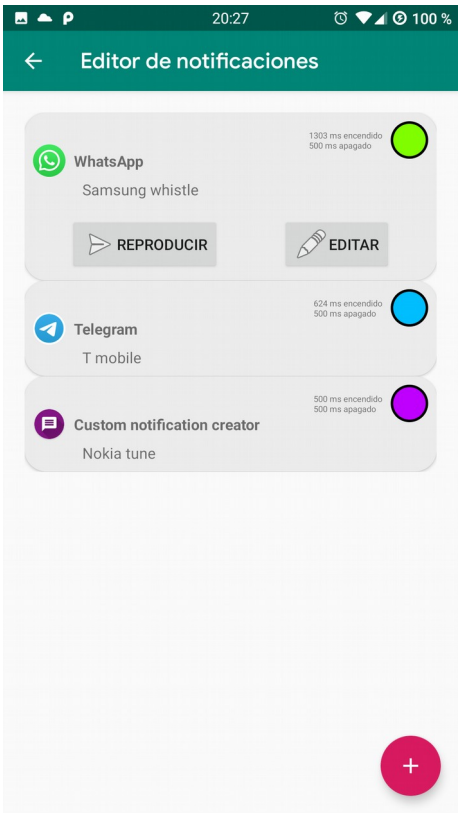

Figura 14: NotificationActivity.

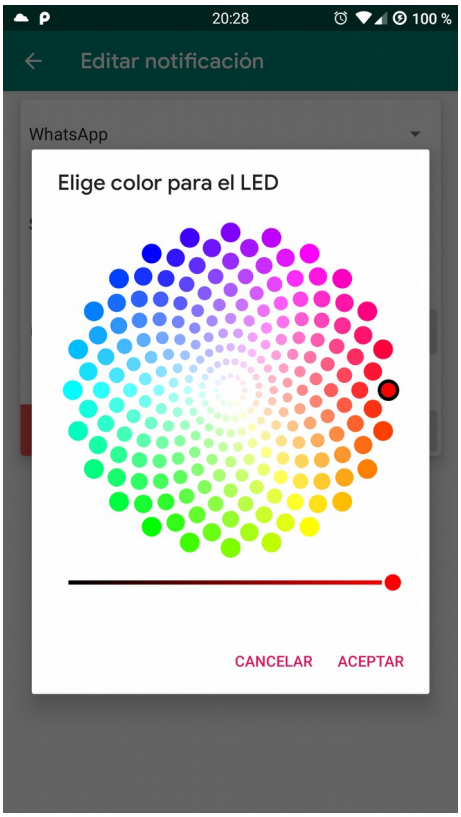

Figura 15: Selector de color.

Se ha resuelto el fallo de visualización de caracteres con eñe o tildes en las notificaciones transformando el texto en la aplicación *Android* a la codificación "CP850" [35] que es una codificación de ancho fijo de 8 bits utilizada por defecto en el la tabla de caracteres de MD_*MAX72XX*.

5.4.4. Pruebas de funcionamiento

Se han creado una serie de melodías y notificaciones de prueba para diversas aplicaciones para comprobar el correcto funcionamiento.

Se han arreglado pequeños fallos que provocaban el bloqueo del despertador en situaciones no controladas. Por ejemplo en el caso de que se reciban muchas notificaciones al mismo tiempo.

Durante las pruebas de notificaciones faltaban tonalidades intermedias del led de notificaciones. Se ha comprobado que el problema se encontraba en el último eslabón de todo el proceso de comunicación, el método utilizado en el *Arduino* para modificar el encendido de los pines del led *RGB* era digitalWrite()

en lugar de analogWrite() por error. Esto conllevaba que cada uno de los colores primarios solo pudiera tomar dos estados.

5.4.5. Evaluación del resultado

En esta iteración se ha conseguido implementar la funcionalidad principal del sistema. Ya se puede utilizar el despertador para mostrar y descartar notificaciones del teléfono. La siguiente iteración se dedicará a implementar unas cuántas funciones accesorias.

Durante esta etapa se ha comprobado que sería buena idea poder descartar las notificaciones en curso, ya que hasta que no se termina el texto o la melodía siguen sonando. Esto se implementará en la siguiente iteración.

5.5. Iteración quinta

Esta será la última iteración presentada debido al agotamiento del tiempo disponible. Realmente como no había una serie de objetivos finales, podrían desarrollarse iteraciones posteriores. Al menos cuando la memoria de programa disponible en el despertador no fuera un limitante para ello.

5.5.1. Planteamiento y requisitos previstos

Se van a implementar las siguientes funcionalidades:

- Mejora del mecanismo de actividades permitiendo más actividades en segundo plano.
- Detener las notificaciones en curso en la pantalla del despertador
- Desactivar las notificaciones en el despertador tras haber sido descartadas en el teléfono.
- Desarrollo de una versión del clásico juego de memoria Simón como una actividad del despertador.

5.5.2. Diseño y desarrollo en Arduino

Aunque se ya existían los métodos para iniciar y terminar actividades en la clase *Actividad.cpp*, lo único que permitían era un máximo de dos aplicaciones. Se ha mejorado implementando una estructura de datos de pila utilizando un

array de punteros a actividades. Cada aplicación que se inicie añadirá una referencia al array y se colocará como actividad principal. Sin embargo, si la aplicación que se inicia es del mismo tipo que la última, esta será reemplazada para evitar llenar la memoria con un número excesivo de notificaciones. Para poder conocer el tipo de la actividad que se está ejecutando debido a que no existe una forma confiable de hacerlo [36], se ha optado por añadir un método para obtener un identificador del tipo de actividad que será implementado por las actividades que lo necesiten.
Para detener las notificaciones en curso se ha optado por que sea el teléfono móvil el que reciba la orden de descartar la notificación y avise al despertador con una notificación notificación sin texto, sin sonido y con el led apagado para que reemplace a la que se estaba mostrando previamente. Se han modificado las actividades encargadas de crear mensajes, sonido y la luz del led para que terminen justo después de recibir esta notificación especial.

Durante el desarrollo ha surgido un problema extraordinario por el cuál se ha congelado el avance durante un tiempo. En una de las subidas del programa a la placa *Arduino* se ha producido un error y ésta ha dejado de funcionar y recibir nuevos programas. Tras un proceso de investigación se encuentra lo que parece ser el motivo del fallo, descartando que la placa se haya estropeado por un fallo de *hardware*.
El fallo se dio cuando se subió un programa de un tamaño superior a la zona de memoria donde se deben almacenar los programas. Como se aprecia en la Figura 16, justo después de esta sección se encuentra el código cargador de arranque o *bootloader*. El *bootloader* de *Arduino* es el programa encargado de programarse a sí mismo desde la información que le llega por el puerto serie o el USB [37]. Solo las instrucciones contenidas en esta sección de memoria tienen permisos para escribir en otras zonas de memoria. De esta manera se evita que un fallo en un programa subido pueda desprogramar el chip.

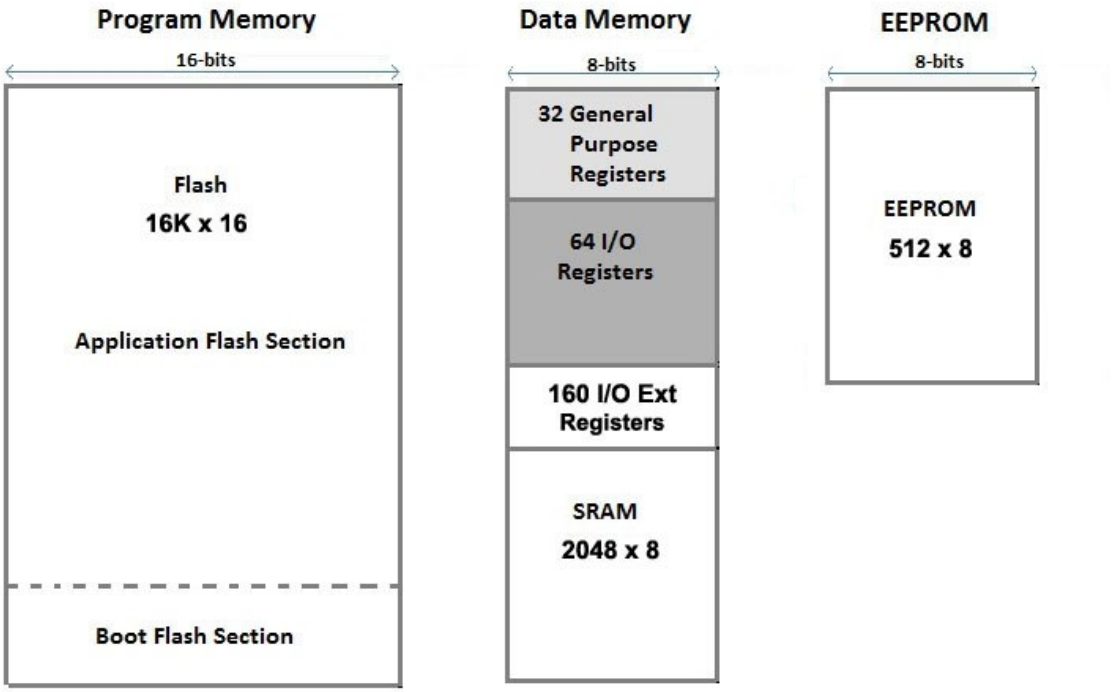

Figura 16: Mapeo de memoria de un chip ATmega328P. Recuperado de https:// www.Arduino.cc/en/Tutorial/ArduinoISP

El *IDE* oficial de *Arduino* comprueba previamente el tamaño que ocupa el programa compilado respecto al disponible en la placa. Sin embargo, el *complemento Sloeber* parece no parece realizar esta comprobación. En teoría también existe un mecanismo que impide que se sobrescriba la zona de la memoria con el cargador de arranque [38], pero es posible que el *bootloader* que se encontraba instalado en el chip no fuera el original de *Arduino* y no realizara esta comprobación.

Por falta de dispositivos y componentes electrónicos necesarios para reprogramar correctamente el chip, se ha obtenido una nueva placa *Arduino* similar para poder terminar el desarrollo con la menor dilación posible. Se ha optimizado el código reduciendo principalmente el tamaño de las constantes de texto y revisado la lógica para ahorrar llamadas a funciones realizadas, las cuáles son muy costosas en cuanto al uso de memoria. Finalmente se ha podido subir el código sin problemas, comprobando antes que no excediera los 32256 *bytes* de espacio para programas.

5.5.3. Diseño y desarrollo en Android

Para detener la notificación en curso, cuando el teléfono recibe un aviso de que se ha pulsado el botón luminoso, se manda desde el teléfono una nueva notificación especial. Es la clase *LedIterator* la que se encarga de esto. Esta se añade a la lista de actividades con segmentos pendientes y cuando se recibe un carácter “l” se encarga de mandar una notificación vacía.

5.5.4. Pruebas de funcionamiento

Cuando se pulsa el botón para descartar una notificación después de haberse mostrado, la hora de la pantalla desaparece por unos instantes como consecuencia de recibir una notificación vacía por parte del teléfono. Es un pequeño fallo que debería ser resuelto en una posible iteración posterior.

5.5.5. Evaluación del resultado

El problema del límite de memoria en *Arduino* produciría un cambio importante en los requisitos posteriores de proseguir con el desarrollo. Intentar añadir más funcionalidades del lado del despertador resultaría costoso de llevar a cabo. Debería optimizarse mejor el código y repensarse la estructura de él para ahorrar memoria suficiente. Esto ya de por sí daría lugar a que se produjeran errores sobre lo que ya está implementado, ralentizando aún más el desarrollo.

6

Conclusiones

En este trabajo se han investigado y utilizado muchas tecnologías con el fin de desarrollar un prototipo final que cumpliera al menos las funcionalidades básicas propuestas. La metodología de desarrollo iterativo incremental ha simplificado este desarrollo permitiendo que se añadieran nuevas funcionalidades interesantes que de otro modo no se hubieran llevado a cabo.

Para poder realizar una correcta evaluación de tiempo y costes en los procesos de ingeniería es necesario que se tenga experiencia previa sobre las tecnologías utilizadas. Sin embargo, en la realización de este trabajo se han realizado muchas actividades para las cuáles no se tenía la certeza de la complejidad y de la curva de aprendizaje requerida. Aunque todos los trabajos de ingeniería conllevan riesgos, esto lo ha aumentado aún más.
En un desarrollo iterativo incremental con un equipo de varias personas experimentadas en la tecnología que vayan a utilizar es mucho más sencillo hacer estimaciones. Se puede establecer una duración que deben tener las iteraciones y se reparten los requisitos de forma que se puedan cumplir a tiempo en cada iteración.

Este trabajo podría dar lugar a futuros trabajos utilizando como base las estructuras utilizadas en el manejo de actividades en *Arduino* y los conocimientos creación de aplicaciones que utilizan Bluetooth LE y leen notificaciones en

Android. Por ejemplo se podrían desarrollar bombillas inteligentes, estaciones meteorológicas, pianos interactivos que faciliten el aprendizaje musical, juegos *arcade* que compartan información con el teléfono, o incluso un dispositivo *Arduino* directamente conectado a la nube por medio de una conexión Wi-Fi.

Una lista con las funcionalidades que podrían ser implementadas en iteraciones posteriores:

- Aviso, recepción y cancelación de llamadas desde el despertador.
- Programación de alarmas y avisos periódicos en el despertador.
- Mostrar notificas, información meteorológica y cita del día a través de servicios web.
- Mejor gestión de las notificaciones permitiendo navegar por una lista de las notificaciones activas.
- Obtener información de humedad y temperatura del despertador para mostrarlas en la aplicación *Android*.
- Sincronizar la hora del despertador con la del teléfono móvil.
- Apartado de ajustes donde poder cambiar el mensaje de bienvenida, el brillo de la pantalla o la velocidad de los textos.
- Cronómetro y cuenta atrás.
- Modo no molestar autoprogramado a ciertas horas donde se reduzca el brillo de la pantalla y se desactiven las notificaciones.
- Integración con la aplicación *Tasker*.

En una fase tardía del desarrollo se conoció la existencia de una implementación para *Arduino* del sistema operativo de tiempo real *FreeRTOS*. Este permite la ejecución simultánea de lo que denominan tareas utilizando diferentes planificadores que permiten realizar un sistema determinista y acotado en el tiempo [39], con lo que podría haberse utilizado como sustituto del sistema de actividades implementado desde cero.

7

Referencias

[1] Daniel López - ¿Qué tiene que ver la salud con el móvil? Así va a cambiar la medicina con el 5G - http://blog.orange.es/innovacion/5g-en-la-medicina/

[2] Wikipedia - Asistente virtual - https://es.wikipedia.org/wiki/Asistente_virtual

[3] proyectosagiles.org - Desarrollo iterativo e incremental - https://proyectosagiles.org/desarrollo-iterativo-incremental/

[4] Wikipedia - Do it yourself - https://en.wikipedia.org/wiki/Do_it_yourself

[5] Arduino - WHAT IS ARDUINO? - https://www.arduino.cc/

[6] Arduino - Trademark - https://www.arduino.cc/en/Trademark/HomePage

[7] Mediatek Labs - Driving 8x8 Dot Matrices with MAX7219 - https://docs.labs.mediatek.com/resource/linkit7697-arduino/en/tutorial/driving-8x8-dot-matrices-with-max7219

[8] Wikipedia - Síntesis aditiva de color - https://es.wikipedia.org/wiki/S%C3%ADntesis_aditiva_de_color

[9] Wikipedia - Fritzing - https://en.wikipedia.org/wiki/Fritzing

[10] - About .INO and .CPP files - https://www.visualmicro.com/page/User-Guide.aspx?doc=INOs-and-CPPs.html

[11] Wikipedia - Java (lenguaje de programación) - https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[12] Wikipedia - Persistencia de la visión - https://es.wikipedia.org/wiki/Persistencia_de_la_visi%C3%B3n

[13] MARCO_C - Parola A to Z – Optimizing Flash Memory - https://arduinoplusplus.wordpress.com/2018/09/23/parola-a-to-z-optimizing-flash-memory/

[14] Wikipedia - Eclipse (software) - https://es.wikipedia.org/wiki/Eclipse_(software)

[15] Jantje - Where Arduino meets Eclipse - http://eclipse.baeyens.it/

[16] Wikipedia - Android - https://es.wikipedia.org/wiki/Android

[17] Wikipedia - Desarrollo de programas para Android - https://es.wikipedia.org/wiki/Desarrollo_de_programas_para_Android

[18] Android Developers - Actividades - https://developer.android.com/guide/components/activities.html?hl=ES

[19] Wikipedia - Mapeo objeto-relacional - https://es.wikipedia.org/wiki/Mapeo_objeto-relacional

[20] Wikipedia - Android Studio - https://es.wikipedia.org/wiki/Android_Studio

[21] Wikipedia - Bluetooth Low Energy - https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

[22] BluegigaAdmin - BLE master/slave, GATT client/server, and data RX/TX basics - https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2015/08/06/_reference_ble_mas-gviy

[23] Wikipedia - Callback - https://es.wikipedia.org/wiki/Callback_(inform%C3%A1tica)

[24] Arduino - Debounce - https://www.arduino.cc/en/Tutorial/Debounce

[25] Wikipedia - Workaround - https://en.wikipedia.org/wiki/Workaround

[26] StackOverflow - Android BLE BluetoothGatt.writeDescriptor() return sometimes false - https://stackoverflow.com/questions/47097298/android-ble-bluetoothgatt-writedescriptor-return-sometimes-false/47103533#47103533

[27] Android Developers - NotificationListenerService - https://developer.android.com/reference/android/service/notification/NotificationListenerService.html

[28] Android Developers - Service - https://developer.android.com/reference/android/app/Service.html

[29] Android Developers - Service Guide - https://developer.android.com/guide/components/services

[30] Android Developers - Servicios Enlazados -

[31] Blackbirds Studio - Custom notification creator - https://play.google.com/store/apps/details?id=net.net16.zaileek_studio.androidsystemcustomsend

[32] Android Developers - Room Persistence Library - https://developer.android.com/topic/libraries/architecture/room

[33] Android Developers - Defining data using Room entities - https://developer.android.com/training/data-storage/room/defining-data

[34] QuadFlask - Color picker for Android - https://github.com/QuadFlask/colorpicker

[35] Wikipedia - Página de códigos 850 - https://es.wikipedia.org/wiki/P%C3%A1gina_de_c%C3%B3digos_850

[36] Stack Overflow - C++ equivalent of instanceof - https://stackoverflow.com/questions/500493/c-equivalent-of-instanceof

[37] Arduino - Bootloader Development - https://www.arduino.cc/en/Hacking/Bootloader

[38] SM - Arduino as ISP and Arduino Bootloaders - https://www.arduino.cc/en/Tutorial/ArduinoISP

[39] Phillip Stevens - Using FreeRTOS multi-tasking in Arduino - https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc

Anexo A

Creación de un prototipo

Contenido:

A.1 Desglose de precios de los componentes

Producto	Precio I.V.A. + envío	Lugar de compra
Placa *Arduino* UNO compatible Lafvin r3 con cable	2,74 €	https://es.aliexpress.com/item/High-Quality-UNO-R3-MEGA328P-CH340-CH340G-for-Arduino-UNO-R3-USB-Cable-Free-Shipping/32518392000.html
Kit de resistencias, leds, botones, zumbador	4,09 €	https://es.aliexpress.com/store/product/New-Electronics-Compo-nents-Basic-Starter-Kit-for-Arduino-UNO-MEGA2560-Raspberry-Pi-with-LED-Buzzer-Capacitor/1590050_32915480309.html
Cables para placas *Arduino*	2,73 €	https://es.aliexpress.com/store/product/30pcs-Jump-Wire-Cable-

Producto	Precio I.V.A. + envío	Lugar de compra
		Male-to-Male-Jumper-Wire-for-Arduino-Breadboard-1-bag/1544094_32324278358.html
Sensor de humedad y temperatura dht11	1,27 €	https://es.aliexpress.com/store/product/orignial-sensor-Temperature-and-humidity-sensor-module-Fapplication-DHT-11-DHT11-PCB/1962508_32562935622.html
Placa de prototipado mini	1,48 €	https://es.aliexpress.com/store/product/400-Tie-Points-Solderless-PCB-Breadboard-Mini-Universal-Test-Protoboard-DIY-Bread-Board-Bus-Test-Circuit/1086484_32967520219.html
Reloj DS3231 AT24C32 CII RTC	1,56 €	https://es.aliexpress.com/item/1PCS-DS3231-AT24C32-IIC-Precision-RTC-Real-Time-Clock-Memory-Module-For-Arduino-new-original/32830730519.html
2 pilas de reloj 3,6 V LIR2032 lir 2032 recargable	2,21 €	https://es.aliexpress.com/item/2-piezas-3-6-V-LIR2032-lir-2032-recargable-de-iones-de-litio-bater-a-de/32862545146.html
Pantalla de 4 matrices leds con controladores MAX7219	4,41 €	https://es.aliexpress.com/item/MAX7219-Dot-Matrix-Module-For-Arduino-Microcontroller-4-In-One-Display-with-5P-Line/32789160918.html
Módulo extra matriz led con controlador	1,59 €	https://es.aliexpress.com/item/Envo-Gratis-1-piezas-MAX7219-matriz-

Producto	Precio I.V.A. + envío	Lugar de compra
MAX7219		m-dulo-microcontrolador-m-dulo-M-dulo-de/32681183937.html
Módulo *Bluetooth* LE HM-10 compatible	2,20 €	https://es.aliexpress.com/item/Freeshipping-HC-06-Wireless-Blue-tooth-Module-With-baseboard-Slave-Wireless-Serial-Port-for-Ardu-ino-with-Dupont/1568545615.html
Botón máquina arcade	2,71 €	https://es.aliexpress.com/item/5-Colors-LED-Light-Lamp-60MM-Big-Round-Arcade-V*IDE*o-Game-Player-Push-Button-Switch/32794775928.html
Caja de plástico para circuitos electrónicos	15,00 €	Tienda de electrónica
Precio total:	41,99 €	

A.2 Esquema de conexiones

Antes del montaje de los componentes que componen el despertador, fue necesaria la realización del esquema de conexiones que aparece en la Figura 17 para analizar cómo deberían hacerse las conexiones.

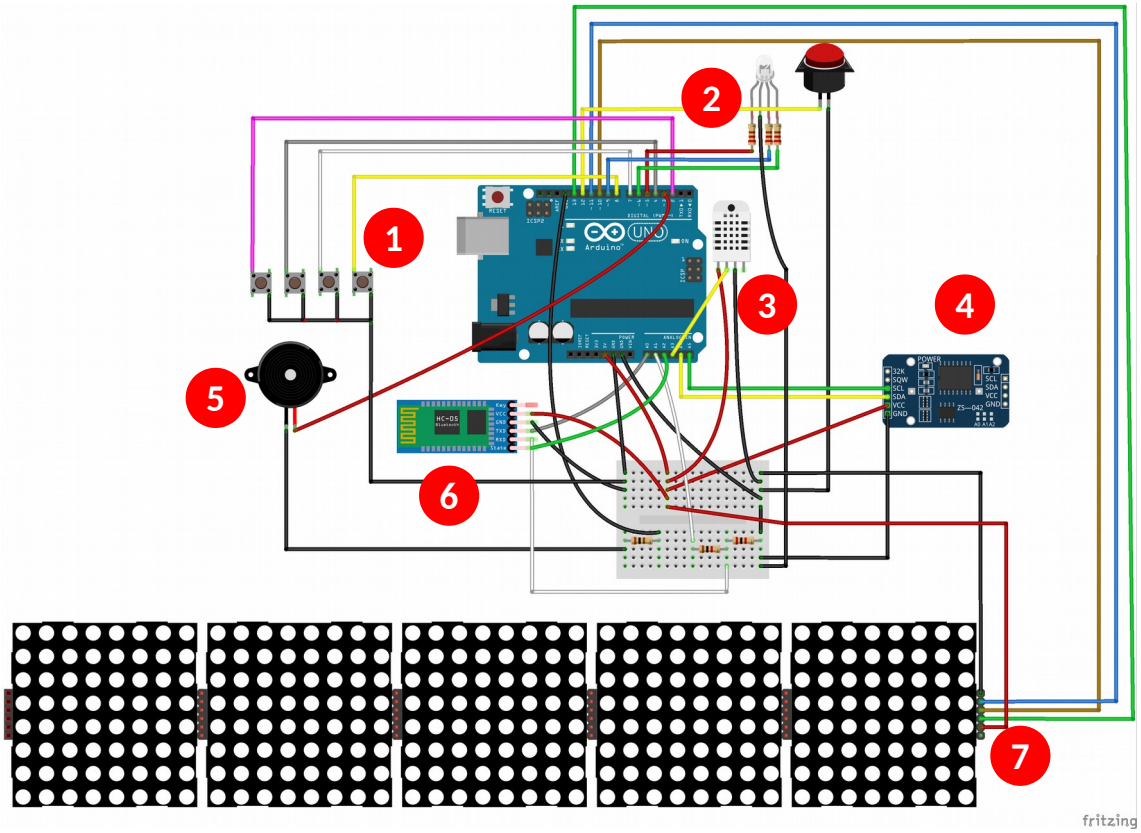

Figura 17: Esquema de conexiones del despertador.

En el esquema aparecen numerados los componentes cuyos circuitos electrónicos se tuvieron que estudiar:

1) **Botones del despertador**

   Los tipos de botones utilizados, en su caso más genérico y también utilizados en el despertador son los pulsadores. Estos son componentes electrónicos que permiten o impiden en el paso de la corriente a través de ellos según hayan sido pulsados o no. Sin embargo, en un circuito electrónico también debemos asegurarnos de que las señales de entrada permanezcan estables. Esto es, si un circuito está cortado debido a que un pulsador no ha sido presionado, debemos asegurar que la corriente tenga donde caer para evitar lo que se conoce como "entrada flotante". De no tener en cuenta esto, se producirían entradas erróneas de las pulsaciones de los botones. Para solucionar este inconveniente hay dos esquemas principales (Figura 18):

2) **Resistencias *pull up*.**

   Lo habitual que uno se puede encontrar en internet en desarrollos sencillos, es el uso de circuitos de entrada conocidos como pull up. Su principales ventaja es que el estado de pulsación del botón se corresponde con un estado de caída de tensión alto que se produce en la entrada.

3) **Resistencias *pull down*.**

   La lógica de las resistencias *pull up* se invierte, pero a cambio se pueden utilizar en *Arduino* sin la necesidad de resistencias extras, puesto que este circuito ya se encuentra en la placa

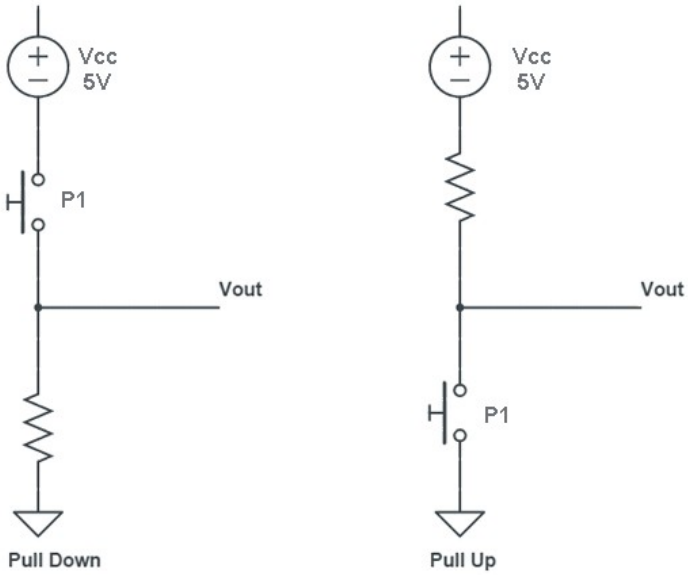

Figura 18: Resistencias pull up y pull down. Recuperado de *https://programarfacil.com/blog/Arduino-blog/resistencia-pull-up-y-pull-down/*

4)

5) **LED *RGB***

   Hay dos tipos principales de led *RGB*, los de ánodo y los de cátodo común. El led de cátodo común es el que tienen el pin negativo común y es utilizado en este proyecto. El resto de las pines corresponden a los colores rojo, verde, y azul, que gracias a la síntesis aditiva de color nos

permite cubrir en principio todo el espectro de colores visible por el ser humano.

Hay que realizar las conexiones con el led *RGB* como si de tres led diferentes se tratase. Para ello debemos poner una resistencia en cada una de las tres entradas. Esto es necesario ya que un LED es un componente que ofrece una resistencia muy pequeña: De la ley de Ohm se deduce que ante una diferencia de potencial constante, la resistencia es muy pequeña la intensidad aumenta enormemente. De no poner una resistencia adecuada tanto la placa *Arduino* como el led acabaría quemándose.

Para hacer el cálculo supongamos que utilizamos un led rojo que produce una caída de tensión de 1,8V y que queremos que circule una corriente típica de 20mA. Para una entrada de 5V la diferencia de potencial que se producirá en la resistencia que añadamos al circuito será la diferencia: 3,2V. Utilizando la ley de Ohm podemos calcular cuál es la resistencia que necesitamos para que circulen 15mA:

$$R=\frac{V}{I}; R=\frac{3{,}2V}{0{,}015A}, R=213,\overline{3}\,\Omega$$

6) Sensor de humedad y temperatura

El sensor viene con una resistencia pull up integrada que nos evita tener que añadirla al circuito. Por lo demás se conectan dos de sus pines a 5V y tierra y el pin restante a uno de los pines de entrada de *Arduino*. Es el propio sensor el que irá sondeando de forma periódica los datos y los transmitirá digitalmente utilizando el pin de entrada.

7) Reloj en tiempo real (RTC)

Dos de sus pines irán conectados a 5V y a tierra. Los otros dos pines establecen un interfaz I2C e irán conectados A4 y A5. Uno de ellos establece cuál es la frecuencia con la que se irán retransmitiendo los datos según se alcance el flanco de subida y el otro de ellos enviará la información. Este interfaz de conexión permite conectar múltiples dispo-

sitivos con diferentes direcciones asociadas, pero nosotros solo conectaremos el reloj.

8) **Zumbador pasivo**

Añadimos una resistencia de 100 ohmios para limitar la corriente en *Arduino* así como reducir su sonido.

9) **Módulo Bluetooth**

Este módulo es más complejo de conectar al *Arduino* porque internamente funciona a 3.3V. Sin embargo se encuentra montado en una placa que reduce el voltaje del pin de entrada, por lo que se conecta directamente a los 5V del *Arduino* y tierra. El pin de estado se activa cuando se encuentra conectado a un dispositivo *Bluetooth*. Sin embargo el resto de pines funcionan a 3V. Para los datos de salida procedentes del módulo no es problema, puesto que *Arduino* puede leer esas entradas. Sin embargo una entrada de 5V procedente del *Arduino* podría dañar el módulo.

Los otros dos pines restantes establecen la interfaz de comunicación del dispositivo con el *Arduino*, que se realiza mediante puerto serie. El pin Tx que es el de salida de datos del módulo al *Arduino* se conecta directamente al puerto Rx de *Arduino*. Sin embargo el pin Rx requiere una conexión de 3.3V, para lo cuál es necesario utilizar un divisor de tensión. Para construir uno nos valemos de la siguiente fórmula:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}}$$

Si tenemos un circuito con dos resistencias conectadas en serie R1 y R2, estando la R1 conectada a una diferencia de potencial V_{in} y R2 a tierra, V_{out} sería la diferencia de potencial entre R1 y R2 y tierra (Figura 19). Para obtener una tensión de 3.3V a partir de 5V (lo cuál son ⅔ de 5), podemos utilizar los valores de resistencias R_1 = 1Ω y R_2 = 2Ω. De esta forma ya podemos conectar la salida Tx de *Arduino* al pin Rx del módulo *Bluetooth*.

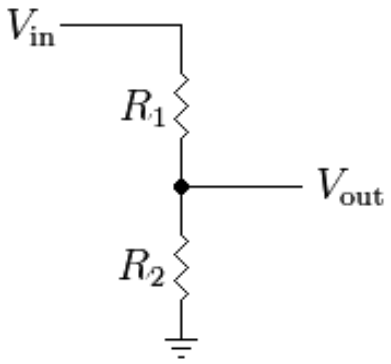

Figura 19: Divisor de tensión. Recuperado de https://es.wikipedia.org/wiki/Divisor_de_tensi%C3%B3n

10) Pantalla con módulos MAX7219

Los módulos MAX7219 permiten ser conectados en cascada utilizando las salidas de cada uno de ellos para las entradas del siguiente. De esta manera conseguimos unir 5 módulos que conformarán la pantalla de nuestro reloj. En el primer módulo deberemos conectar los cables de 5V y tierra. Los otros pines corresponden a la interfaz de conexión SPI y son MISO, CS y SCK:

- MISO (*Master In, Slave Out*) : Este pin se conecta a todos los dispositivos SPI que quieran recibir información del maestro.
- CS (*Chip Select*): Este pin se encarga de habilitar uno de los dispositivos para que reciba información. El uso de este pin tiene sentido cuando hay más de un dispositivo y se requiera el uso de multiplexación de tiempo para ahorrar conexiones. Por tanto en nuestro caso este pin debería de estar siempre activado.
- SCK: Establece la frecuencia con la que se van a comunicar los dispositivos.

Anexo B

Manual de usuario

Contenido:

B.1 Requisitos para poder utilizar el sistema e instalación

Es necesario disponer de un prototipo de despertador así como disponer de un teléfono móvil compatible con la aplicación *Despertador Inteligente* instalada. sin embargo, sin un teléfono móvil conectado aún se podrá hacer uso de algunas funciones del despertador.

La versión de *Android* mínima requerida para que funcionen todas las características es 4.4 KitKat[1]. Además necesitaremos que nuestro dispositivo disponga de conexión Bluetooth LE. Para instalar la aplicación será necesario obtener una copia del archivo con la extensión apk al teléfono y marcarlo en el teléfono para instalar[2].

---

1 En la versión 4.3 podría funcionar aunque sin ciertas funcionalidades. Versiones anteriores a *Android* 4.3 no son compatibles con dispositivos Bluetooth LE.

2 Según la configuración de nuestro teléfono puede que se necesiten permisos para instalar aplicaciones de fuentes desconocidas. Si el sistema no te sugiere qué hacer, hay que ir a Ajustes > Seguridad y activar la opción de Orígenes desconocidos.

Una vez instalada la aplicación hay que ir a la información de la aplicación (Esto se puede hacer dejando pulsado el icono de la aplicación en la pantalla de aplicaciones recientes) y activar el permiso de ubicación (Figura 20). Este permiso es necesario puesto que en las nuevas versiones de *Android* se considera que mediante una conexión Bluetooth se puede localizar la ubicación de un dispositivo. Una vez hecho esto será necesario activar el permiso de acceso a notificaciones para la aplicación. Para ello nos vamos a Ajustes > Aplicaciones y Notificaciones > Acceso especial de aplicaciones > Acceso a notificaciones y marcamos la aplicación Despertador Inteligente[1] (Figura 21).

Una vez seguidos estos pasos la aplicación ya podemos usar la aplicación.

Enchufamos nuestro despertador inteligente, comprobamos que la pantalla de diagnóstico no devuelve ningún error y procedemos a abrir la aplicación *Android*. Una vez abierta, se nos solicitará que activemos el Bluetooth si no lo tenemos activado. Tras ello aparecerá la pantalla principal con un aviso en verde confirmando que estamos conectados al despertador. Un aviso en rojo significará que ha habido algún problema de conexión (Figura 22).

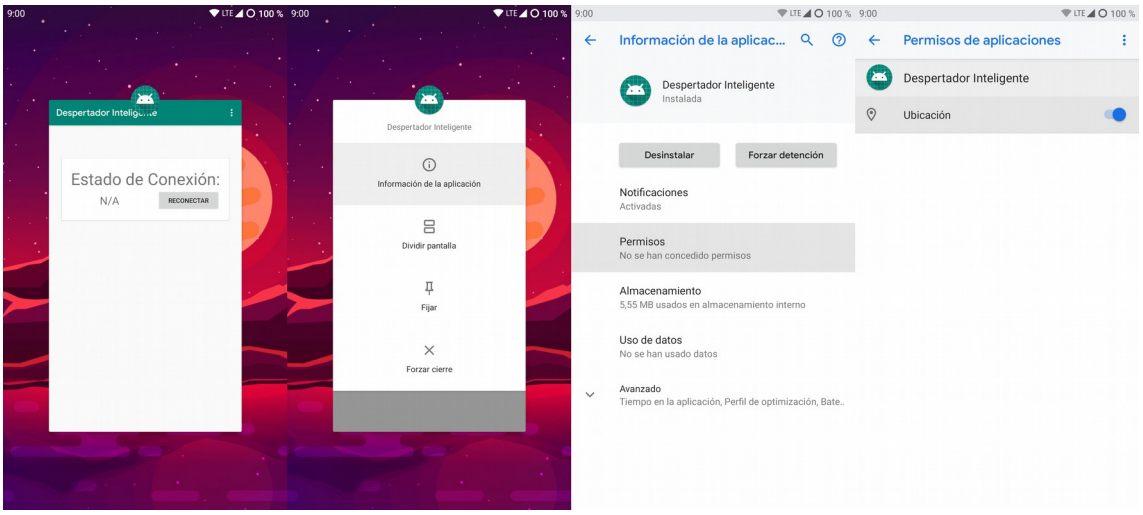

Figura 20: Activación del permiso Bluetooth.

1 En versiones de *Android* anteriores a 9.0 Pie la ubicación de este ajuste podría variar.

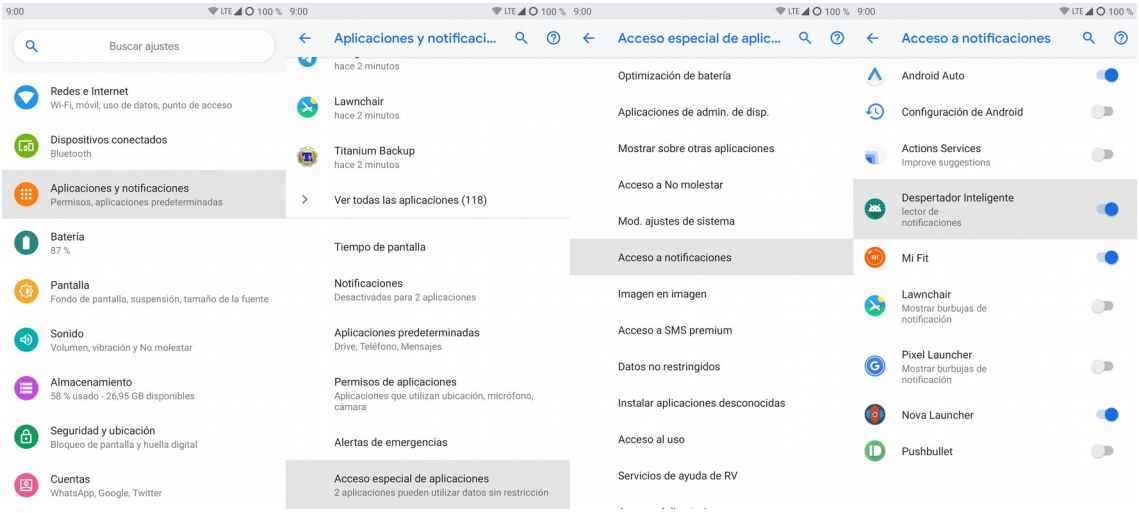

Figura 21: Activación del permiso de notificaciones.

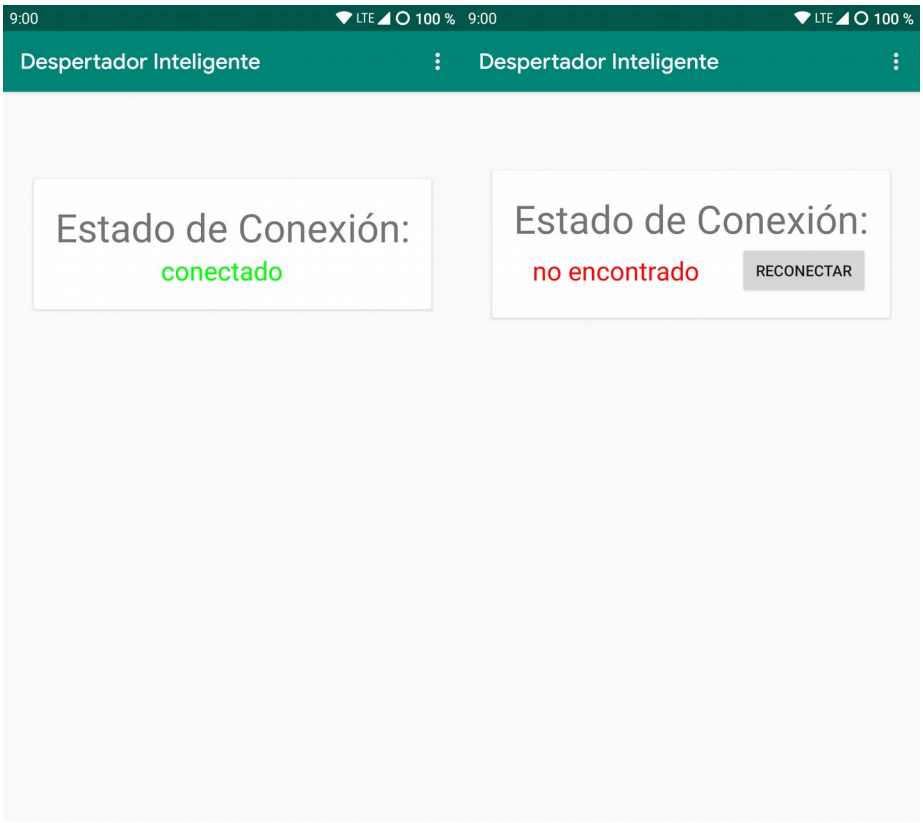

Figura 22: Estados de conexión Bluetooth.

B.2 Componentes del despertador

En la Figura 23 aparecen las diferentes partes de las cuáles se componen el despertador. Los botones “RETROCEDER”, “SUBIR”, “BAJAR” y “ACEPTAR” sirven para desplazarse entre los menús del despertador. El “BOTÓN LUMINOSO” es un botón especial que puede realizar diferentes acciones dependiendo de su estado de luminosidad y el contexto. En la pantalla se mostrará información tanto del propio despertador como la que reciba del teléfono móvil. En la parte trasera se encuentra una conexión USB tipo B de alimentación necesaria para el funcionamiento el dispositivo. Esta conexión también puede utilizarse para recibir actualizaciones de software.

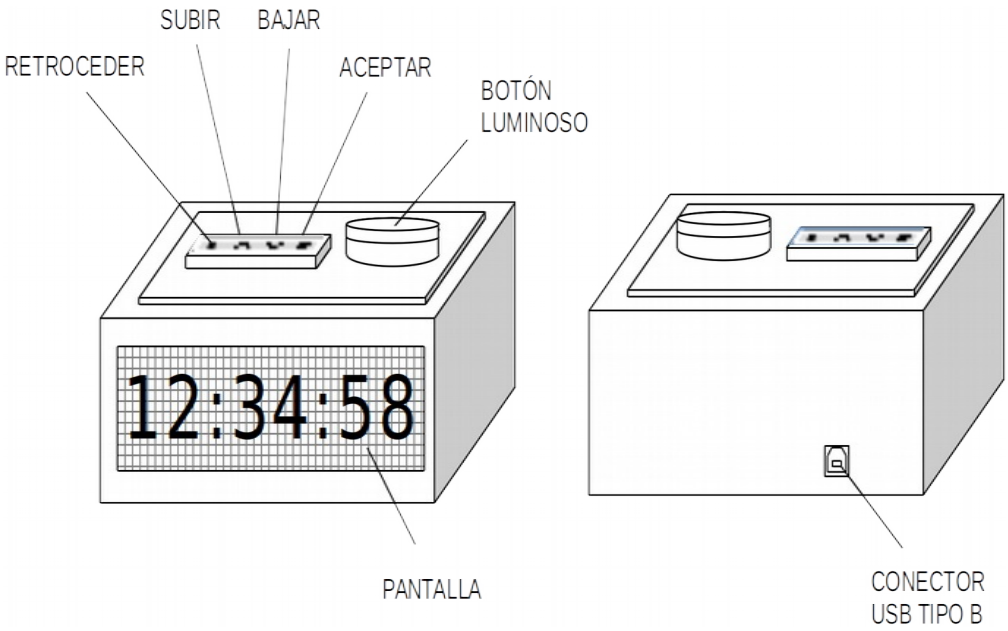

Figura 23: Partes del despertador inteligente.

B.3 Uso de la actividad de Reloj

La actividad de reloj es la primera que se carga al encender el despertador tras la pantalla de diagnóstico. Esta actividad se queda siempre activada mostrando la hora, la humedad o la temperatura. Podemos desplazarnos entre estas tres funciones utilizando las teclas “SUBIR” y “BAJAR”. La humedad se mide en grados Celsius y la humedad en *RH*.

Podemos editar la hora pulsando el botón “ACEPTAR en la pantalla que nos muestra la hora. Tras ello el número de horas empezará a parpadear. Utilizando las teclas “SUBIR” y “BAJAR” podemos modificar los valores. Con la tecla “ACEPTAR” pasaremos a los minutos, si la pulsamos otra vez a los segundos y otra vez volveremos a las horas. Una vez tengamos configurada la hora deberemos pulsar el botón “RETROCEDER”

B.4 Composición de melodías

Es posible crear nuevas melodías que puedan ser utilizadas para avisar de nuevas notificaciones. Para ello debemos ir a la opción “Editar melodías” situada el menú de tres puntos de la pantalla principal de la aplicación.

Dentro de la pantalla aparecerá un botón fucsia con un símbolo más cuya función es añadir nuevos sonidos. Si lo pulsamos aparecerá una nueva pantalla para introducir una melodía. Se nos preguntará su nombre, la melodía y el tempo.

Para introducir las melodía debemos escribir las notas musicales separadas por comas. Por ejemplo “do, re, mi”. A las notas musicales se les asigna por defecto la cuarta octava[1]. Se puede especificar cuál es la escala poniendo un número a continuación de la nota. Se pueden subir y bajar semitonos de las notas añadiendo a continuación de la nota cualquier número de símbolos ‘#’ y ‘~’, que representan sostenidos y bemoles respectivamente. Hay que tener en cuenta que el rango de notas musicales soportadas va desde Sol sostenido en tercera a Do en sexta.

Al final de la nota se puede especificar la duración, que por defecto es de una negra utilizando: ‘B’, ‘N’, ‘C’, ‘S’, para duraciones de blanca, negra, corchea y semicorchea respectivamente. Por último también se pueden añadir ligaduras utilizando el símbolo ‘_’ entre dos o más notas. Si se utiliza este símbolo más de una vez se ligará realizará una ligadura con la nota anterior. Un ejemplo que utiliza todo lo anterior sería:

```
Mi5C, Re5C, Fa#, Sol#, Re~5C, SiC, Re, MiC_MiC, SiC, LaC, Do#, Mi, La4__
```

1 Se utiliza como referencia una frecuencia de la nota La de 440Hz.

El tempo representa el número de figuras negras que se van a reproducir en minuto. Desde esta pantalla podremos probar a reproducir la melodía antes de guardarla.

Una vez creada la melodía aparecerá en la lista de melodías. Tocándola aparecerán las opciones para editarla o reproducirla en el despertador. En la Figura 24 se puede ver un flujo de uso de la aplicación.

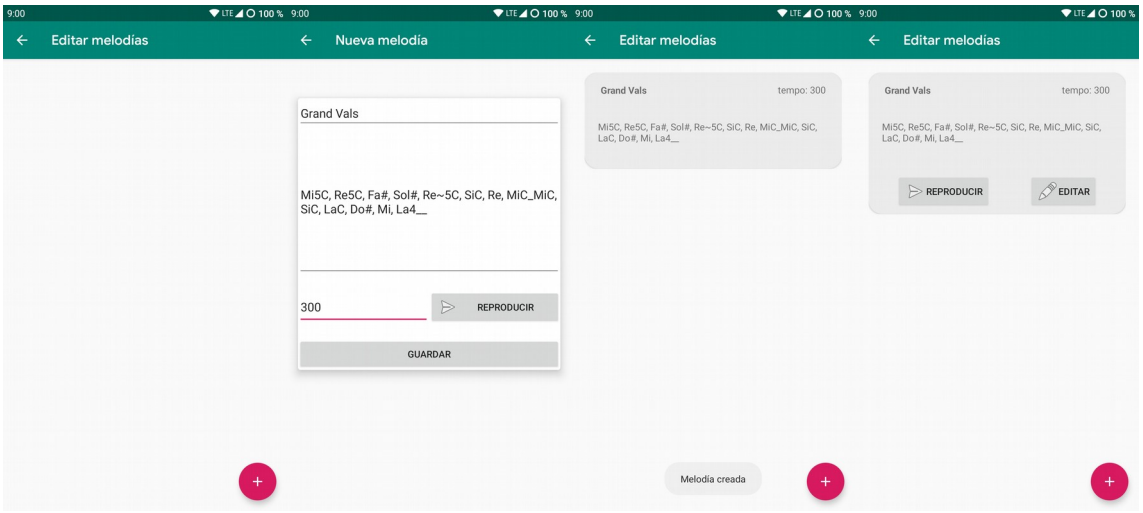

Figura 24: Creación de una melodía.

B.5 Configuración de notificaciones personalizadas

Cuando hayamos creado la primera melodía podemos crear nuestra primera notificación personalizada. Para ello seleccionamos la opción “Editar melodías” desde el mismo menú de opciones de la pantalla principal. Pasamos a crear una nueva notificación pulsando el botón más de color fucsia.

Dentro de la pantalla de crear una notificación seleccionamos la aplicación que queramos activar, seleccionamos una melodía de la lista desplegable y deslizamos para seleccionar un tiempo de encendido y de apagado del led. Si tocamos el círculo se nos aparecerá un selector de color que permite seleccionar un color para el led de notificaciones. Cada vez que pulsemos en un color se nos mostrará en el despertador cómo se verá.

Desde la pantalla de creación de melodías podemos previsualizar cómo quedaría una notificación y guardarla. Tras guardarla apareceremos en la

pantalla principal, y de forma similar a las melodías podremos mandar una notificación de prueba o editarla.

Las notificaciones que aparezcan en la lista serán las únicas que sean mostradas en el despertador. Podemos ver el flujo de la creación de una melodía en la Figura 25.

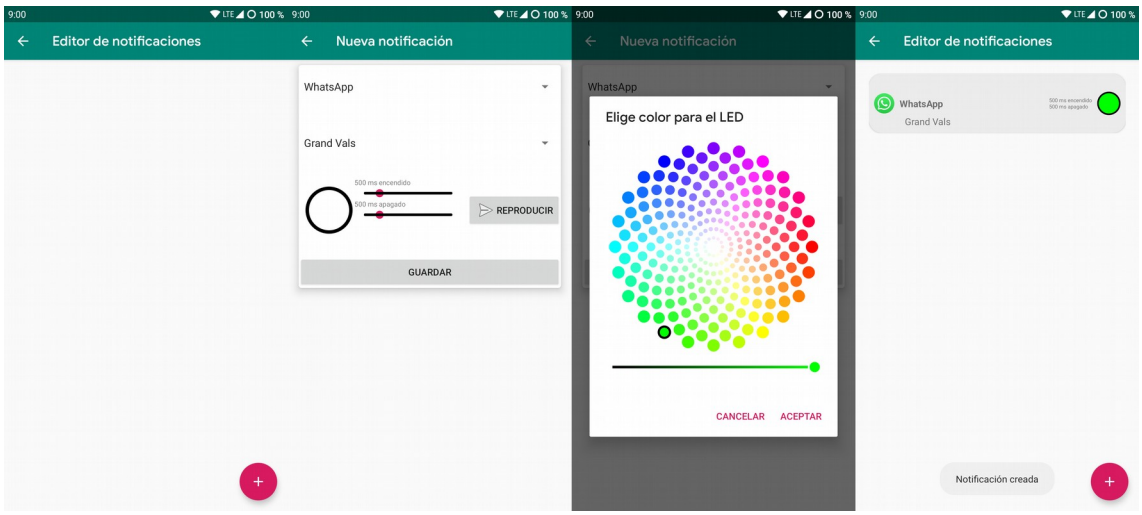

Figura 25: Creación de una notificación personalizada

B.6 Jugar a *Simon*

El despertador posee una versión del clásico juego de memoria *Simon*. Para acceder a él solo es necesario pulsar el botón “RETROCEDER” desde la pantalla donde se muestra la hora. Una vez dentro se reproducirá una música y un mensaje de bienvenida invitándonos a seleccionar una dificultad.

Para seleccionar la dificultad debemos utilizar las teclas “SUBIR” y “BAJAR” que harán que el color del led varíe siguiendo el siguiente código de colores:

Verde	Fácil
Azul	Intermedio
Amarillo	Avanzado
Rojo	Experto

Pulsando el botón luminoso empezará la partida. La partida consiste en la sucesión de rondas en las cuáles hemos de repetir la secuencia de colores que

genera la máquina. Cada ronda aumenta la velocidad y aumenta en un color la secuencia. Para repetir la secuencia generada por la máquina debemos pulsar los botones en el orden mostrado siguiendo la correspondencia según la tabla siguiente:

RETROCEDER	Verde
SUBIR	Azul
BAJAR	Amarillo
ACEPTAR	Rojo

Si nos equivocamos de color o pasa demasiado tiempo sin que se realice ninguna pulsación, la partida se acabará. Cuando esto suceda se mostrará un sonido de fin de partida, un mensaje con la puntuación que hemos conseguido y el led parpadeará del color que debíamos haber pulsado. Si pulsamos el botón grande podremos empezar una nueva partida. Desde la pantalla de nueva partida podremos salir del juego pulsando el botón “RETROCEDER”.