

**APLICACIÓN ANDROID
PARA REPRODUCCIÓN DE AUDIO
EN PLATAFORMA ARDUINO**

Autor: Javier Colomer Barberá

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la “*Universitat Politècnica de València*” (UPV), para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2014-15

Valencia, 7 de julio de 2015

Resumen

El Trabajo Final de Grado (TFG) consiste en la creación de un prototipo de reproductor de audio mediante la plataforma Arduino que está controlado por una app desde un dispositivo móvil con sistema operativo Android.

Para la realización del TFG: se han integrado diferentes campos: el de la programación junto con el de la electrónica y el de las telecomunicaciones. Uniendo estos, se puede desarrollar una amplia variedad de proyectos, entre ellos el que hemos elegido para el TFG.

La funcionalidad de este consiste en qué, mediante una app desde un dispositivo móvil con sistema operativo Android, puedes controlar la reproducción del audio almacenado en la tarjeta μ SD que posee el *Shield* de Arduino. La comunicación WPAN entre ambos dispositivos se establece mediante *Bluetooth Low Energy (BLE)*.

Este control te permite el poner en play, pausa o stop la reproducción de una canción, siempre eligiendo primero la canción que quieres reproducir, junto con los botones de pasar a la siguiente o anterior. También nos aporta el control de volumen, el cual nos proporciona en todo momento el porcentaje de volumen que está saliendo del Arduino hacia los altavoces.

A todo esto, se le debe añadir qué el Arduino nos devuelve a la app de Android los atributos de la canción que se está reproduciendo: título, artista y álbum.

Juntando la parte del *software*, es decir, las aplicaciones que hemos realizado para Arduino (en todo el *hardware* libre utilizado) y para Android (la app *ad hoc* creada), y el montaje del hardware libre que hemos realizado sobre una placa de pruebas (*Breadboard*). El conjunto de todo esto, forma el proyecto que se va a desarrollar en la siguiente memoria.

Resum

El Treball Final de Grau (TFG) consisteix en la creació d'un prototip de reproductor d'àudio mitjançant la plataforma Arduino que esta controlada per una app des d'un dispositiu mòbil amb sistema operatiu Android.

Per a la realització del TFG: se han integrat diferents camps: el de la programació junt amb el de l'electrònica i el de les telecomunicacions. Unint aquestos, es poden desenvolupar una ampli ventall de projectes, entre ells el que hem triat per al TFG.

La funcionalitat d'aquest consisteix en que, mitjançant una app d'un dispositiu mòbil amb sistema operatiu Android, pots controlar la reproducció de l'àudio emmagatzemat a la targeta μ SD que posseeix el *Shield* d'Arduino. La comunicació WPAN entre els dos dispositius s'estableix mitjançant *Bluetooth Low Energy (BLE)*.

Aquest control et permet posar a *play*, *pausa* o *stop* la reproducció d'una cançó, sempre triant prèviament la cançó que vols reproduir, junt amb els botons de passar a la següent o anterior. També ens aporta el control del volum, el que ens indica en tot moment el percentatge de volum que esta enviant del Arduino cap als altaveus.

A tot això, deguem afegir que el Arduino retorna a l'app d'Android els atributs de la cançó que se esta reproduint: títol, artista i àlbum.

Ajuntant la part de *software*, es a dir, les aplicacions que hem realitzat per a Arduino (en tot el hardware lliure utilitzat) i per a Android (l'app *ad hoc* creada), i el muntatge el *hardware* lliure que hem realitzat sobre una placa de probes (*Breadboard*). El conjunt de tot això, forma el projecte que es va a descriure en la següent memòria.

Abstract

The Undergraduate Final Project (TFG) is about the creation of a prototype audio player using the Arduino platform which is controlled by an app on a mobile device with Android system.

To carry out TFG: I have integrated different fields: programming along with the electronics and telecommunications. By doing this, you can develop a wide variety of projects, including the one I chose for TFG.

The functionality consists of controlling the audio playback stored on the μ SD card that Arduino Shield has, through an app on a mobile device with Android operating system. The WPAN communication between the two devices is established via *Bluetooth Low Energy (BLE)*.

This control allows you to put in play, pause or stop the playback of a song, always choosing the song to play, along with buttons to move to the next or previous one. We also provide a volume control, which let us know the percentage of volume that is going from Arduino to the speakers.

Furthermore, Arduino gives to the Android app the attributes of the song being played: title, artist and album.

Together with the software, that is, applications that have been made for Arduino (all free hardware used) and Android (the ad hoc app created), and the free hardware assembly I've done on a Breadboard, the set of all this is forming my project, which I'm going to develop in the following memory.

Índice

Capítulo 1.	Introducción.....	6
Capítulo 2.	Objetivos	7
Capítulo 3.	Propuesta inicial.....	8
Capítulo 4.	Metodología.....	10
4.1	Distribución de tareas	10
4.2	Diagrama temporal	11
Capítulo 5.	Android	12
5.1	Introducción sobre Android	12
5.1.1	Historia de Android.....	12
5.1.2	Evolución de API.....	13
5.1.3	Arquitectura de Android.....	14
5.1.4	Entorno de desarrollo	15
5.1.5	Conceptos básicos	16
5.2	Estructura del proyecto	18
5.3	AndroidManifest	19
5.3.1	Declaración Manifest	20
5.3.2	Versión mínima y permisos	21
5.3.3	Application	21
5.3.4	Activity y Service	22
5.4	Interfaz gráfica	23
5.4.1	Layouts.....	24
5.4.2	Nuestros Layouts	26
5.5	Clases - Actividades y Servicios	28
5.5.1	Estructura de una Clase.....	28
5.5.2	Class MainActivity	31
5.5.3	Class DeviceListActivity.....	34
5.5.4	Class UartService.....	35
5.6	Aplicación.	36
5.6.1	Creación del archivo .apk.....	36
5.6.2	Flujo de funcionalidad de la aplicación.....	38
Capítulo 6.	Arduino	39
6.1	Introducción sobre Arduino	39
6.1.1	Conceptos previos.....	39
6.1.2	¿Qué es Arduino?.....	40

6.1.3	Historia de Arduino.....	41
6.1.4	¿Cómo se programa?.....	41
6.1.5	Bibliotecas (Libraries).....	43
6.1.6	Modelos.....	44
6.2	Dispositivos externos.....	45
6.2.1	Protocolo de comunicación SPI.....	47
6.3	Antena Bluetooth nRF8001.....	48
6.3.1	Conexionado.....	48
6.3.2	Sketch- Parte Comunicación	50
6.3.3	Esquema Parte Comunicaciones	53
6.4	MP3 Player Shield	54
6.4.1	Conexionado.....	54
6.4.2	Puesta a punto del MP3 Player Shield	55
6.4.3	Sketch – Parte Audio.....	55
6.4.4	Esquema Parte Audio.....	59
6.5	Conjunto Parte Arduino	60
Capítulo 7.	Reproductor Arduino	62
7.1	Casos de uso.....	62
7.1.1	Establecimiento de la conexión	62
7.1.2	Desconexión	63
7.1.3	Poner a reproducir.....	63
7.1.4	Petición de información	64
7.1.5	Poner en pausa la reproducción	64
7.1.6	Poner en parada la reproducción.....	65
7.1.7	Petición de pasar a la siguiente/anterior canción	65
7.1.8	Petición de incrementar/decrementar el volumen	66
7.1.9	Reproducción continua.....	66
7.1.10	Finalizado todas las canciones	67
7.2	Repositorio	67
7.3	Apariencia del Proyecto.....	68
Capítulo 8.	Presupuesto.....	69
Capítulo 9.	Propuestas futuras	70
Capítulo 10.	Conclusiones.....	71
	Bibliografía.....	72

Tabla de Figuras

Figura 1.1 Plataformas utilizadas.....	6
Figura 3.1 Antena Bluetooth nRF8001 y escudo reproductor MP3 Player Shield. [7] [11]	8
Figura 3.2 Protocolo comunicación SPI.....	8
Figura 3.3 Colisión en SPI	9
Figura 5.1 Utilización del SO Android	12
Figura 5.2 Arquitectura SO Android.....	14
Figura 5.3 Software Android Studio	16
Figura 5.4 Ciclo de vida de Actividad	17
Figura 5.5 Árbol del proyecto.....	18
Figura 5.6 Fichero de Strings	19
Figura 5.7 Estructura del Manifest.....	20
Figura 5.8 <i>Manifest</i> versión.....	20
Figura 5.9 <i>Manifest sdk/permission</i>	21
Figura 5.10 <i>Manifest Application</i>	21
Figura 5.11 <i>Manifest Activity/Service</i>	23
Figura 5.12 Software de Diseño del Layout	24
Figura 5.13 Partes Layout principal	25
Figura 5.14 <i>LinearLayout</i>	25
Figura 5.17 Estructura todos Layouts	26
Figura 5.15 <i>Layout 7 pulgadas</i>	26
Figura 5.16 <i>Layout 4 pulgadas</i>	26
Figura 5.18 <i>Main Layout</i>	27
Figura 5.19 <i>Device Layout</i>	27
Figura 5.20 Llamada a package	28
Figura 5.21 <i>Imports</i>	28
Figura 5.22 Declaración <i>Class MainActivity</i>	28
Figura 5.23 Declaración <i>Class DeviceListActivity</i>	29
Figura 5.24 Declaración <i>Class UartService</i>	29
Figura 5.25 Declaración <i>onCreate</i>	31
Figura 5.26 <i>onClick</i> Conectar	31
Figura 5.27 <i>onClick</i> Play.....	32
Figura 5.28 Recepción del número total de canciones	33
Figura 5.29 Recepción del título, artista y álbum	33
Figura 5.30 Avisos de siguiente o fin.....	34
Figura 5.31 Declaración de <i>onCreate</i> y llamada a <i>.device_list</i>	35
Figura 5.32 Llamada a <i>.device_element</i>	35

Figura 5.33 Campos de cada dispositivo BLE.....	35
Figura 5.34 Posibles estados de la conexión	36
Figura 5.35 Estado <i>release</i>	36
Figura 5.36 Recompilar Proyecto	36
Figura 5.37 Firmar la aplicación.....	37
Figura 5.38 App	37
Figura 5.39 Máquina de estados de la aplicación	38
Figura 6.1 Estructura del microcontrolador.....	39
Figura 6.2 Software para la programación en Arduino	40
Figura 6.3 Primer prototipo de Placa Arduino.....	41
Figura 6.4 Entorno de desarrollo	41
Figura 6.5 Enlazar placa Arduino	42
Figura 6.6 Declaración de variables.....	42
Figura 6.7 Arduino Uno R3 [9]	44
Figura 6.8 Arduino Mega 2560 R3 [9].....	44
Figura 6.9 Acoplamiento de Shields	45
Figura 6.10 Dispositivo con <i>Breadboard</i>	46
Figura 6.11 <i>Proto Shield</i>	46
Figura 6.12 Protocolo SPI múltiples esclavos	47
Figura 6.13 nRF8001 Breakout [7].....	48
Figura 6.14 Arduino Mega – nRF8001 BLE.....	49
Figura 6.15 Intercambio de datos desde controlado a nRF8001.....	50
Figura 6.16 Cabecera Sketch - Parte Comunicación.....	50
Figura 6.17 Declaración de variables – Parte Comunicación.....	50
Figura 6.18 <i>Setup</i> – Parte Comunicación	51
Figura 6.19 <i>Loop</i> – Parte Comunicación.....	52
Figura 6.20 Acciones cuando está conectado.....	52
Figura 6.21 Acciones cuando está desconectado.....	53
Figura 6.22 Montaje Arduino Mega con Antena Bluetooth nRF8001	53
Figura 6.23 Arduino Uno – MP3 Player Shield.....	54
Figura 6.24 Conexionado MP3 Player Shield [11]	55
Figura 6.25 Cabecera sketch – Parte Audio	56
Figura 6.26 Librerías y declaraciones	56
Figura 6.27 <i>Setup</i> – Parte Audio.....	57
Figura 6.28 Llamamiento a la funcion <i>menu()</i>	57
Figura 6.29 Estado Reproduciendo.....	57
Figura 6.30 Estado Pausa/Stop	57

Figura 6.31 Siguiete canción	57
Figura 6.32 Ultima canción	58
Figura 6.33 Montaje Arduino Uno con MP3Player Shield	59
Figura 6.34 Amplificador <i>TPA2005D1</i> [12].....	60
Figura 6.35 Altavoz 8 Ohmios 0.5W	60
Figura 6.36 Esquema completo Reproductor Arduino.....	61
Figura 7.1 Establecimiento de la conexión.....	62
Figura 7.2 Desconexión	63
Figura 7.3 Poner a reproducir	63
Figura 7.4 Solicitud de información	64
Figura 7.5 Pausar la reproducción	64
Figura 7.6 Parar la reproducción.....	65
Figura 7.7 Avance o retroceso de canciones	65
Figura 7.8 Gestión del volumen.....	66
Figura 7.9 Reproducción continúa	67
Figura 7.10 Ultima canción terminada	67
Figura 7.11 Proyecto.....	68
Figura 9.1 Antena Wi-Fi <i>ESP8266</i>	70

Tablas

Tabla 4.1 Distribución temporal de las tareas	10
Tabla 5.1 Evolución de la API de Android	13
Tabla 5.2 Tipos de variables.....	29
Tabla 5.3. Visibilidad.....	30
Tabla 6.1 Bibliotecas oficiales.....	43
Tabla 6.2 Bibliotecas del fabricante.....	43
Tabla 6.3 Características del Arduino Uno R3	44
Tabla 6.4 Características del Arduino Mega 2560 R3	45
Tabla 6.5 Distribución de pines	48
Tabla 6.6 Pines de interrupción	49
Tabla 8.1 Tabla de materia con precios.....	69

Capítulo 1. Introducción

En los últimos años los dispositivos móviles han pasado a ser una parte esencial de nuestra vida. Las grandes compañías han apostado fuerte en el desarrollo de estos dispositivos y esto ha significado un aumento exponencial de sus características técnicas y de sus repositorios de aplicaciones (apps).

De entre los dispositivos móviles destacan los denominados Teléfonos Inteligentes (*Smartphone*). Estos ya no son solo aparatos de comunicación, sino que son pequeños ordenadores de bolsillo que nos acompañan a todas partes en todo momento de forma ubicua. Lo mismo pasa con las Tabletas (*Tablets*), que aunque, no de manera tan arrolladora como los anteriores, han entrado en el mercado pisando fuerte estos últimos años (desde 2010).

El sistema operativo más usado dentro de los dispositivos móviles es Android. Propiedad de Google y basado en un núcleo de GNU/Linux que junto con las miles de apps disponibles en *Google Play Store* son los principales motivos de su éxito. Además, permite que crees tus propias aplicaciones de una manera relativamente sencilla mediante un entorno de desarrollo integrado (IDE) basado en Java, como puedes ser el Android Studio, junto al SDK Manager de Android.

También está en plena expansión el uso de microcontroladores (desde 2012). Siendo su mayor exponente la plataforma de hardware libre Arduino que está abierta para la creación de prototipos, englobando la parte de hardware (con una placa y un microcontrolador) y software (con un entorno de desarrollo propio). La utilización de estos dispositivos es una buena manera de iniciarse en el mundo de la electrónica y la programación, gracias a su sencillez. Dejando atrás los tiempos en que programar estos tipos de dispositivos solo se podía hacer con lenguajes de bajo nivel tan poco amigables como puede ser el ensamblador o similares.

La gran acogida que ha recibido esta plataforma abierta, en tan poco tiempo, ha hecho que aparezcan una gran variedad de dispositivos, y con ellos muchas posibilidades de hacer nuevos prototipos. Para la creación de los mismos existen multitud de sensores y componentes electrónicos que permiten hacer proyectos sobre cualquier ámbito, cada día aparecen nuevos componentes a utilizar. Este es un campo en plena expansión que permitirá implementar muchas soluciones para el Internet de las Cosas (IoT).



Figura 1.1 Plataformas utilizadas

Capítulo 2. Objetivos

He elegido hacer un proyecto con Android y Arduino, porque el mundo de la programación del *Smartphone* y los microcontroladores está en pleno crecimiento y pienso que consolidar los conocimientos de este campo es importante para la realización de futuras aplicaciones (de las cuales ya tengo alguna más en mente).

El objetivo principal de este proyecto es la creación de un prototipo de reproductor de Audio mediante la plataforma Arduino, controlado por cualquier dispositivo que tenga con sistema operativo Android, y contenga la versión mínima requerida para nuestra aplicación.

Para conseguir este objetivo principal, se plantean una serie de objetivos secundarios:

El primero es familiarizarse en el entorno de desarrollo tanto de Android como de Arduino. Para conocer sus peculiaridades y sus principales facilidades a la hora de crear nuevos proyectos.

En cuanto a la creación del proyecto se divide en dos partes:

- La creación de la App para Android, capaz de comunicar con una antena Bluetooth de Arduino, junto con la creación del código para de configuración y recepción de datos.
- La interpretación y gestión de los datos que reciben, para traducirlos en comandos o estados en los que tiene que estar el *MP3Player Shield* de Arduino.

En este proyecto, se une mi interés por la programación y por la electrónica, ya que todo esto integra un mundo de nuevos prototipos, por lo que empiezo con mucho entusiasmo y ganas.

Capítulo 3. Propuesta inicial

Cuando se planteó hacer el Reproductor Arduino como proyecto, se empezó a leer literatura sobre Arduino, y a buscar posibles *Shields* (escudos) que hicieran la parte de comunicación y reproducción del audio. Decidimos elegir dos escudos que se comunican por protocolo SPI (*Serial Peripheral Interface*). La elección de usar dicho protocolo, fue tomada por las facilidades que ofrece a la hora de comunicar varios esclavos al maestro.

Después de comparar diferentes dispositivos, decidimos hacerlo usando como maestro el **Arduino UNO v3**, y como escudos la antena Bluetooth **nRF8001** y el reproductor de audio **MP3 Player Shield**.

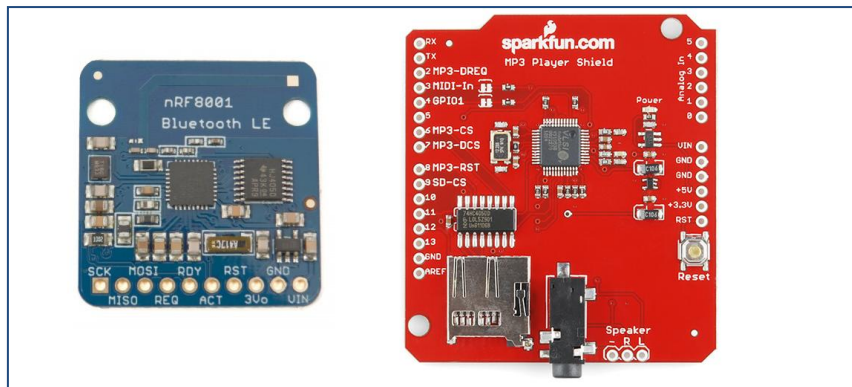


Figura 3.1 Antena Bluetooth nRF8001 y escudo reproductor MP3 Player Shield. [7] [11]

En primer lugar, empezamos programando la parte de comunicación de la antena con la aplicación móvil y, por otra parte, la reproducción del audio almacenado en la tarjeta μ SD del MP3 Player Shield.

Cuando nos disponemos a juntar las dos partes nos encontramos con un problema, en relación con el protocolo de comunicación SPI (se explica con más detalle en el apartado 6.2.1 *Protocolo de comunicación SPI*). Ahora explicaré los conceptos que afecta a este punto.

Dicho protocolo usa cuatro puertos para la comunicación, de los cuales tres son comunes para todos los esclavos (SCLK, MISO, MOSI), el que es diferente para cada dispositivo es el SS o CS (dependiendo del autor lo nombran como *Chip Select* o *Slave Select*, pero ambos son lo mismo). Este es el que marca en cada momento que dispositivo tiene que leer o escribir lo que está pasando por los canales MISO y MOSI.

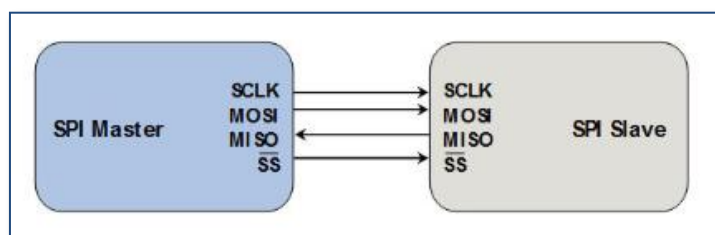


Figura 3.2 Protocolo comunicación SPI

Al juntarlos, observamos que el *MP3 Player Shield* gestiona sus dos *Chip Select* manteniendo todo el canal ocupado (uno para leer datos de la μ SD y otro para mantener el chip procesando la música y reproduciendo). Cuando la antena quiere enviar o recibir datos colisionan dos esclavos.

El *Chip Select* funciona a nivel bajo, es decir, solo puede haber una CS a 0 en el mismo instante de tiempo. En la siguiente figura, se emula lo que pudimos ver que sucedía en el osciloscopio, cuando estaba reproduciendo y la antena entraba a comprobar su conexión o a enviar o recibir información.

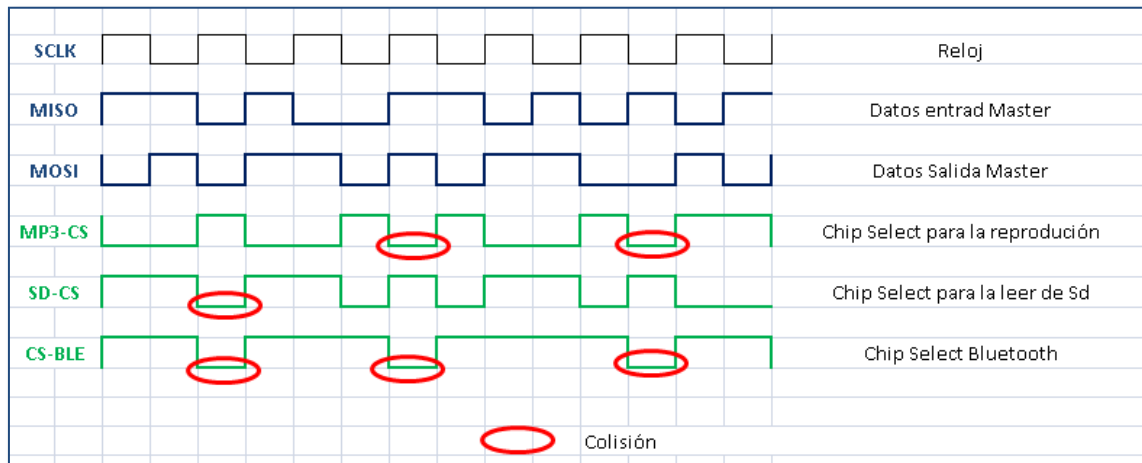


Figura 3.3 Colisión en SPI

Tras analizar las posibilidades de gestionar los *Chips Selects*. Para permitir que cada cierto tiempo la antena refrescara su estado y fuera a mirar si tenía datos que leer o escribir. Dicha acción no funcionaba correctamente, ya que la gestión de los *Chips Selects* del MP3 vienen definidos para tener una bitrate de reproducción.

La solución que se decidió emplear fue la utilización de un segundo Arduino, en este caso, el Arduino MEGA 2560. Así cada uno gestionaría un esclavo SPI y ambos se comunicarían por puertos Serie UART.

El proyecto final quedaría con el Arduino UNO v3 conectado el *MP3 Player Shield* (por SPI) y al Arduino MEGA 2560 se le conectaría la antena Bluetooth (por SPI), ambos conectados por UART serie entre ellos. Elegimos el Arduino MEGA 2560 para las comunicaciones, ya que tiene más puertos UART, y para el uso en futuras ampliaciones del proyecto, como puede ser añadir otros tipos de conectividad.

La principal ventaja que nos proporciona el uso de dos Arduinos es que disponemos de dos microcontroladores y lo que con ello conlleva, poder aportar al proyecto de muchos más dispositivos externos y más funcionalidades. Ya que las diferentes placas Arduino se pueden comunicar entre ellas muy fácilmente.

En los siguientes puntos, se describe el proyecto siguiendo estas últimas pautas, el Arduino MEGA 2560 encargado de gestionar las comunicaciones, (aunque en la actualidad sólo existe una posibilidad comunicación) y el Arduino UNO, que se encarga de gestionar la reproducción de audio con todo lo que eso conlleva.

Capítulo 4. Metodología

En este capítulo se engloba todo lo referente a la planificación del proyecto desglosando cada una de las tareas, esta previsión se ha realizado antes de empezar la realización de éste. Planificamos que el proyecto durase aproximadamente cinco meses, como muestran los siguientes apartados.

4.1 Distribución de tareas

Para el desarrollo del proyecto se han llevado a cabo una serie de tareas, las cuales se muestran en el siguiente cronograma.

TAREAS	FEBRERO 2015	MARZO 2015	ABRIL 2015	MAYO 2015	JUNIO 2015
a. Elección del proyecto *					
b. Búsqueda de bibliografía y documentación					
c. Estudio de conceptos básicos de Android					
d. Estudio de conceptos básicos de Arduino					
e. Elección de dispositivos					
f. Familiarizarse con el entorno de desarrollo					
g. Desarrollo de la memoria					
h. Programación parte 1					
i. Programación parte 2					
j. Integración de las dos partes					
k. Testear la aplicación					
l. Perfilar la interfaz visual					

Tabla 4.1 Distribución temporal de las tareas

A continuación, vamos a realizar una breve explicación de dichas tareas:

- a. **Elección del proyecto** *. Elaboración de la idea principal del proyecto, contrastando mis inquietudes con el tutor y creamos la idea principal. Esto tuvo lugar en Julio del 2014.
- b. **Búsqueda de bibliografía y documentación.** Recopilación de toda la información necesaria para la realización de éste.
- c. **Estudio de conceptos básicos de Android.** Repaso de conceptos de Android dados en la asignatura Aplicaciones Telemáticas, junto con el estudio de nuevas herramientas.
- d. **Estudio de conceptos básicos de Arduino.** Repaso de conceptos de Arduino dados en la asignatura Aplicaciones Telemáticas, ampliando los conocimientos.
- e. **Elección de dispositivos.** Investigación de todos los componentes que puedan hacer las funciones requeridas para este proyecto, eligiendo los más idóneos.
- f. **Familiarizarse con el entorno de desarrollo.** Instalación del Android Studio y el SDK Manager, para la realización de aplicaciones Android, y el software de programación para Arduino. Desarrollo de aplicaciones de ejemplo en ambos entornos.

- g. **Desarrollo de la memoria.** Documentación de todos los procesos que se realizan, y realizar una pequeña guía para realizar proyectos Android-Arduino.
- h. **Programación parte 1.** Realización del prototipo de aplicación en que se comunica el dispositivo externo con sistema operativo Android con la antena *Bluetooth Low Energy* de Arduino. En la cual se intercambia mensajería ASCII entre ambos.
- i. **Programación parte 2.** Realización del prototipo de aplicación en el cual se reproduce música en el Arduino, mediante el uso del *MP3 Player Shield*.
- j. **Integración de las dos partes.** Se engloban las dos partes anteriores creando el prototipo final del proyecto, salvando algunos inconvenientes con el protocolo de comunicación.
- k. **Testear la aplicación.** Comprobar el correcto funcionamiento del proyecto.
- l. **Perfilar la interfaz visual.** Se modifica el interfaz visual de la aplicación para su propósito final.

4.2 Diagrama temporal

En la división de las tareas a realizar, se tuvo en cuenta los siete días de la semana. Se decidió, por motivos laborales, que los días entre semana sólo dedicaríamos un máximo de 3 horas por día. En cambio, los fines de semana se dedicarían el máximo tiempo posible.

La *Tabla 4.1 Distribución temporal de las tareas*, es sólo de ámbito orientativo, ya que nos hemos intentado ceñir a ellas, pero a lo largo de la ejecución ha habido ciertos contratiempos en la implementación de este.

Una vez finalizado el proyecto, cabe mencionar que es difícil diferenciar las horas empleadas en el aprendizaje y las horas usadas en la ejecución, debido a que dichas tareas se han podido hacer en paralelo en la mayoría de estas.

Capítulo 5. Android

5.1 Introducción sobre Android

5.1.1 Historia de Android

El pistoletazo de salida viene marcado en 2005, cuando Google adquiere Android Inc. En ese momento era una pequeña empresa que acababa de ser creada, principalmente orientada a la programación de aplicaciones para móviles.

En 2007 se crea el consorcio *'Open Handset Alliance'* con el objetivo de desarrollar estándares abiertos para móviles. Este consorcio está formado por Google, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone y otros más. El mayor logro de este consorcio fue la creación del sistema operativo Android. Al mismo tiempo Google liberó la mayoría del código fuente bajo la licencia *Apache*, una licencia libre y de código abierto. Para que cualquier fabricante pudiera implementarlo en sus dispositivos.

También en 2007 fue lanzada la primera versión de *Android Software Development kit (SDK)*. El Android SDK es un conjunto de herramientas de desarrollo de software. En la actualidad el SDK contiene un depurador de código, bibliotecas, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales.

Durante el año 2011, se consolida como la plataforma para dispositivos móviles más importante obteniendo cuotas de mercado superiores al 50%.

Según la consultora *"Kantar Worldpanel ComTech"*, podemos observar como en Abril del 2015, el 88.6% de los *Smartphones* utilizados en España usan como sistema operativo el Android, dejando los otros SO muy por debajo en cuota de mercado.

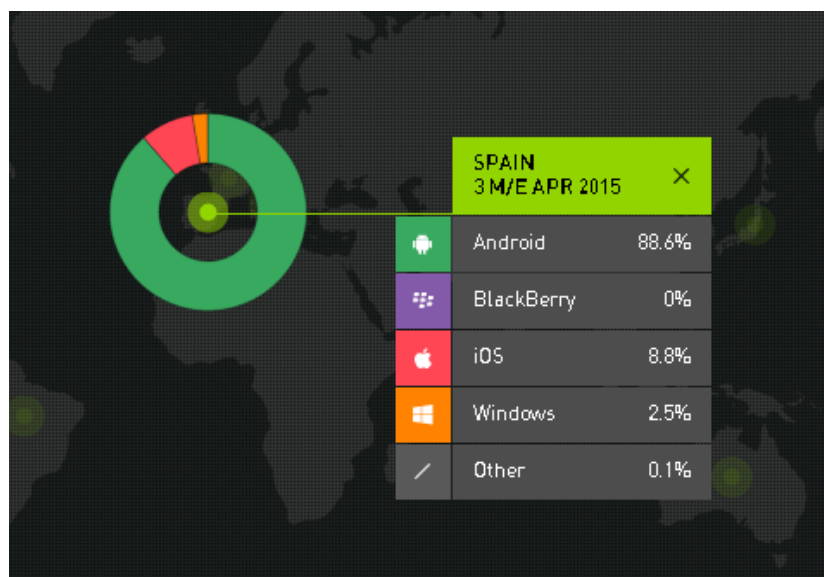


Figura 5.1 Utilización del SO Android

5.1.2 Evolución de API

Como era de esperar las actualizaciones del sistema operativo han ido evolucionando progresivamente desde su lanzamiento hasta la 5.1 que es la más reciente.

Las versiones han ido saliendo por mejoras que los dispositivos requerían, tanto por mejoras de hardware (pantallas más grandes, desaparecían los teclados), como de software (nuevas funcionalidades y requerimientos de los dispositivos, entre ellos la masificación de las redes sociales).

En la siguiente tabla se hace un pequeño repaso de todas las versiones existentes y resaltando alguna de sus características.

VERSION	NOMBRE	API	FECHA	OBSERVACIONES
1.0	<i>Apple Pie</i>	1	Septiembre 2008	Primera versión. No se utilizó comercialmente.
1.1	<i>Banana Bread</i>	2	Febrero 2009	Una App generada en esta versión sería compatible con todos los dispositivos .
1.5	<i>Cupcake</i>	3	Abril 2009	Incorpora teclado en la pantalla con predicción de texto. <i>Widgets</i>
1.6	<i>Donut</i>	4	Septiembre 2009	Permite trabajar en diferente densidad de pantalla. Incorpora <i>gestures y multi-touch. onClick</i>
2.0	<i>Éclair</i>	5	Octubre 2009	Bluetooth 2.1 . Más ajustes de cámara. Más velocidad de <i>hardware</i> . Soporte HTML5
2.1	<i>Eclair</i>	7	Enero 2010	Reconocimiento de voz permite introducir campos de texto sin el teclado.
2.2	<i>Froyo</i>	8	Mayo 2010	Incremento considerable velocidad CPU . Instalar aplicación en un medio externo. Actualización automática de aplicaciones. Compartir internet con el móvil (tethering). WIFI IEEE802.11n
2.3	<i>Gingerbread</i>	9	Diciembre 2010	Soporta diferentes tamaños de pantalla
3.0	<i>Honeycomb</i>	11	Febrero 2011	Exclusivo para Tablets
3.1	<i>Honeycomb</i>	12	Mayo 2011	Permite manejar dispositivos conectados por USB. Optimizaciones para Tablets.
3.2		13	Julio 2011	
4.0	<i>Ice Cream Sandwich</i>	14	Diciembre 2011	Unifica versiones 2.X y 3.X en una compatible tanto para móviles como Tablets. Interfaz renovado . Reconocimiento de voz y facial.
4.1	<i>Jelly Bean</i>	16	Julio 1012	Su enfoque primario fue en mejorar la funcionalidad y el rendimiento de la interfaz de usuario.
4.2	<i>Jelly Bean</i>	17	Noviembre 2012	
4.3	<i>Jelly Bean</i>	18	Julio 2013	Esta actualización tubo bastantes avances en temas de seguridad y otros temas, pero el que nos incunve a nosotros es el soporte para Bluetooth 4.0 .
4.4	<i>Kit Kat</i>	19	Octubre 2013	Actualizaciones para mejorar la interfaz del usuario, y rendimiento del dispositivo.
5.0	<i>Lollipop</i>	21	Noviembre 2014	Pequeñas modificaciones para ahorro de batería . Nuevas maneras de visualizar las notificaciones, etc.
5.1	<i>Lollipop</i>	22	Abril 2015	Modificaciones en la interfaz del usuario del sistema operativo. Como Wi-Fi y Bluetooth en los accesos rápidos.

Tabla 5.1 Evolución de la API de Android

A la hora de crear una nueva aplicación hay que seleccionar a partir de que plataforma de desarrollo la hacemos compatible, teniendo en cuenta si necesitamos algunas características especiales que sólo están disponibles a partir de una versión en concreto. En nuestro caso elegimos la versión 4.3, se argumentaran los motivos de la elección en el apartado 5.3 *AndroidManifest*.

5.1.3 Arquitectura de Android

También es importante conocer la estructura del sistema operativo, en la siguiente Figura muestra la arquitectura de Android. Como se puede observar está formada por cuatro capas. Todas ellas basadas en *software* libre.

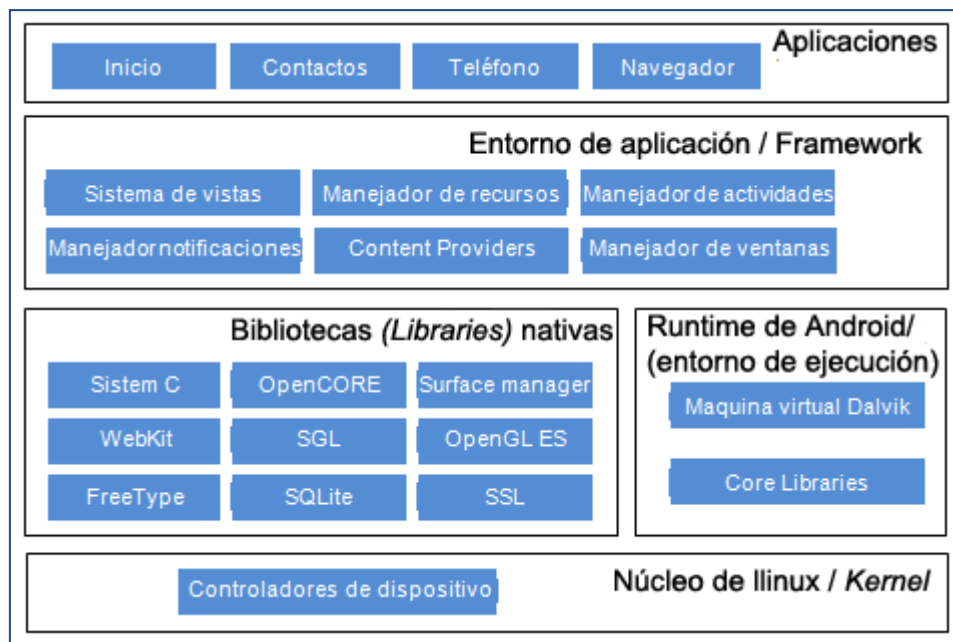


Figura 5.2 Arquitectura SO Android

5.1.3.1 El núcleo Linux

El núcleo de Android está basado en el sistema operativo Linux v2.6. Esta capa proporciona servicios como la seguridad, gestión de memoria, multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

También actúa como una capa de abstracción entre el *hardware* y el resto de las capas. Es la única que es dependiente del *hardware*. De esa manera evitamos el tener que conocer las características de cada dispositivo. Por ejemplo, si necesitamos utilizar la cámara, el sistema operativo se encarga de usar la que tiene el teléfono, siendo transparente para la aplicación.

El núcleo de Android también se suele llamar como *kernel*. En la mayoría de dispositivos podéis consultar la versión, mirando en *ajustes/ acerca del dispositivo/ Versión del kernel*.

5.1.3.2 Bibliotecas (Libraries)

Esta capa está situada justo por encima del núcleo (*kernel*), y está formada por las bibliotecas nativas de Android. Están creadas en lenguaje C o C++ y complementadas por la estructura *hardware* específica del dispositivo.

Estas bibliotecas suelen estar hechas por el fabricante, que también se encarga de instalarlas al dispositivo. El objetivo principal de estas bibliotecas es proporcionar funcionalidad a las aplicaciones que se realizan con bastante frecuencia.

Entre las bibliotecas incluidas habitualmente se encuentran **OpenGL** (se encarga de los gráficos), **WebKit** (navegador), **SSL** (cifrar las comunicaciones), **FreeType** (Fuentes de texto), **SQLite** (Base de datos), entre otras.

5.1.3.3 Runtime Android

También conocidos como el entorno de ejecución no está considerado como una capa por sí sola, se encuentra al mismo nivel que las librerías nativas de Android. Aquí encontramos librerías con funcionalidad habitual de Java, como otras específicas de Android (*Core libraries*).

El componente principal del entorno de ejecución es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y se compilan de una forma específica para ser ejecutadas por esta máquina virtual. La principal ventaja es que la aplicación sólo se compila una única vez, y se puede garantizar que se podrá ejecutar en cualquier dispositivo que tenga la versión mínima del sistema operativo.

Es importante recalcar que Java es únicamente el lenguaje de programación, los ejecutables los genera el SDK de Android y tiene la extensión *.dex* que es específica para Dalvik, por este motivo no se pueden ejecutar aplicaciones de Java en Android ni viceversa.

5.1.3.4 Entorno de la aplicación / Framework

Capa formada por todas las clases y servicios que utilizan directamente las aplicaciones. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos de capas anteriores a partir de la máquina virtual Dalvik. En esta capa podemos encontrar:

- **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estados.
- **Resource Manager**: proporciona acceso a recursos que no están en el código, es decir, cadenas de texto traducidas, imágenes, sonidos.
- **Views**: Elementos que ayudan a la construcción de la interfaz del usuario, como es el caso de botones, cuadros de texto, listas, etc.
- **Content Providers**: Mecanismo sencillo para acceder a datos de otras aplicaciones.

5.1.3.5 Aplicaciones

Es la última capa que engloba todas las aplicaciones del dispositivo, tanto las que tiene interfaz gráfica como las que no la tienen, las nativas (programadas en C o C++) y las administradas (programadas en Java), las preinstaladas y las que el usuario instala.

5.1.4 Entorno de desarrollo

Para el desarrollo del proyecto vamos a poder utilizar un potente y moderno entorno de desarrollo. Al igual que Android todas sus herramientas están basadas en el *software* libre. Aunque existen varias alternativas para el desarrollo aplicaciones. En este proyecto vamos a utilizar el *software* enumerado a continuación:

Android Studio, entorno de desarrollo integrado (IDE) para la plataforma Android. Está disponible para descargar en Windows, Mac OS X y GNU/Linux.

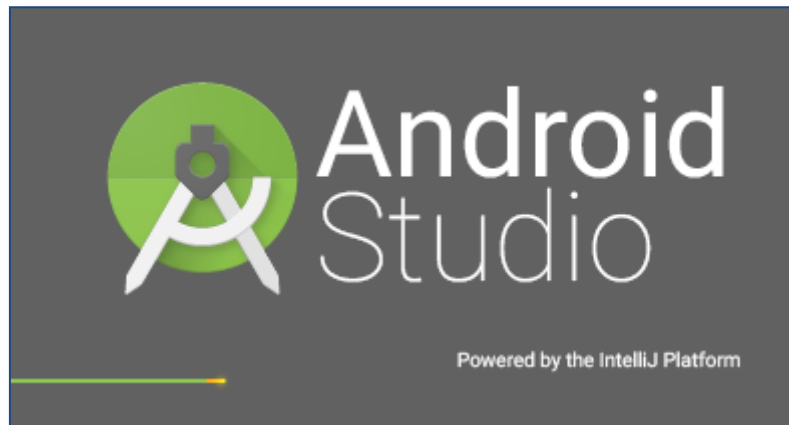


Figura 5.3 Software Android Studio

Deberemos comprobar la versión del Java SE Development Kit (JDK), una vez instalada procederemos a la instalación del SDK.

Instalar el SDK, *Software Development Kit*, es imprescindible, ya que este contiene herramientas para:

- Creación de proyectos.
- Compilación.
- Emulación.
- Depuración.

El entorno también dispone de un emulador Android, cuando la aplicación esté en fase de pruebas, se puede comprobar si el funcionamiento es el esperado. También se puede simular la aplicación en un dispositivo externo. Colocando éste en modo “*Opciones de desarrollo y depuración USB*”. En nuestro caso utilizamos la segunda opción y comprobamos el funcionamiento de la aplicación en dos dispositivos, *Samsung Galaxy S4* y *Tablet ASUS Pad7*.

5.1.5 *Conceptos básicos*

En este apartado voy a hacer un pequeño repaso por los conceptos más relevantes de la programación Android.

- **Actividad (*Activity*):** Parte de la funcionalidad de una aplicación, esta debe tener asignada una interfaz gráfica de usuario. (Toda interfaz necesita de una actividad que la llame).
- **Ciclo de vida de una actividad:** Las actividades se almacenan en una pila. Cuando una nueva actividad es ejecutada es pone la primera de la pila. Las actividades anteriores no se ejecutan hasta que esta quede cerrada.

Este ciclo tiene 4 estados:

1. **Ejecución (*running*):** Si la actividad está en pantalla.

2. **Pausa (*paused*):** Si otra actividad se encuentra delante. Una actividad pausada está totalmente viva, pero puede ser eliminada por el sistema.
3. **Parada (*stopped*):** Si otra actividad la tapa por completo encara que conserva su estado e información, pero no es visible para el usuario.
4. **Finalizada (*finished*):** Si una actividad está en pausa o parada el sistema la puede eliminar cerrando el proceso.

En la siguiente figura se pueden ver los cuatro estados que pueden tener. También incluye los métodos que deberíamos re-programar para realizar los cambios de estados.

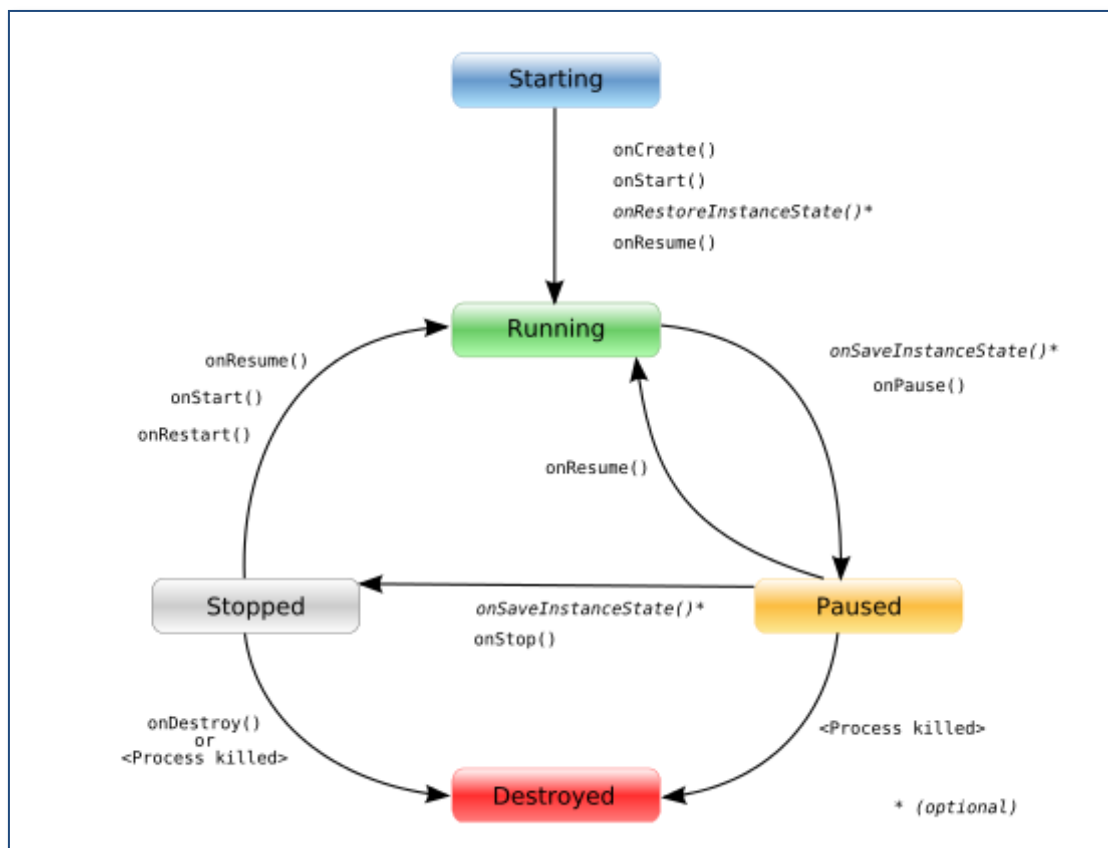


Figura 5.4 Ciclo de vida de Actividad

- **Service:** Comúnmente se conoce como proceso. Son actividades que no tienen un interfaz visual asignado. En nuestro caso lo hemos utilizado para mantener la comprobación del enlace realizado con la antena Bluetooth del Arduino.
- **Intents:** Mecanismo asíncrono, sirve para poder comunicar varias aplicaciones o actividades. También para notificar advenimientos del sistema, como puede ser la conexión del cable USB.
- **Layout:** Es el recurso con el que puedes describir lo que quieres mostrar por pantalla y como lo quieres mostrar. La manera más común de crearlo es a través de un archivo de descripción de interfaces de tipo `.xml`.

5.2 Estructura del proyecto

En este apartado vamos a conocer la estructura de un proyecto Android, es importante saber dónde encontrar y guardar cada elemento. Este puede tener pequeñas variaciones dependiendo de las características del proyecto, o del entorno de desarrollo que utilices.

Aunque lo que realmente vamos a ver no es puramente la estructura del proyecto, sino el árbol de proyecto que nos muestra el Android Studio, cuando en la pestaña superior tenemos seleccionado “Android”, si seleccionáramos “Project” nos mostraría la estructura que podremos encontrar en cualquier libro de la documentación. Pero nosotros hemos trabajado en la visualización que vamos a explicar.

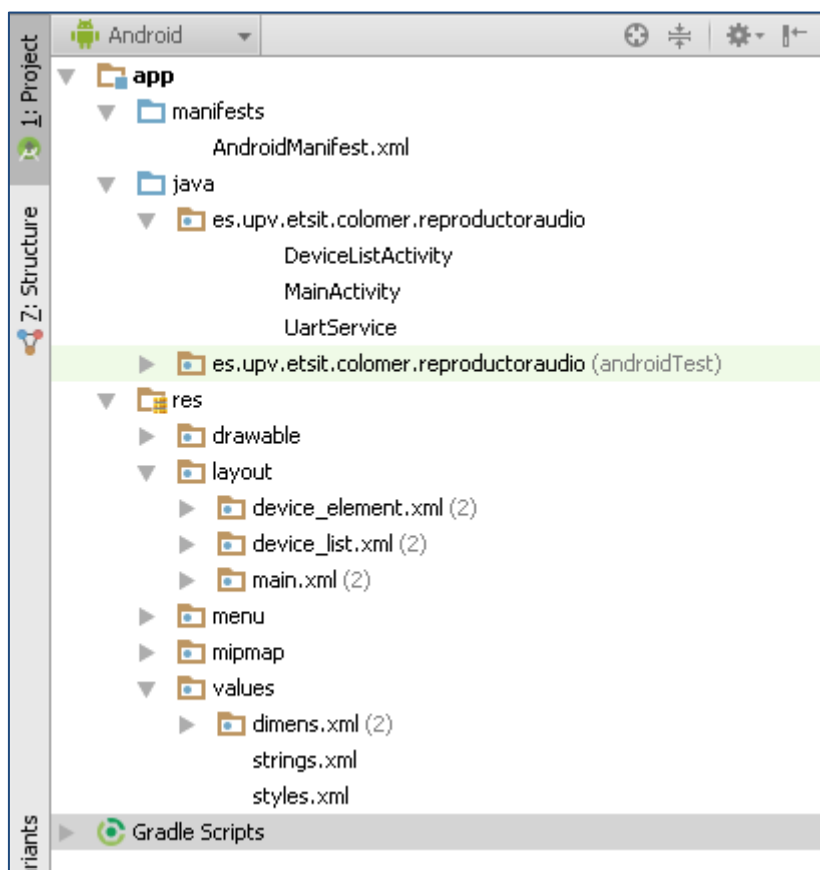


Figura 5.5 Árbol del proyecto

Lo primero que encontramos es la carpeta que contiene el *manifest*, este contiene el número de versión y permisos que requiere la aplicación, entre otras cosas. En el apartado 5.3 AndroidManifest entraremos con más detalle sobre este.

A continuación en la carpeta **java**, se ubican todos los archivos *.java*, contienen las funciones donde se programa lo que realmente hace la aplicación. En el apartado 5.5 *Clases - Actividades y Servicios* se explica de qué funciones está hecha nuestra aplicación.

La carpeta que viene a continuación es *res/* en esta se encuentran todos los recursos.

- **Drawable:** Esta carpeta contiene todas las imágenes que van a mostrarse en los *layouts*.
- **Layout:** Aquí se ubican las diferentes pantallas que forman la interfaz visual de la aplicación. Esto se explica con más detalle en el apartado 5.4 *Interfaz gráfica*

- **Menú:** Aquí se configura si hay algún menú de herramientas de la aplicación, en nuestro caso no tenemos.
- **Mipmap:** Este es el repositorio donde se ubican los diferentes logotipos de la aplicación para sus diferentes versiones de tamaño.
- **Values:** Dentro de esta se encuentra las dimensiones de las diferentes configuraciones de pantalla, el documento de *Strings* y de *styles*.

El documento de *strings*, es especialmente útil, ya que aquí puedes agrupar todos los *TextView* que van a salir por los diferentes *layouts*, realizando un cambio en este documento se cambiarían en todos los sitios donde hace referencia ese *String*.

También te permite que crees un fichero diferente para cada idioma, entonces la aplicación detectaría el idioma del dispositivo, y mostraría los *Strings* del idioma que correspondiera.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2
3  <resources>
4
5      <string name="app_name">Reproductor Arduino</string>
6      <string name="device">Dispositivo:</string>
7      <string name="no_device">&lt;Selecciona el dispositivo&gt;</string>
8      <string name="select_device">Selecciona el dispositivo</string>
9      <string name="status">Status</string>

```

Figura 5.6 Fichero de Strings

El último fichero que encontramos en este árbol del proyecto es de estilos, donde se configura el tema (en el ámbito de colores y tonalidades) de la aplicación.

5.3 AndroidManifest

Este archivo es uno de los más importantes de la aplicación. Es un documento *.xml* donde se declaran los elementos de la aplicación, así como sus restricciones, permisos, procesos e interacciones con otras aplicaciones. Este lo genera el SDK a partir de la información que se ha proporcionado al crear el proyecto. Después puedes añadir más permisos, para el correcto funcionamiento de la aplicación.

Estas son las principales partes que engloba este documento:

- La versión de la aplicación, para poder realizar actualizaciones al *Android Market*.
- La versión mínima del SO de Android, a partir de la que se puede utilizar la App.
- Requisitos *Hardware* de la aplicación.
- Permisos para utilizar diferentes componentes, por ejemplo Bluetooth.
- Declaración de actividades.


```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="es.upv.etsit.colomer.reproductoraudio"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7
8     <uses-sdk...>
11    <uses-permission android:name="android.permission.BLUETOOTH" />
12    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
13    <uses-permission android:name="android.permission.VIBRATE" />
14
15    <application...>
34
35 </manifest>
36
```

Figura 5.7 Estructura del Manifest

En los siguientes apartados paso a describir lo que se encuentra en nuestro *Manifest*.

5.3.1 Declaración Manifest

La primera etiqueta que nos encontramos es `<manifest>`, esta es la etiqueta principal, ya que engloba todo lo que contiene este *.xml*, esta se cierra al final del documento.

El primer atributo que encontramos es el `xmlns:android`, este define el espacio de nombres de Android y siempre tiene que ser `"http://schemas.android.com/apk/res/android"`.

A continuación se encuentra definido el `package`, este es el nombre completo del directorio de carpetas para la aplicación. Este nombre siempre debe de ser único, por eso como norma general los programadores suelen escribirlo como si fuera un dominio de internet pero a la inversa. En nuestro caso he utilizado `"es.upv.etsit.colomer.reproductoraudio"`.

También se encuentra el `versionCode`, este indica el número de versión de la aplicación. Este número es elegido por el programador, y se utiliza para cuando saques una nueva versión, lo incrementas y así el usuario puede detectar que existe una versión superior a la que tiene instalada. Esto va enlazado con un `versionName` que no es más que el nombre de la versión explicada anteriormente.

En nuestro caso hemos asignado como numero de versión 1, ya que es la primera versión de esta aplicación, si sacáramos alguna actualización en el futuro iríamos incrementado como corresponda.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="es.upv.etsit.colomer.reproductoraudio"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
```

Figura 5.8 Manifest versión

5.3.2 Versión mínima y permisos

En la etiqueta `<uses-sdk/>`, se puede observar como la versión mínima requerida para la aplicación es la API 18, es decir, la versión 4.3 del SO Android. Se necesita de esta, porque es la versión más baja que engloba librerías y funciones para el manejo de la antena BLE (*Bluetooth Low Energy*), la cual se usa en el dispositivo Arduino.

En cuanto a los permisos, la aplicación requiere de permisos para manejar la configuración Bluetooth, y también permisos de vibración para generar un efecto al pulsar los botones.

```
7
8   <uses-sdk
9       android:minSdkVersion="18"
10      android:targetSdkVersion="18" />
11   <uses-permission android:name="android.permission.BLUETOOTH" />
12   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
13   <uses-permission android:name="android.permission.VIBRATE" />
14
```

Figura 5.9 Manifest sdk/permission

5.3.3 Application

En la etiqueta de `<application/>`, se declara la aplicación, con todos sus subelementos y actividades que componen la aplicación.

Los tres primeros atributos son comunes a todos: *label* (etiqueta) pone el nombre a la app, *theme* configura el tema que se va a usar en nuestro caso “*Theme.Holo.Light*”, e icono que enlaza donde este el logo creado para la aplicación.

```
14
15   <application
16       android:label="@string/app_name"
17       android:theme="@android:style/Theme.Holo.Light"
18       android:icon="@mipmap/logo_app">
19
20       <activity...>
34
35       <activity android:name=".DeviceListActivity" android:label="@string/app_name"
36           android:theme="@android:style/Theme.Dialog" />
37
38       <service android:enabled="true" android:name=".UartService" />
39
40   </application>
41
42 </manifest>
43
```

Figura 5.10 Manifest Application

5.3.4 Activity y Service

Dentro de la etiqueta de *application* deben estar todas las *activities* y los *services* que tiene que ejecutar la aplicación. La principal diferencia entre un *activity* y un *service* es que las *activities* tienen interfaz de usuario y los *services* no. Ahora vamos a pasar a explicar que atributos contienen cada uno de ellos.

El *activity* declara una actividad, la cual tendrá que tener parte de interfaz visual del usuario, cambios del *layout*, aparezca una ventana emergente, etc. Todas las actividades que hace la aplicación deben de estar reflejadas aquí. Cualquiera que no se declare no será visto por el sistema, y nunca se ejecutará.

Vamos a pasar a ver los atributos que hemos definido para cada una de las actividades:

El primer atributo de nuestras dos actividades es *name*, el cual va precedido del nombre de la clase que implementa la actividad. Para nuestra actividad principal el nombre completo es “*es.upv.etsit.colomer.reproductoraudio.MainActivity*”, pero podemos poner el nombre abreviado “*MainActivity*”. La otra actividad que tenemos en nuestro proyecto es “*DeviceListActivity*”. Esta última es la que se encarga de buscar y generar el listado de dispositivos Bluetooth disponibles. En el apartado correspondiente explicaremos el funcionamiento de ambas *activities*.

El segundo atributo es la etiqueta, en nuestro caso el nombre de la App, el cual está en el fichero *Strings*, y le hacemos referencia a él “*@String /app_name*”.

En siguiente atributo de la actividad principal “*windowSoftInputMode*” es la que define la gestión del teclado virtual en pantalla, si está presente o se oculta, controla el tamaño, etc. En nuestro caso hemos decidido que el teclado este oculto, para eso el atributo debe tener este valor “*stateHidden*”.

También hemos tomado la decisión que la interfaz del usuario siempre este en posición vertical, por eso el siguiente atributo “*screenOrientation*” tiene como valor “*portrait*”. Si quisiéramos que siempre estuviera en horizontal pondríamos “*landscape*”.

Dentro de la actividad principal, también tenemos la etiqueta *<intent-filter>* esta se encarga de definir la intención de dicha actividad. En el *action /name* que sirve para añadir una acción al listado de intenciones. En nuestro caso para la acción *ACTION_MAIN* basta con poner “*android.intent.action.MAIN*”, con esto indicamos que esta actividad debe añadirse a la entrada principal. A continuación definimos la categoría de la intención, “*android.intent.category.LAUNCHER*”, con esto se indica que se debe añadir al lanzador de actividades.

En cuanto a la segunda actividad, “*DeviceListActivity*” tiene sólo los dos primeros atributos que hemos comentado anteriormente, y el atributo del tema, que elegimos “*style/Theme.Dialog*”, ya que va a ser una ventana emergente que salga.

Por último, tenemos un *service*, que como hemos dicho anteriormente es lo mismo que una *activity* pero sin tener interfaz visual. Sus dos únicos atributos son, *enabled* que está a “*true*” para indicar que el servicio está habilitado, y el nombre, que es “*es.upv.etsit.colomer.reproductoraudio.UartService*”

```

19
20 <activity
21     android:name=".MainActivity"
22     android:label="@string/app_name"
23     android:windowSoftInputMode="stateHidden"
24     android:screenOrientation="portrait">
25
26     <intent-filter>
27         <action android:name="android.intent.action.MAIN" />
28
29         <category android:name="android.intent.category.LAUNCHER" />
30     </intent-filter>
31
32 </activity>
33
34 <activity android:name=".DeviceListActivity" android:label="@string/app_name"
35     android:theme="@android:style/Theme.Dialog" />
36
37 <service android:enabled="true" android:name=".UartService" />
38

```

Figura 5.11 Manifest Activity/Service

5.4 Interfaz gráfica

La interfaz de usuario es la principal sección de interacción entre la persona y el dispositivo. A todas las funcionalidades disponibles se accede a través de la pantalla. Es muy importante conseguir que el manejo sea intuitivo y sencillo.

Para generar las pantallas de interfaz visual. Se construyen ficheros nombrados *Layouts* a través de nodos *.xml*. Aunque tenemos una ayuda extra ya que nuestro entorno está dotado de un panel de diseño del estilo arrastrar y soltar (*Drag and Drop*), lo cual facilita mucho el diseño de estas.

En este modo de diseño, la creación de pantallas consiste en coger del panel de la izquierda el componente que quieres, como puede ser un *TextView*, *Button*, *ImageView*, etc. Lo arrastras hasta situarlo sobre el sitio de la pantalla que deseas. Y una vez en la pantalla puedes retocar las propiedades del componente en el cuadro que sale bajo a la derecha.

Todo lo que hacemos en esta pantalla de manera visual, se convierte en texto *.xml* en la configuración del *layout*. Para poder ver o modificar el texto que has producido debes conmutar la pestaña que hay bajo de *Design* a *Text*.

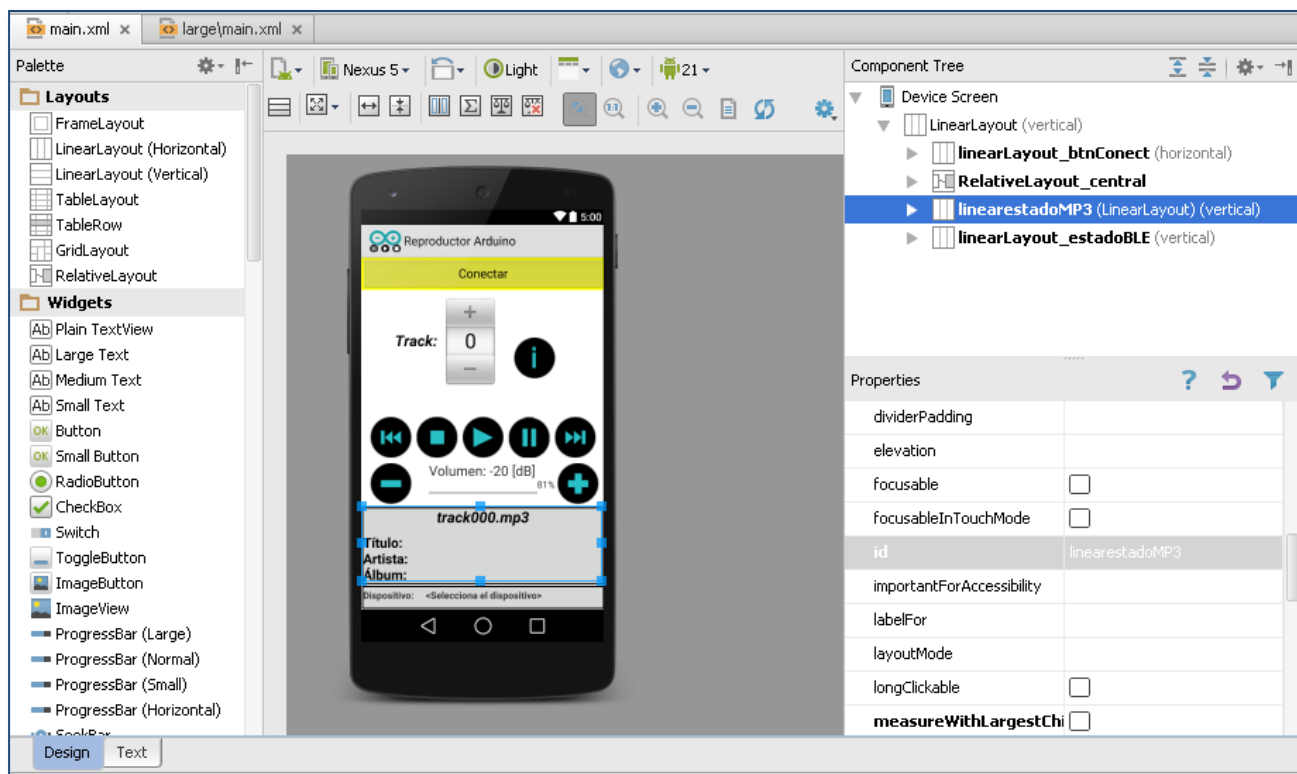


Figura 5.12 Software de Diseño del Layout

5.4.1 Layouts

En este apartado pasé a describir las principales características que hay que tener en cuenta para la creación de diferentes *layouts*.

Como he dicho anteriormente, es un recurso que describe lo que quieres mostrar en pantalla, y como lo quieres mostrar. La manera más común de crearlos es mediante un archivo *.xml* (en el directorio *res/layout* del proyecto). Tiene un formato muy similar al de HTML, siempre sigue el patrón:

```
<Nombre_del_layout atributo1="valor1"
    atributo2="valor2"
    [...]
    atributoN="valorN">
</Nombre_del_layout>
```

Todos los *layout* tienen unos parámetros de configuración. Hay dos que son comunes para todos *layout*: *height* y *width*, que sirven para especificar el valor de la altura y anchura. Estos atributos pueden recibir un valor numérico, o bien *wrap_content* se ajusta a las dimensiones del contenido, *fill_parent* será tan grande como su límite superior lo permita.

Cada elemento tiene sus propios atributos, entre ellos está el *id*, se encarga de distinguir ese elemento del resto, otorgándole un nombre único. Por ejemplo, `android:id="@+id/linearLayout2"`.

En nuestro caso hemos decidido dividir los *layouts* en diferentes secciones como pueden ser *LinearLayout* o *RelativeLayout*, que son estructuras para englobar todos los elementos que vayamos a introducir. Como muestra la siguiente figura, hemos dividido el *Layout* Principal en 4 secciones.

- **LinearLayout_btnConect.** Corresponde al botón superior.
- **RelativeLayout_central:** Corresponde a el conjunto de botones que hay en el centro de la pantalla
- **LinearestadoMP3:** El bloque donde salen reflejado el título, el artista y el álbum.
- **LinearLayout_estadoBLE:** Donde se informa del estado de la conexión Bluetooth.

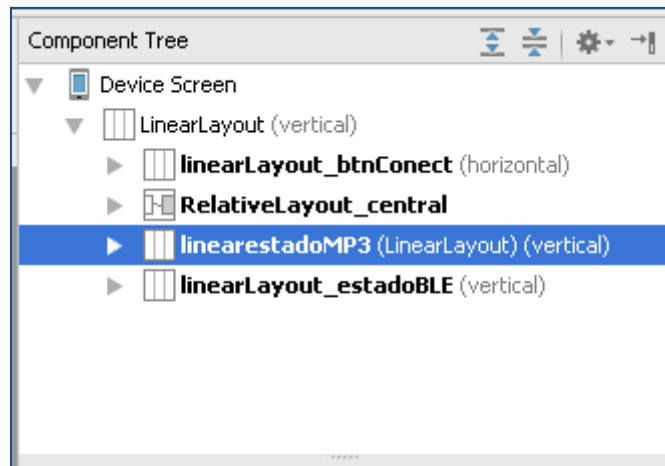


Figura 5.13 Partes Layout principal

En la siguiente figura podemos observar los diferentes atributos que contiene nuestro primer *LinearLayout*:

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      android:orientation="vertical" >
6
7      <LinearLayout
8          android:id="@+id/linearLayout_btnConect"
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:orientation="horizontal"
12         android:background="#ffffff00">
13
14         <Button
15             android:id="@+id/btn_select"
16             android:layout_width="match_parent"
17             android:layout_height="wrap_content"
18             android:padding="12dp"
19             android:text="Conectar" />
20     </LinearLayout>
21

```

Figura 5.14 *LinearLayout*

El resto de elementos siguen la misma estructura pero con atributos específicos para cada elemento. Los elementos que más hemos usado son los *ImageButton*, para los ocho botones que gestionan la reproducción del audio. También bastantes *TextView* para mostrar todos los textos de la pantalla. Un *progressBar* para mostrar el estado del volumen. Y por último un *NumberPicker* para elegir el número de canción.

5.4.2 Nuestros Layouts

Como hemos probado el funcionamiento en dos dispositivos diferentes y con pantallas de diferentes dimensiones, hemos duplicado cada uno de los layouts, con el layout normal que será el que por defecto arrancara en los dispositivos cercanos a 4 pulgadas y otro con la versión *layout-large* que lo utilizarán los que contenga 7 pulgadas. En las siguientes figuras se pueden ver los dos *layouts* principales.

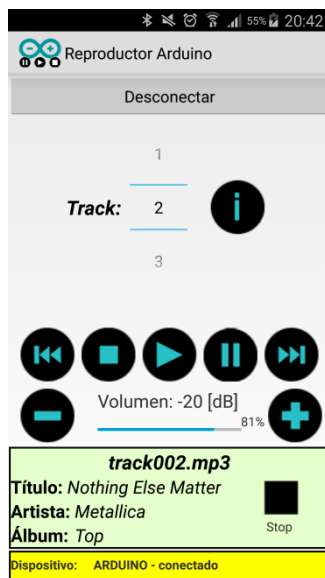


Figura 5.15 Layout 4 pulgadas

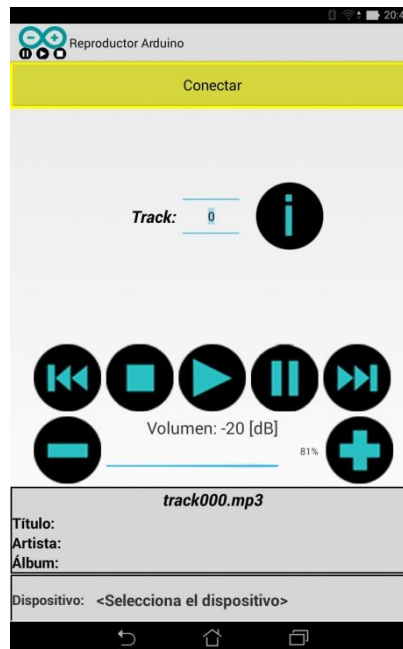


Figura 5.16 Layout 7 pulgadas

Nuestra aplicación consta de tres *layouts* diferentes, cada uno de ellos sus dos versiones para las diferentes dimensiones de pantalla, como se puede ver en la siguiente figura.

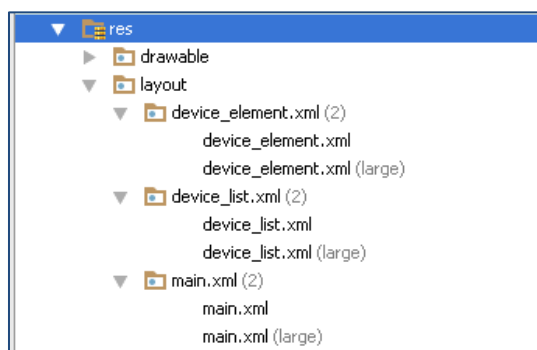


Figura 5.17 Estructura todos Layouts

Ahora paso a mostrar las características de los diferentes *layouts*:

- **Layout Principal (*main*):**

Este es el *Layout* principal, es el *Layout* base sobre el cual se ejecuta la aplicación, se muestra este nada más arrancara esta.

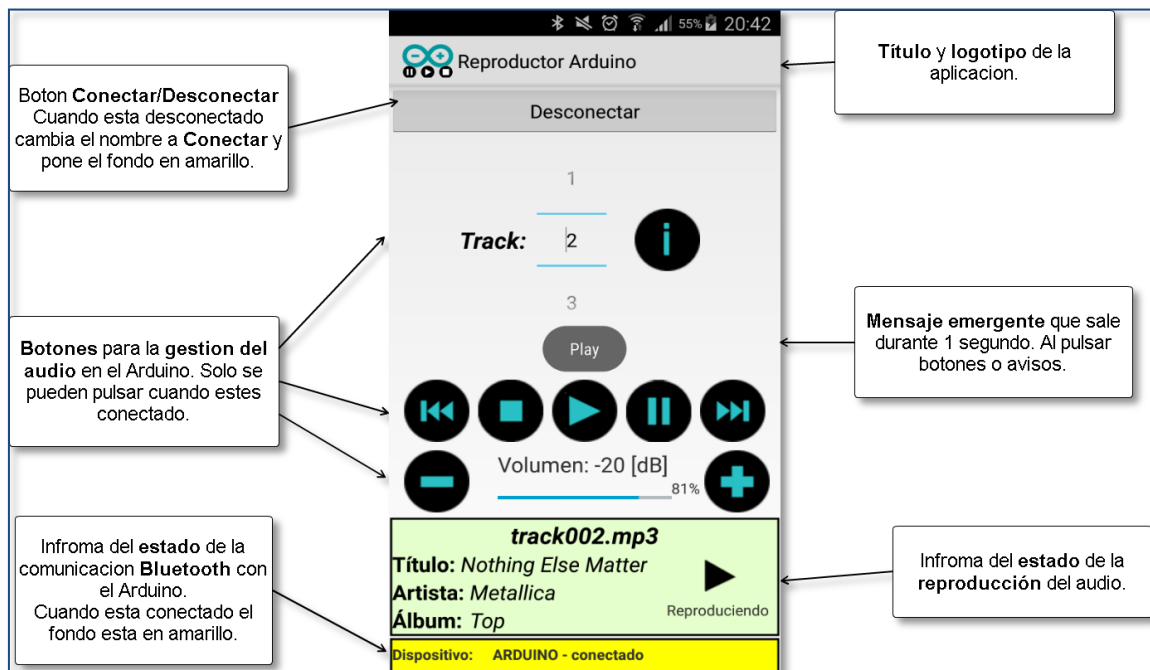


Figura 5.18 Main *Layout*

- **Layout *device_list* y *device_element***

Estos dos van unidos y el código del programa salta cuando se pulsa el botón de conectar. Estos generan la lista de los dispositivos Bluetooth a los que puede conectarse la aplicación para que puedas elegir. Cuando seleccionas uno vuelve al *layout* principal.

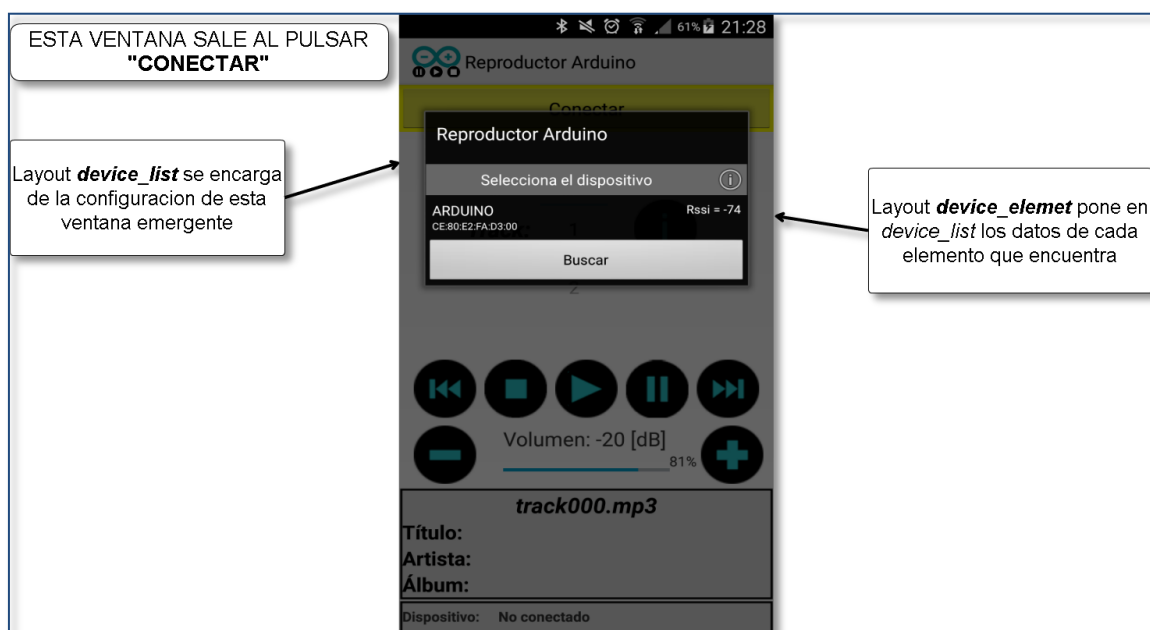


Figura 5.19 *Device Layout*

5.5 Clases - Actividades y Servicios

La parte lógica de las Actividades y los servicios son los archivos .java, estos nombrados como clases (*class*), se crean para poder manipular, interactuar y colocar el código de esa actividad o servicio.

5.5.1 Estructura de una Clase

En este punto vamos a introducir lo que obligatoriamente tiene que contener las clases.

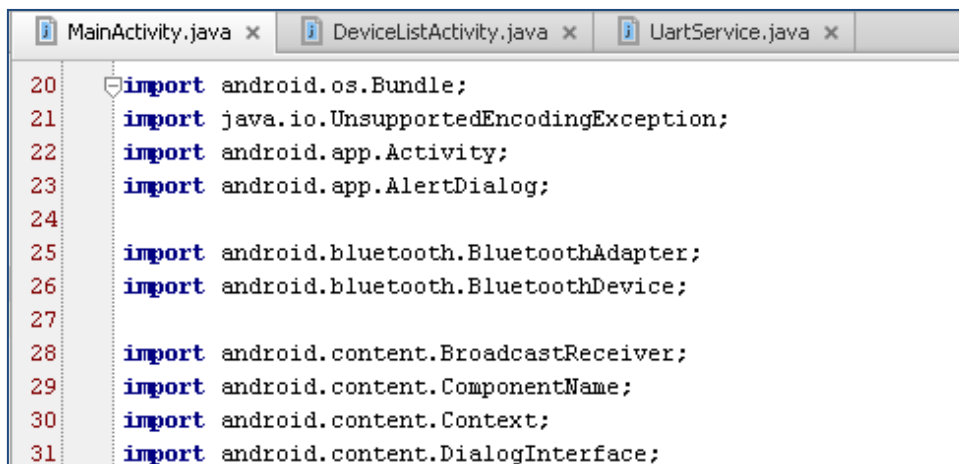
- Lo primero que debemos ubicar es el **package**, este es el directorio al cual pertenece esa clase y allí se encuentran todos los archivos que pertenecen a ese paquete, es decir, otras clases que interactúan en esta aplicación. En nuestro caso: *package es.upv.etsit.colomer.reproductoraudio*. Que contiene tres clases.

```
15 package es.upv.etsit.colomer.reproductoraudio;
16
```

Figura 5.20 Llamada a package

- A continuación vendrían todos los **imports**, con esta instrucción importamos la clase que se nombra en la declaración. Estas clases son bibliotecas creadas por Android.

Su importación sigue el siguiente formato:



```
MainActivity.java x DeviceListActivity.java x UartService.java x
20 import android.os.Bundle;
21 import java.io.UnsupportedEncodingException;
22 import android.app.Activity;
23 import android.app.AlertDialog;
24
25 import android.bluetooth.BluetoothAdapter;
26 import android.bluetooth.BluetoothDevice;
27
28 import android.content.BroadcastReceiver;
29 import android.content.ComponentName;
30 import android.content.Context;
31 import android.content.DialogInterface;
```

Figura 5.21 Imports

- El siguiente punto es donde se **declara la clase** que vas a definir en ese documento. En el caso que sea una actividad deberá extender de la clase *Activity*, esta nos la proporciona Android para que crees tus actividades sobrescribiendo (*override*) sus métodos.

Esto ocurre en nuestra clase principal, y en la actividad que se encarga de buscar dispositivos. (*MainActivity.java* y *DeviceListActivity.java*)

```
61
62 public class MainActivity extends Activity
63
```

Figura 5.22 Declaración Class MainActivity

```

52
53 public class DeviceListActivity extends Activity {
54

```

Figura 5.23 Declaración Class DeviceListActivity

En cuanto a los servicios, también extiende de una clase que nos proporciona Android, pero en este caso es la clase *service*. En nuestra aplicación tenemos un servicio que es el que se encarga de comprobar que mantiene la conexión Bluetooth establecida.

```

41 public class UartService extends Service {
42

```

Figura 5.24 Declaración Class UartService

- Una vez declarada la clase lo que corresponde a continuación es la declaración de las variables. Estas pueden ser de diferentes tipos, y tener diferente campo de aplicación.

Tipos de variables:

Tipo	Tamaño (bits)	Mínimo	mín ·	Máximo	Wrapper (Envoltorio)
byte	8	-128	0	127	Byte
short	16	$-32\,768 = -2^{15}$	0	$32\,767 = 2^{15} - 1$	Short
int	32	$-2\,147\,483\,648 = -2^{31}$	0	$2\,147\,483\,647 = 2^{31} - 1$	Integer
long	64	$-9\,223\,372\,036\,854\,775\,808 = -2^{63}$	0	$9\,223\,372\,036\,854\,775\,807 = 2^{63} - 1$	Long
float	32	$-3,4 \cdot 10^{38}$	$1,4 \cdot 10^{-45}$	$3,4 \cdot 10^{38}$	Float
double	64	$-1,8 \cdot 10^{308}$	$4,9 \cdot 10^{-324}$	$1,8 \cdot 10^{308}$	Double
boolean	?	false	-	true	Boolean
char	16	Unicode 0 = '\u0000'	-	Unicode 65 535 = '\uffff' = $2^{16} - 1$	Character
void	-	-	-	-	Void

Declaración

```
<tipo> <nombre> [ = <valor> ] ;
```

Ejemplo

```
double x = 9.7;
int j;
```

Tabla 5.2 Tipos de variables

Los diferentes campos de aplicación, se marca con la etiqueta que llevan delante. Se llaman la visibilidad o modificadores de acceso y pueden ser de estos diferentes tipos:

```

visibilidad: [ ] | public | protected | private
modificadores: [ ] | final | static | abstract

```

hline	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
<code>public</code>	X	X	X	X
<code>protected</code>	X	X	X	
<code>[]</code>	X	X		
<code>private</code>	X			

Tabla 5.3. Visibilidad

Visibilidad:

La visibilidad modifica tanto variables como métodos y clases, dependiendo de a lo que acompañe.

- `[]` o ningún modificador (defecto): el elemento puede ser accedido sólo desde las clases que pertenecen al paquete.
- **public**: se puede acceder desde cualquier clase.
- **private**: sólo se puede acceder desde la propia clase (se puede acceder a miembros privados de otros objetos de la misma clase).
- **protected**: sólo se puede acceder desde las clases que pertenecen al paquete y por cualquier clase que extienda a ésta.

Modificadores:

Pueden afectar tanto a variables como a métodos y clases, en este punto veremos lo que afecta a las variables.

- **static**: Existen independientemente de las instancias (no es necesario instanciar un objeto). Existe sólo una copia de la variable estática entre todos los objetos.
 - **final**: ya no podrá modificarse su valor. Existe una sola copia de la variable entre todos los objetos.
 - **abstract**: este a las variables no afecta.
- Todo seguido vendría en conjunto de métodos que componen la clase, tanto los estándares sobrescritos de *Activity* como puede ser: *onCreate*, *onDestroy*, *onPause*, etc, como los definidos por el programador para realizar diferentes funciones.

Ahora pasamos a describir la funcionalidad de las clases que componen nuestra aplicación:

5.5.2 Class MainActivity

Esta es la clase principal, la cual gestiona el funcionamiento general de la aplicación y realiza las llamadas a las otras clases cuando lo requiere.

En esta se gestiona toda la interfaz de la pantalla principal, y lo que tiene que realizar cuando pulsas los botones para enviar comandos, y también cuando recibe la información de la canción que está reproduciendo, o del volumen que contiene el dispositivo.

En este apartado vamos a resaltar los métodos más importantes de esta clase, aunque en el siguiente enlace, se adjunta el código fuente completo con los respectivos comentarios.

<https://goo.gl/FUOpGE>

- Declaración del método *onCreate* y llamamiento al *layout* principal:

```
63      @Override
64      public void onCreate(Bundle savedInstanceState) {
65          super.onCreate(savedInstanceState);
66          setContentView(R.layout.main);
67      }
```

Figura 5.25 Declaración *onCreate*

- Método que tiene que ejecutar cuando se pulsa el botón de “Conectar”. Si el Bluetooth esta encendido y disponible (*mBtAdapter.isEnabled()*) entonces **llama a la clase *DeviceListActivity.class***.

```
92
93      // Handler Disconnect & Connect button
94      btnConnectDisconnect.setOnClickListener(new View.OnClickListener() {
95          @Override
96          public void onClick(View v) {
97              if (!mBtAdapter.isEnabled()) {
98                  Log.i(TAG, "onClick - BT no habilitado todavia");
99                  Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
100                 startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
101             }
102             else {
103                 if (btnConnectDisconnect.getText().equals("Conectar")){
104                     //LLAMADA A 'DEVICE LIST ACTIVITY'
105                     Intent newIntent = new Intent(MainActivity.this, DeviceListActivity.class);
106                     startActivityForResult(newIntent, REQUEST_SELECT_DEVICE);
107                 } else {
108                     if (mDevice!=null){
109                         mService.disconnect(); //Desconectamos
110                     }
111                 }
112             }
113         }
114     });
115 }
```

Figura 5.26 *onClick* Conectar

- A continuación vienen el conjunto de botones que tiene la aplicación, (play, pause, stop, siguiente, anterior, subir volumen, bajar volumen, información).

Voy a explicar el funcionamiento del botón “play”, el resto de botones son similares a éste.

Cuando pulsas el botón, hace el efecto de vibración y muestra un *toast*, llamando a la función *showMessage(“Play”)*, esta función está programada por nosotros al final de esta clase.

Todo seguido lee el número que hay en el *NumberPicker*, y compara para ver si es diferente al número que tenía anteriormente guardado, si no es diferente no ejecuta nada más, y si es diferente lo convierte a *String* para poder enviarlo, añade una “F” al final del *String*, para que el Arduino identifique donde está el final del número de canción. Actualiza el *TextView* que hace referencia a la canción que se está reproduciendo.

Convierte el mensaje a enviar a formato ASCII (UTF-8), y lo envía mediante la función *mService.writeRXCharacteristic(String)*. Activa el booleano que indica que el siguiente *String* que va a recibir del Arduino es el título y después actualiza todos los *View* que hacen referencia al estado de reproducción.

```
156 btnplay.setOnClickListener(onClick(v) -> {
161     showMessage("Play"); //llamada al toast
162     Vibrator vl = (Vibrator) getSystemService(VIBRATOR_SERVICE); //vibracion
163     vl.vibrate(tempVib);
164     NumberPicker np2 = (NumberPicker) findViewById(R.id.numberPicker2);
165     track = np2.getValue();
166
167     if(last_track != track) {
168         last_track = track;
169         String s_track = Integer.toString(track);
170         s_track = s_track + "F";
171         convertir(track); //calcula n1, n2 y n3
172         ((TextView) findViewById(R.id.track00)).setText("track" + n1 + n2 + n3 + ".mp3");
173
174         byte[] valuel;
175         try {
176             valuel = s_track.getBytes("UTF-8"); //Envia el message
177             mService.writeRXCharacteristic(valuel);
178             rec_titulo = true; //Activa booleano para la recepción
179             rec_artista = false; rec_album = false; rec_vol = false;
180
181         } catch (UnsupportedEncodingException e) {
182             e.printStackTrace(); // TODO Auto-generated catch block
183         }
184         last_track = track;
185         ImageView imageView = (ImageView) findViewById(R.id.status);
186         imageView.setImageResource(R.drawable.estatus_play);
187         ((TextView) findViewById(R.id.text_status)).setText("Reproduciendo");
188         RelativeLayout RLayout_estado = (RelativeLayout) findViewById(R.id.RelativeLayout_estadoMP3);
189         RLayout_estado.setBackgroundColor(0xFFE5FFCC);
190     }
191 };
```

Figura 5.27 onClick Play

- Parte de recepción de Strings del Arduino.

Si hay datos de recepción que nos lo indica el servicio *UartService*. Escribimos en *TxValue* lo que recibimos en formato ASCII UTF-8.

Dependiendo del booleano que tengamos activado entraremos a un caso u otro, en la figura que tenemos a continuación se corresponde a cuando nos acabamos de conectar se activa el *rec_total*, éste indica que lo que vamos a recibir es el número total de canciones que disponemos en la tarjeta μ SD. Por lo cual el número máximo que debemos poner en el *NumberPicker*

```

494 //TIENE DATOS DE LA ANTENA PARA INTERPRETAR
495 if (action.equals(UartService.ACTION_DATA_AVAILABLE)) {
496
497     final byte[] txValue = intent.getByteArrayExtra(UartService.EXTRA_DATA);
498     runOnUiThread(new Runnable() {
499         public void run() {
500             try {
501                 if (rec_total == true) { //recibo el total_tracks
502                     String text1 = new String(txValue, "UTF-8");
503                     total_tracks = Integer.parseInt(text1);
504                     NumberPicker np2 = (NumberPicker) findViewById(R.id.numberPicker2);
505                     np2.setMinValue(1);
506                     np2.setMaxValue(total_tracks);
507                     np2.setValue(1);
508                     rec_total = false;
509                 } else

```

Figura 5.28 Recepción del número total de canciones

Cuando lo enviamos un play, siguiente, anterior o información, El Arduino nos va a devolver 3 Strings diferenciados en el tiempo por 100ms, entonces activaremos el booleano de recibir título, cuando éste se reciba el de artista y a continuación el del álbum, como se puede ver en la siguiente figura.

```

519     } else
520     if (rec_titulo == true) { //recibo el titulo, por petición de la App
521         String titulo = new String();
522         titulo = new String(txValue, "UTF-8");
523         ((TextView) findViewById(R.id.text_titulo)).setText(" "+titulo);
524         rec_titulo = false;
525         rec_artista =true;
526     }else
527     if (rec_artista == true) { //recibo el artista
528         String artista = new String();
529         artista = new String(txValue, "UTF-8");
530         ((TextView) findViewById(R.id.text_artista)).setText(" "+artista);
531         rec_artista = false;
532         rec_album =true;
533     }else
534     if (rec_album == true) { //recibo el album
535         String album = new String();
536         album = new String(txValue, "UTF-8");
537         ((TextView) findViewById(R.id.text_album)).setText(" "+album);
538         rec_album = false;
539     }else {

```

Figura 5.29 Recepción del título, artista y álbum

También ocurre el caso que el Arduino nos envíe datos sin que le hagamos una petición desde el móvil, esto ocurre en dos casos:

- Cuando termina de reproducir una canción, pasa a la siguiente, entonces nos avisa que ha realizado dicho cambio, para actualizar el número de la canción, y para ponernos a esperar recibir el título de dicha canción.

- Cuando llega a la última canción nos avisa que ha llegado al final.

```

539         }else {
540             // si no tengo ningun booleano activado, recibo por iniciativa del Arduino
541             String entrada = new String(txValue, "UTF-8");
542             String compare_continuo = "SIGUIENTE"; // siguiente canción
543             String compare_ultima = "FINAL"; // llegado a la ultima
544
545             if (entrada.equals(compare_continuo)) {
546                 rec_titulo = true;
547                 last_track = track;
548                 track = track + 1;
549                 NumberPicker np2 = (NumberPicker) findViewById(R.id.numberPicker2);
550                 np2.setValue(track); //incrementar en el picker
551                 convertir(track); //calcula n1, n2 y n3
552                 ((TextView) findViewById(R.id.track00)).setText("track" + n1 + n2 + n3 + ".mp3");
553                 Vibrator vl = (Vibrator) getSystemService(VIBRATOR_SERVICE); //vibracion
554                 vl.vibrate(500);
555                 final ToneGenerator tg = new ToneGenerator(AudioManager.STREAM_NOTIFICATION, 100);
556                 tg.startTone(ToneGenerator.TONE_CDMA_ABBR_ALERT); //dos pitidos
557                 showMessage("Siguiente: track"+ n1 + n2 + n3 + ".mp3");
558             }else
559             if(entrada.equals(compare_ultima)){
560
561                 Vibrator vl = (Vibrator) getSystemService(VIBRATOR_SERVICE); //vibracion
562                 vl.vibrate(1000);
563                 ImageView imageView = (ImageView)findViewById(R.id.status);
564                 imageView.setImageResource(R.drawable.estatus_stop);
565                 ((TextView) findViewById(R.id.text_status)).setText("Stop");
566                 showMessage("Has llegado al ultimo track");
567                 final ToneGenerator tg = new ToneGenerator(AudioManager.STREAM_NOTIFICATION, 100);
568                 tg.startTone(ToneGenerator.TONE_CDMA_ALERT_CALL_GUARD); //3 pitidos
569             }
570         }
571     }

```

Figura 5.30 Avisos de siguiente o fin

El resto de acciones que tiene esta actividad son gestiones como, a la hora de conectarse con el Arduino que activa los botones, cuando se da cuenta que se ha desconectado bloquea los botones, también incluye la gestión de parar la aplicación o ponerla en pausa (*en background*). Todos estos métodos se pueden ver en el código fuente de la aplicación, que se encuentra en el siguiente enlace.

<https://goo.gl/FUOpGE>

5.5.3 Class DeviceListActivity

Esta clase es llamada por la actividad principal, donde hemos visto en el apartado anterior, cuando se pulsa el botón de conectar. Esta actividad secundaria ejecuta una ventana emergente. Se encarga de buscar los dispositivos *Bluetooth Low Energy* disponibles para poder establecer la conexión.

Ahora vamos a ver las partes más importantes de esta clase, el código fuente completo se encuentra en el siguiente enlace.

<https://goo.gl/YDq6TI>

- En el método *onCreate* se llama al *layout* que corresponde a esta actividad, *setContentview(R.layout.device_list)* y la ventana que muestra los dispositivos, como se observa en la siguiente figura.

```

50     @Override
51     protected void onCreate(Bundle savedInstanceState) {
52
53         super.onCreate(savedInstanceState);
54         Log.d(TAG, "onCreate");
55         getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.main);
56         setContentView(R.layout.device_list);
57         android.view.WindowManager.LayoutParams layoutParams = this.getWindow().getAttributes();
58         layoutParams.gravity=Gravity.TOP;
59         layoutParams.y = 200; // posicion en la que sale la ventana emergente.
60         mHandler = new Handler();
61
62         if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
63             Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
64             finish();
65         } //Toast: Bluetooth Low Energy no compatible
66

```

Figura 5.31 Declaración de onCreate y llamada a .device_list

- En la siguiente figura se puede ver la parte donde se visualizan las características los dispositivos encontrados, llamando a el *layout* que los muestra (*R.layout.device_element*).

```

257     @Override
258     public View getView(int position, View convertView, ViewGroup parent) {
259         ViewGroup vg;
260
261         if (convertView != null) {
262             vg = (ViewGroup) convertView;
263         } else {
264             vg = (ViewGroup) inflater.inflate(R.layout.device_element, null);
265             //lamada al R.layout.device_element
266         }
267

```

Figura 5.32 Llamada a .device_element

- En esta parte se pone la dirección MAC (única e identificadora del dispositivo), nombre (configurable en el dispositivo) y el RSSI (potencia de señal recibida) de cada uno de los dispositivos encontrados.

```

267
268         BluetoothDevice device = devices.get(position);
269         final TextView tvadd = ((TextView) vg.findViewById(R.id.address));
270         final TextView tvname = ((TextView) vg.findViewById(R.id.name));
271         final TextView tvrssi = ((TextView) vg.findViewById(R.id.rssi);
272

```

Figura 5.33 Campos de cada dispositivo BLE

5.5.4 Class UartService

Esta clase no es una actividad sino un servicio. Este se encarga de comprobar que la conexión Bluetooth está establecida. La clase principal le hace preguntas periódicas para que le informe del estado de la conexión.

Esta tiene diferentes estados de la conexión, éste irá marcándose en la variable *mConnectionState*, esta inicialmente se pone en desconectado.

```
34
35     private int mConnectionState = STATE_DISCONNECTED; // INTEGER QUE VARIA ENTRE 0,1,2
36     private static final int STATE_DISCONNECTED = 0;
37     private static final int STATE_CONNECTING = 1;
38     private static final int STATE_CONNECTED = 2;
39
```

Figura 5.34 Posibles estados de la conexión

Dependiendo de lo que vaya ocurriendo en la aplicación ese estado irá variando como corresponda. El código fuente completo de esta clase se puede descargar del siguiente enlace.

<https://goo.gl/xLn2q8>

5.6 Aplicación.

En el siguiente enlace se encuentra proyecto de Android Studio archivado para que se pueda comprobar el funcionamiento de estas funciones y pueda ser útil para proyectos de otros programadores.

<https://goo.gl/0bpp22>

5.6.1 Creación del archivo .apk

Una vez finalizada toda la programación de la aplicación se procede a generar el archivo **.apk** que será el archivo que contendrá la aplicación compilada para la instalación en cualquier dispositivo con sistema operativo Android.

Para hacer esto tendremos que seguir una serie de pasos:

1. Cambiar el estado del proyecto de *debug* a *release*. Y recompilación del proyecto para asegurarnos que vamos a archivar la última versión

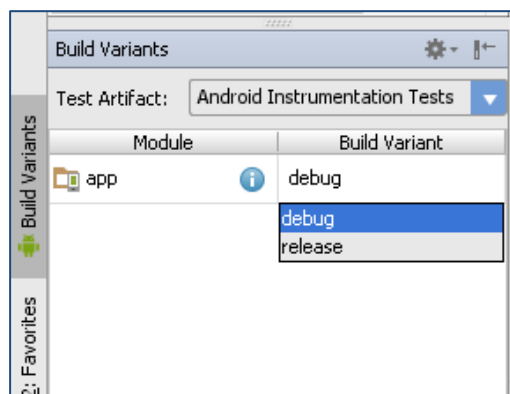


Figura 5.35 Estado *release*

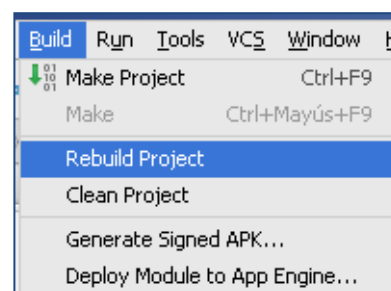


Figura 5.36 Recompilar Proyecto

2. Ahora debemos ir a la opción de generar la aplicación firmada. En mi caso ya tenía una clave creada, en ese caso se exporta, y se ponen las contraseñas oportunas.

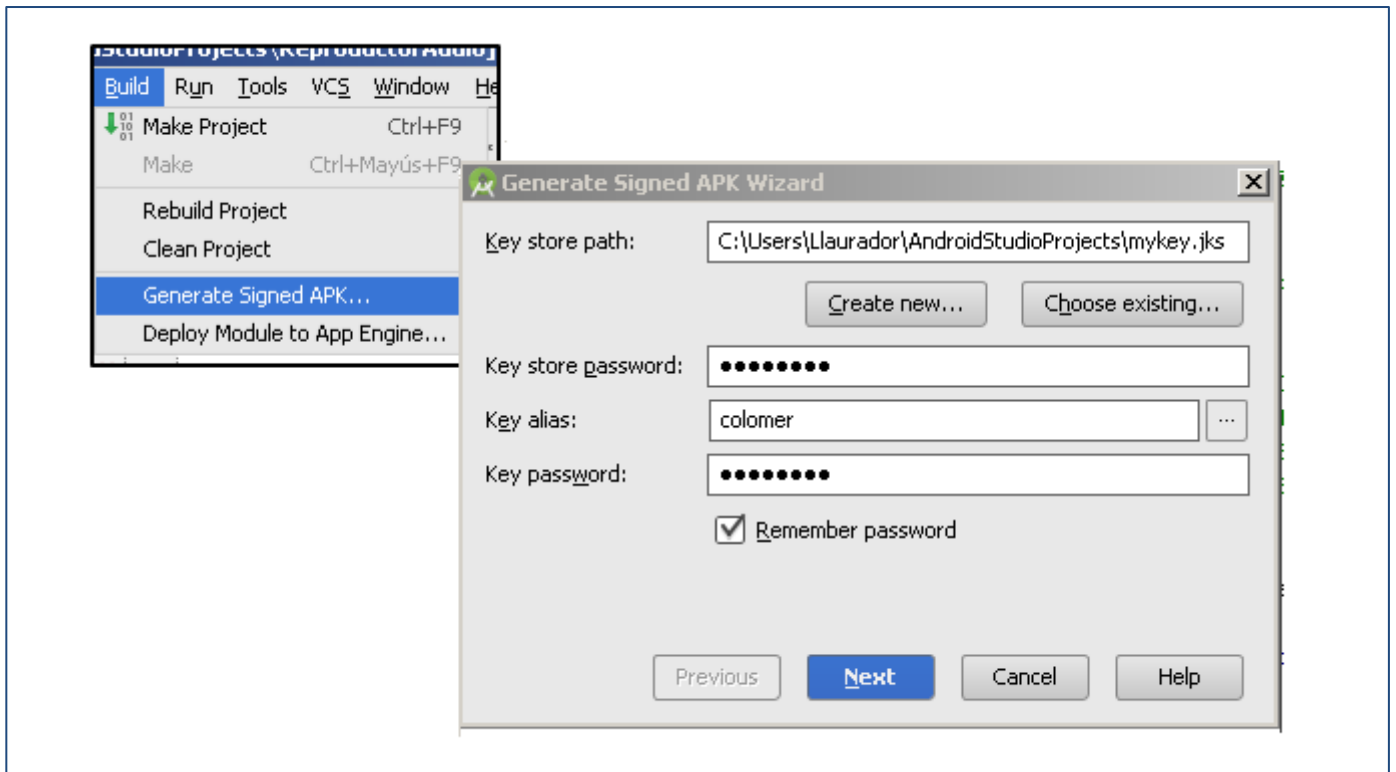


Figura 5.37 Firmar la aplicación

3. Una vez generado el archivo **.apk** lo trasladaremos al dispositivo con sistema operativo Android, en nuestro caso a los dos dispositivos que usamos para hacer las pruebas, hay que tener en cuenta que en un dispositivo con la versión de Android menor que la mínima requerida no te dejará que lo instales.



Figura 5.38 App

Enlace para descargar la App:

<https://goo.gl/d6GNMK>

5.6.2 Flujo de funcionalidad de la aplicación

En la siguiente figura se observa el diagrama de funcionamiento de la aplicación. Esta consta de dos estados principales, “No Conectado” y “Conectado”.

Ahora voy a describir las peculiaridades más importantes de esta:

La aplicación sólo se podrá cerrar cuando pulses el botón de retroceso y estés desconectado del dispositivo Arduino, si estás conectado y pulsas dicho botón, la aplicación pasa a *background* pero no se cierra. Para poder cerrarla deberías desconectar primero.

Cuando estés en desconectado, todos los botones quedan cancelados excepto el de conectar, es decir, la única opción es conectar o cerrar la aplicación. Una vez conectado, los botones del control de audio se activan.

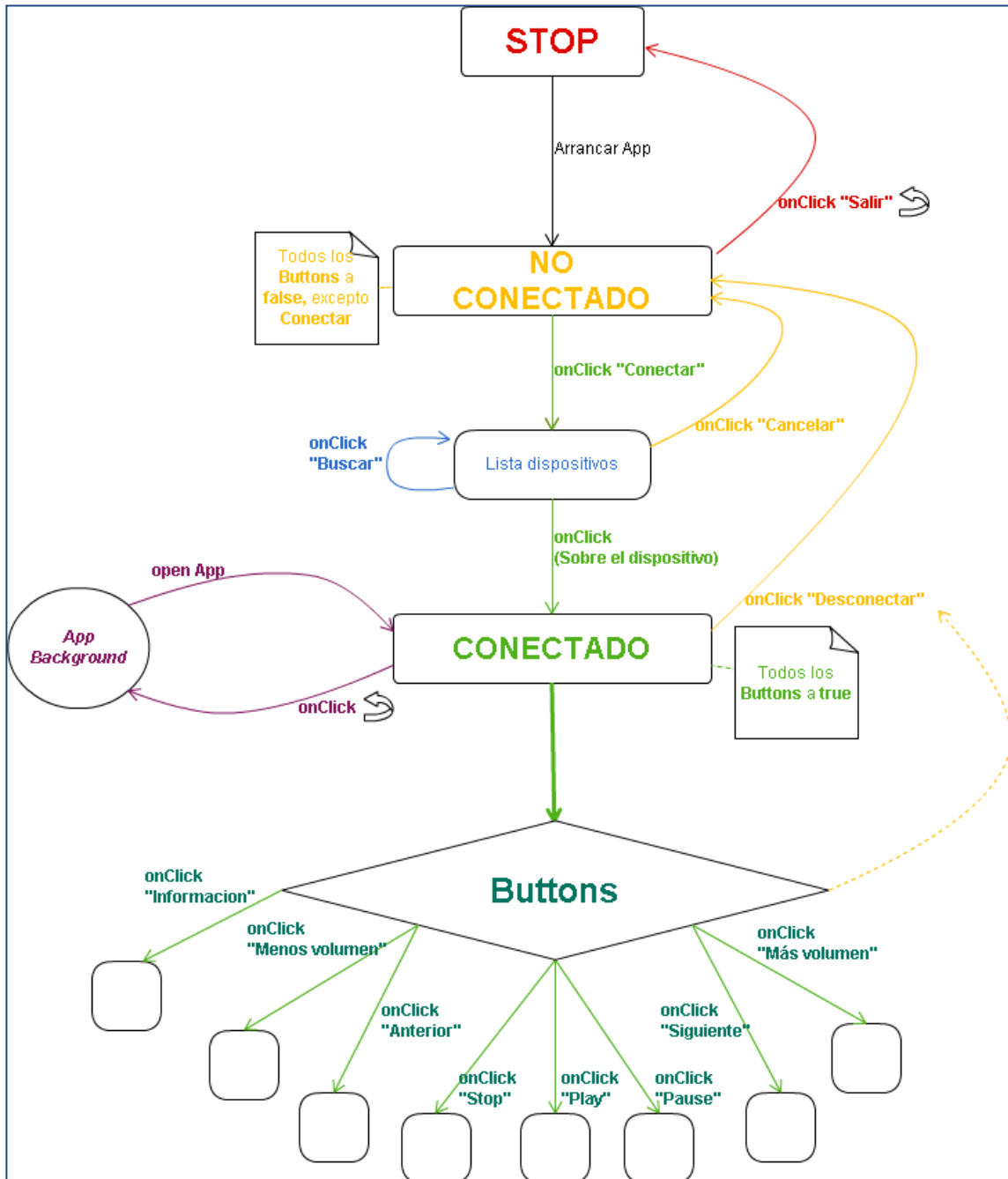


Figura 5.39 Máquina de estados de la aplicación

Capítulo 6. Arduino

6.1 Introducción sobre Arduino

Para muchos la palabra Arduino, no sabrán lo que realmente engloba, en este apartado vamos a realizar una breve introducción a algunos de los conceptos previos que debemos saber.

6.1.1 Conceptos previos

El concepto más importante que debemos conocer, es saber: ¿Qué es un microcontrolador? Es un circuito integrado o “chip”, es decir, un dispositivo electrónico que integra en un solo encapsulado un gran número de componentes, y además tiene la característica de ser programable.

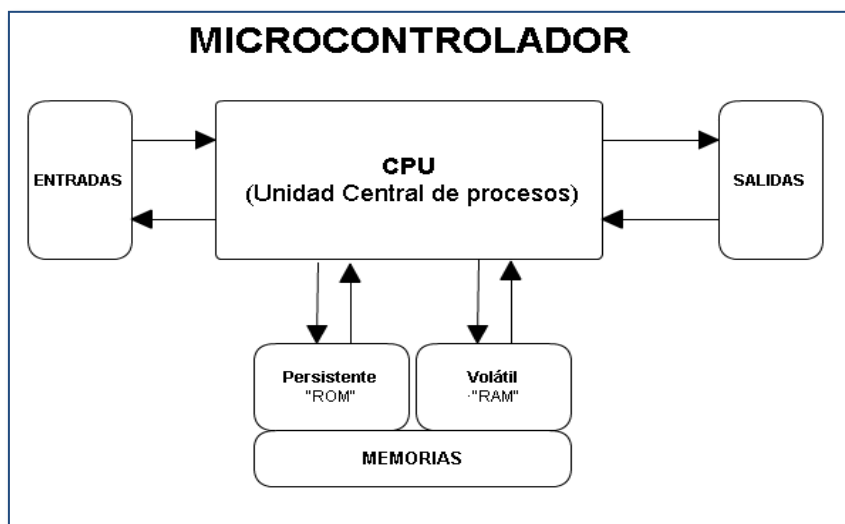


Figura 6.1 Estructura del microcontrolador

Todo microcontrolador tiene que incluir en su interior tres elementos básicos:

- **CPU (Unidad Central de Proceso).** La encargada de ejecutar cada instrucción y controlar que la ejecución se realiza correctamente. Normalmente, estas instrucciones hacen uso de datos de entrada, y producen unos datos de salida.
- **Diferentes tipos de memorias.** Son las encargadas de almacenar las instrucciones y los datos tanto de entrada como de salida, siempre disponible para el rápido acceso de la CPU. Hay dos tipos de memoria, las “volátiles o temporales”, también llamadas como RAM, que cuando quitas la alimentación pierden la información, estas normalmente se usan para el almacenamiento de datos, y las “persistentes”, también llamadas como ROM, estas mantienen la información aun quitándole la alimentación, se suelen usar para almacenar las instrucciones.

- **Entradas/ Salidas.** Son las encargadas de comunicar el microcontrolador con el exterior. Se pueden conectar sensores a la entrada, que alimentan al microcontrolador de datos para que pueda realizar las instrucciones, y actuadores a la salida.

Otro concepto importante es tener conocimientos elementales sobre la electrónica, tanto teóricos para el cálculo de potencias, voltaje o intensidades, como de diferentes componentes como es el caso de resistencias, diodos LED, transistores, condensadores, etc.

También tendremos que tener nociones de señales, saber las diferencias entre una señal digital y analógica. La señal digital es aquella que tiene un número finito de valores y la analógica, es aquella que tiene infinitos valores posibles dentro de un rango determinado.

6.1.2 ¿Qué es Arduino?

La palabra Arduino engloba mucho más que un microcontrolador, más bien es una plataforma de hardware abierta para la creación de nuevos prototipos. Estos ocupan un espacio físico relativamente pequeño y pueden tener una gran funcionalidad.

Principalmente consta de tres partes:

- **Placa hardware libre.** Esta incorpora un microcontrolador programable y una serie de pines, los cuales están unidos a las E/S del microcontrolador. Cuando nos referimos a una “placa hardware”, nos estamos refiriendo a un PCB (*printed circuit board*), es decir, una placa de circuito impreso. Aunque cuando hablamos de “placa Arduino”: deberíamos especificar el modelo, ya que existen una gran variedad de placas Arduino oficiales. Cada una con unas características concretas. Conviene conocer-las para hacer una correcta elección del dispositivo para que se adapte mejor a cada proyecto.
- **Un software** (más en concreto, “entorno de desarrollo integrado”) gratis y libre. Está disponible para descargarse desde su página web de Arduino (<http://www.arduino.cc/>). Contiene versiones para GNU/Linux (como *Ubuntu o Fedora*), Mac OS y Windows. Este tiene las funciones de verificar, compilar y cargar al microcontrolador las instrucciones que deseemos, es decir, nos permite programarlo.



Figura 6.2 Software para la programación en Arduino

- **Lenguaje de programación libre:** Es cualquier idioma artificial diseñado para expresar tareas mediante algoritmos utilizando una serie de instrucciones que pueden ser llevadas a cabo por máquinas. Dentro del lenguaje propio de Arduino encontramos muchas similitudes a otros lenguajes de programación (en la declaración de variables, bloques condicionales, bucles o ciclos, etcétera). El lenguaje de programación está inspirado en el lenguaje *Processing*, pero su base interior está creada sobre C/C++.

6.1.3 Historia de Arduino

Arduino nació en 2005, en el instituto de Diseño Interactivo de Ivrea (Italia). Su profesor *Massimo Banzi*, vio la necesidad de crear dispositivos de coste muy económico (inferior a 100\$) que no existían en el mercado. Con la premisa de que fuera compatible con todos los sistemas operativos y sin barreras de entrada para los principiantes.

Pese a que el instituto cerró sus puertas, se generó un proyecto llamado “*Arduino Team*”. Se decidió abrir el proyecto a todo el mundo interesado para así poder recibir información y ayuda de toda la comunidad. Surtiendo efecto al poco tiempo, se empezaron a sacar nuevas ideas y modificaciones que agrandaron el proyecto. Actualmente tenemos un conjunto de placas programables a un precio muy asequible, y con un software de programación oportuno para dichos microcontroladores.

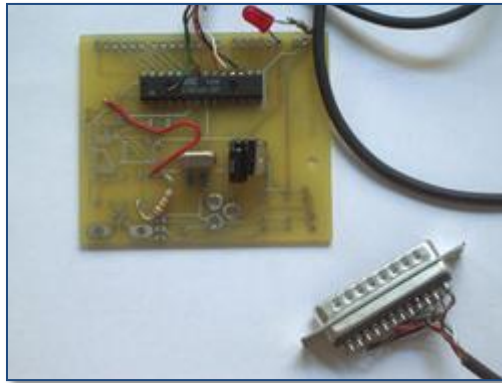


Figura 6.3 Primer prototipo de Placa Arduino

6.1.4 ¿Cómo se programa?

Como hemos comentado anteriormente tiene un software propio para la programación. En el cual se puede editar, verificar y cargar las instrucciones en el microcontrolador. En la mayoría de placas esto se hace mediante USB, previamente debes tener instalado los controladores (*drivers*) para que reconozca el dispositivo. Para la descarga del programa, en las placas más antiguas se deberían mantener el botón del *reset* pulsado, en las placas actuales eso ya no es necesario.

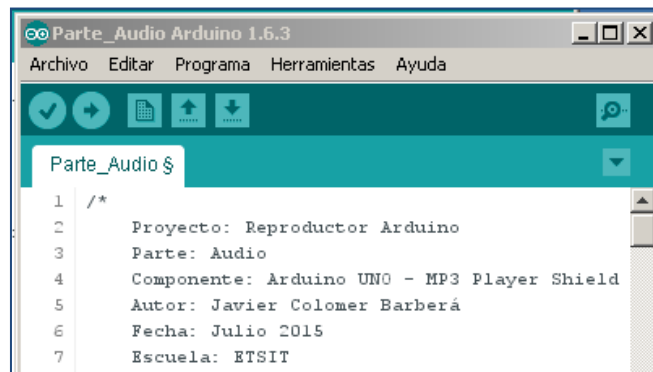


Figura 6.4 Entorno de desarrollo

Lo primero que tenemos que hacer para empezar a trabajar con el IDE es configurar las comunicaciones entre la placa y el PC. Para ello debemos abrir el menú “Herramientas” y elegir la opción “Puerto”. Indicando el puerto serie que está asociado a nuestra placa.

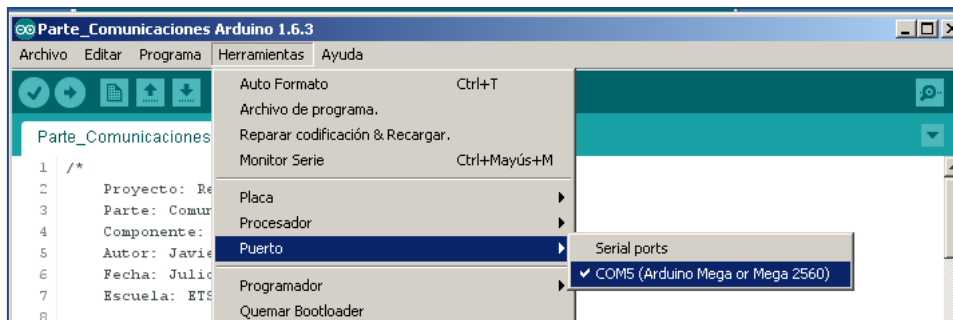


Figura 6.5 Enlazar placa Arduino

6.1.4.1 Lenguaje Arduino

La estructura de un programa diseñado para ejecutarse sobre un Arduino recibe el nombre de “*sketch*”, y siempre está compuesto por tres secciones:

- **Declaración de variables globales.** Ubicada al principio del *sketch*. Este no tiene ningún símbolo delimitador de inicio y final. En este se declaran todas las variables globales que se van a usar. También se incluyen el llamamiento a las librerías que serán usadas en las siguientes secciones.

```

25 //Variables
26 char comando; //Comando leído del móvil
27 int n1=0; // n1= centenas n2=decenas n3=unidades
28 int n2=0; //SE PONEN VALORES EN FUNCION convertir(canción)
29 int n3=1;
30 int max_canciones; //Se introduce el valor al llamar a calculo_canciones();
  
```

Figura 6.6 Declaración de variables

- **“*setup()*”.** Esta sección consta de un bloque lógico que está delimitado por llaves de apertura y cierre. Sólo se ejecuta una única vez, al encender la placa o resetear esta. Aquí se suele incluir funciones de preconfiguración como la apertura de los canales de comunicación, como es el caso de *Serial.begin()*;
- **“*loop()*”.** Esta sección también consta de un bloque lógico que está delimitado por llaves de apertura y cierre. Se ejecuta después de la inicialización del *setup()* de forma continua hasta que la placa se apague. Es decir, el contenido de esta sección es un bucle infinito que se estará ejecutando indefinidamente de principio a fin hasta que se deje de alimentar el dispositivo.

No es obligatorio que todas las instrucciones estén dentro de estas secciones, como en cualquier lenguaje de programación puedes hacer tus propias funciones, pero su llamamiento sí que tiene que estar dentro del *setup()* o del *loop()*, sino nunca se utilizarán.

6.1.5 Bibliotecas (Libraries)

En la programación de Arduino, es muy común el uso de bibliotecas, en alguna documentación aparece como librerías, esto es debido a que proviene del término en inglés *Libraries*. La RAE (*Real Academia Española*), acepta estos dos términos como sinónimos, es decir ambos son válidos.

Estas sirven para proporcionar funcionalidad extra en nuestros *sketches*. Pueden ser creadas por Arduino o por el fabricante de *Shields*.

- **Bibliotecas creadas por Arduino**, son las bibliotecas oficiales que permite la manipulación de datos, comunicación de diversos dispositivos, interactuar con hardware, etcétera. Estas se instalan automáticamente cuando realizamos la instalación de IDE.

BIBLIOTECAS OFICIALES	
Bridge	SD
EEPROM	Servo
Esplora	SoftwareSerial
Ethernet	SpacebrewYun
Firmata	SPI
GSM	Stepper
LiquidCrystal	Temboo
Robot Control	TFT
Robot IR Remote	WiFi
Robot Motor	Wire

Tabla 6.1 Bibliotecas oficiales

En nuestro caso hacemos uso de la biblioteca SPI (*Serial Peripheral Interface*), la cual gestiona el protocolo de comunicación, entre un maestro (placa Arduino) y un esclavo (dispositivo externo). En el apartado 6.2.1 (*Protocolo de comunicación SPI*) entraremos más en detalle sobre el funcionamiento.

- **Bibliotecas creadas por fabricantes de Shields**. Los fabricantes crean sus propias bibliotecas para facilitar la implementación de sus dispositivos. Nosotros solo tendremos que llamar a dicha biblioteca y a los métodos que contenga.

BIBLIOTECAS DE NUESTROS DISPOSITIVOS EXTERNOS	
SdFat	SPFEMP3Shield
SdFatUtil	Adafruit_BLE_UART

Tabla 6.2 Bibliotecas del fabricante

En nuestro proyecto usamos estas cuatro bibliotecas, para la gestión de nuestros dos esclavos. En el apartado que corresponda explicaremos el motivo y su uso.

Para incluir un biblioteca tanto oficial como no, basta con añadir una declaración `#include <Nombre_libraries>` al comienzo del *sketch*. Por ejemplo, si quieres incluir la biblioteca de cristal líquido, que se utiliza para exhibir datos en una pantalla LCD, basta con incluir la siguiente declaración al principio del *sketch*.

```
#include <LiquidCrystal.h>
```


6.1.6 Modelos

Por las circunstancias expuestas en el *Capítulo 3 Propuesta inicial* hemos usado dos placas Arduino. Las dos placas que se describen a continuación.

- **Arduino Uno R3.** Esta es la placa estándar de Arduino. Es la más usada, y desde que en 2010 apareció ha sufrido tres revisiones.



Figura 6.7 Arduino Uno R3 [9]

Esta placa está basada en microcontrolador ATmega328P, que es el mismo que ATmega328, pero este incluye un ahorro energético.

ARDUINO Uno R3	
Microcontrolador	ATmega328
Tensión De Funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Pines Digitales E/S	14 (de las cuales 6 proporcionan salida PWM)
Pines Analógicos	6
Memoria Flash	32 Kbytes
RAM	2 Kbytes
EEPROM	1 KB
Frecuencia de Reloj	16 MHz
Longitud	68,6 mm
Anchura	53,4 mm

Tabla 6.3 Características del Arduino Uno R3

- **Arduino MEGA 2560** Esta placa es la que contiene mayor número de señales. La elegimos por la cantidad de puertos UART serie que tiene, como va a ser la encargada gestionar las comunicaciones, necesita dos puertos para comunicar con el otro Arduino y el resto serán usados para añadir dispositivos de comunicación.

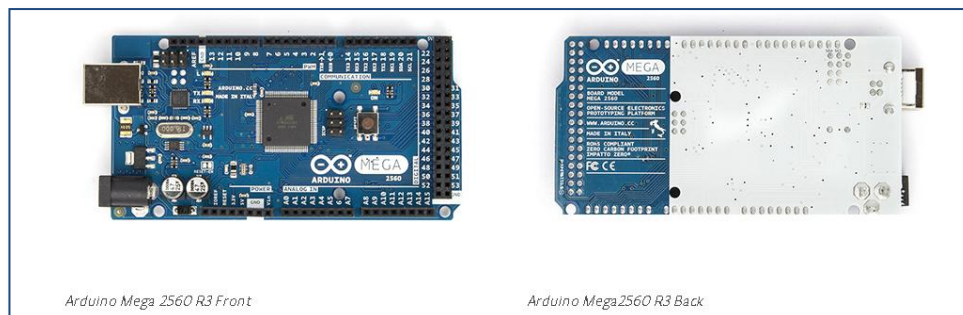


Figura 6.8 Arduino Mega 2560 R3 [9]

En la siguiente tabla se muestran las principales características de esta placa.

ARDUINO Mega 2560	
Microcontroladores	Atmega2560
Tensión De Funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Pines Digitales E/S	54 (de las cuales 15 proporcionan salida PWM)
Pines Analógicos	16
Memoria Flash	256 Kbytes
RAM	8 KB
EEPROM	4 KB
Frecuencia de Reloj	16 MHz
Longitud	102 mm
Anchura	53,4 mm

Tabla 6.4 Características del Arduino Mega 2560 R3

6.2 Dispositivos externos.

En este apartado vamos a ver como incorporar funcionalidades a las placas Arduino, añadiéndoles dispositivos externos, estos pueden ser de dos tipos: sensores (reciben información de exterior) o actuadores (producen alguna respuesta en el medio). Aunque la mayoría de ellos son capaces de realizar las dos cosas: recibir y producir. Se suelen llamar como esclavos, ya que en la mayoría dependen de la placa maestro, la placa Arduino. En cuanto al conexionado se pueden dividir en dos grupos.

- **Shields** (traducido del inglés como “escudo”), son placas de circuito impreso que se colocan en la parte superior del Arduino, para añadir funcionalidades, se conectan a ella mediante el acoplamiento de sus pines sin necesidad de cables.

Dependiendo del modelo incluso se pueden apilar varios *shields*, uno encima de otro. Estos deben compartir las líneas GND, 5V (o 3.3V), RESET y AREF con el Arduino.

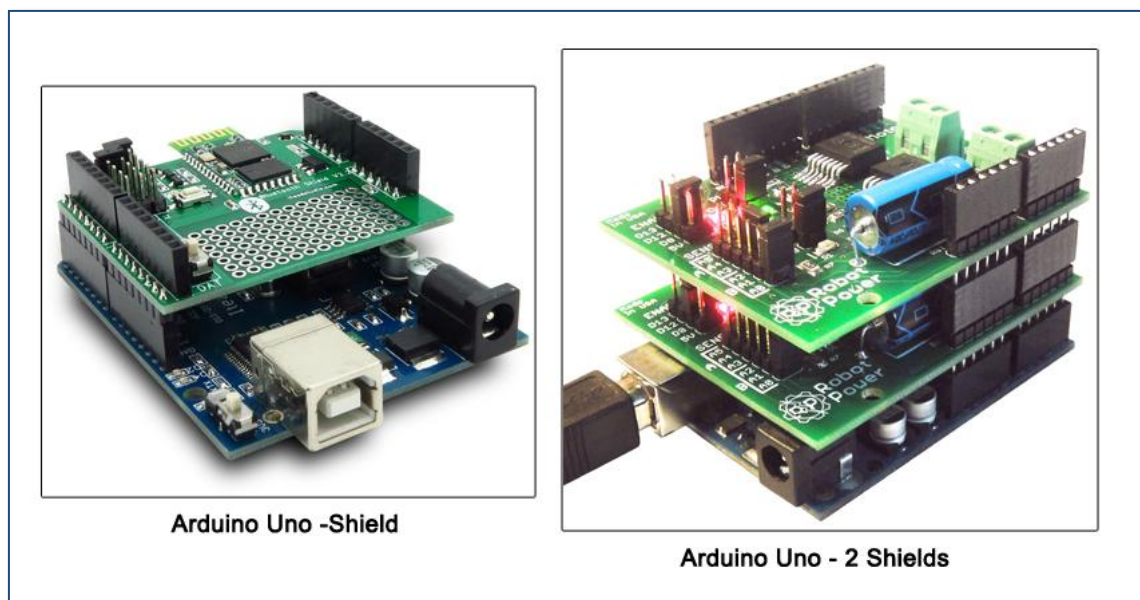


Figura 6.9 Acoplamiento de Shields

Existe un amplio listado de *shields* oficiales de Arduino, es decir, fabricados por Arduino. Por ejemplo, los nombrados a continuación:

- Arduino Ethernet Shield.
- Arduino Wireless SD Shield.

- Arduino Wireless Proto Shield.
- Arduino Wi-Fi Shield.
- Arduino Motor Shield.

Aunque también hay una gran cantidad de *shields* diseñados y construidos por otros fabricantes (en Julio de 2015 casi 500), y por eso no dejan de ser útiles, todo lo contrario incluyen muchas más funcionalidades que complementan a las oficiales. Para la elección del Shield es interesante consultar este sitio web <http://shieldlist.org/> donde ofrece una lista centralizada de prácticamente todos los modelos existentes, clasificados por fabricantes.

- **Dispositivos externos no estándar.** Estos también son placas de circuito impreso como los *Shields*, la única diferencia respecto a estos es que no pueden ir acoplados encima, y se debe utilizar un *Breadboard* con sus respectivos cables para el conexionado.

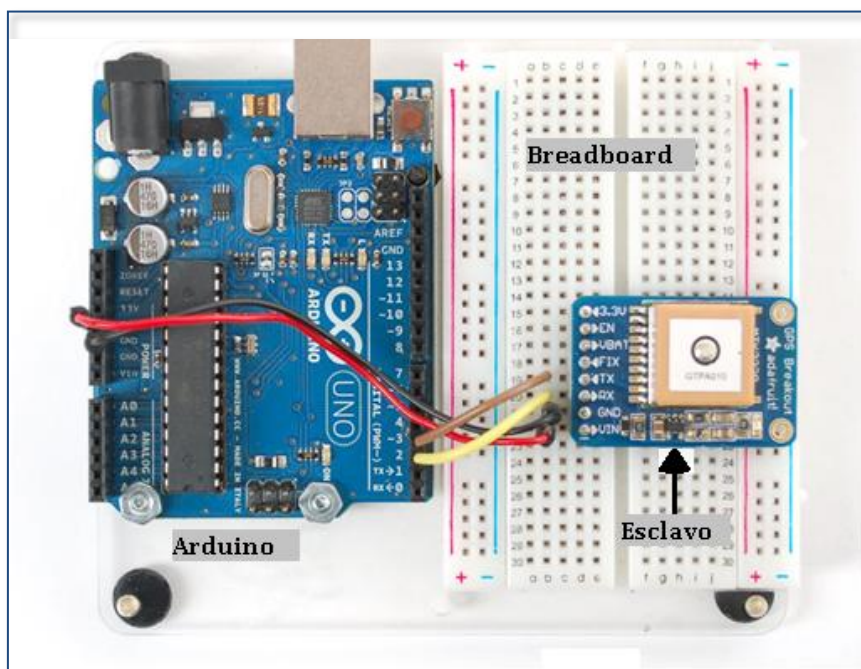


Figura 6.10 Dispositivo con *Breadboard*

Para poder evitar usar una *breadboard* y poder implementar los circuitos de una forma mucho más compacta, existe el “*proto shield*”. Este permite que el montaje que tendrías sobre la *breadboard* externa lo pudieras crear sobre el propio Arduino, como si de otro *Shield* se tratara.

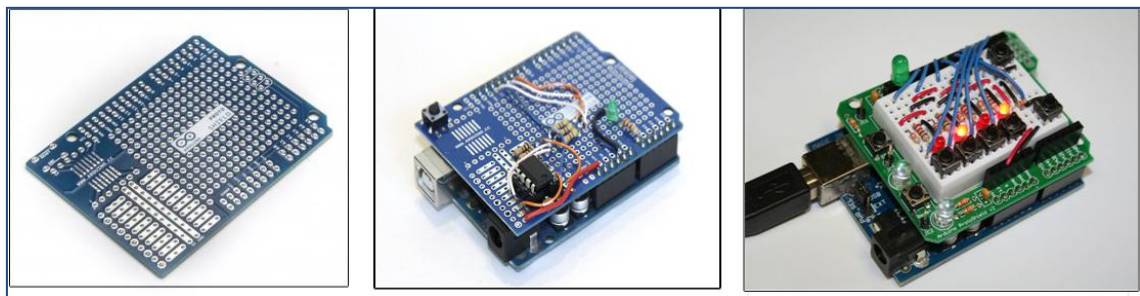


Figura 6.11 *Proto Shield*

6.2.1 Protocolo de comunicación SPI

Todos los dispositivos externos nombrados anteriormente, se deben comunicar con el Arduino. Existen diferentes protocolos para este fin como es el caso de *Serie, I²C o SPI*. En nuestro caso los dispositivos que vamos a utilizar emplean el protocolo SPI (*Serial Peripheral Interface*).

SPI internamente usa la comunicación serie, es decir, un bit detrás de otro por un único canal. Ya que la comunicación en paralelo supondría la necesidad de más canales para un único dato.

SPI parte de la premisa de ser un protocolo de transmisión síncrona que permite alcanzar altas velocidades, su diseño está pensado para comunicar un microcontrolador con distintos periféricos y que funcione en modo full dúplex, es decir, que puede enviar y recibir datos al mismo tiempo.

Este protocolo está formado por cuatro canales.

- **SCLK** (reloj). Esta señal indica en que instante de tiempo concreto deben leer o escribir tanto los dispositivos externos como el microcontrolador. Por este motivo no es necesario pactar la velocidad de transmisión con anterioridad, incluso puede ser variable a lo largo de la comunicación sin que esto sea un problema.
- **MOSI** (Master Output – Slave Input). Por este canal solo irán los datos que procedan del maestro (*master*) hacia el esclavo (*slave*). Siendo el canal de transmisión del primero y de recepción del segundo.
- **MISO** (Master Input – Slave Output). Este es el canal opuesto al anterior, por esto solo irán los datos del esclavo (*slave*) hacia el maestro (*master*). Siendo el canal de transmisión del primero y de recepción del segundo.
- **SS** (Slave Select). El *master* activará esta canal para indicar al esclavo que puede usar los canales MOSI y MISO. La activación del canal es a nivel bajo, es decir, cuando el SS este a 0 es cuando podrá leer y escribir.

Si existe más de un esclavo (*slave*) entonces se comparten los canales SCLK, MOSI y MISO. Teniendo un SS diferente para cada uno de los *slaves*, para así poder elegir cuál es el que tiene que ocupar los canales de escritura/lectura. Hay que tener en cuenta que solo podrá estar un SS activado a la vez, si esto no se cumple varios *slaves* intentará acceder al mismo tiempo al bus de datos.

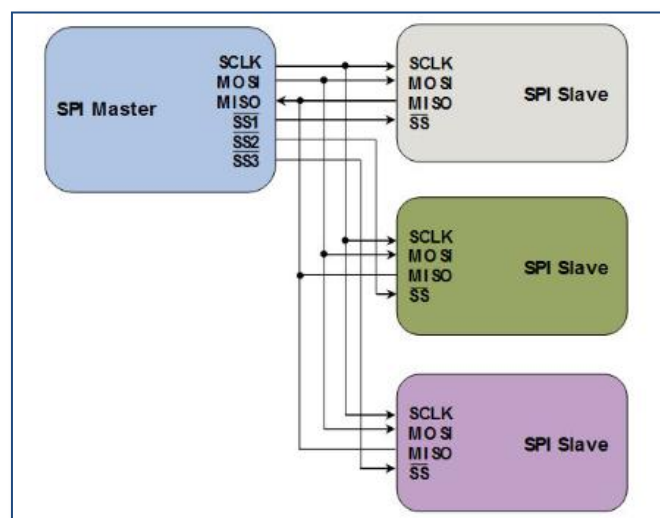


Figura 6.12 Protocolo SPI múltiples esclavos

Dependiendo del modelo de placa Arduino varia la ubicación de los pines del protocolo SPI, los pines MISO, MOSI y SCLK no son configurables, es decir, no se pueden cambiar de ubicación física, sin embargo el SS se puede cambiar al pin que mejor convenga. En la siguiente tabla se observa la distribución de estos pines dependiendo de la placa Arduino.

Arduino Board	MOSI	MISO	SCK	SS (slave)
Uno o Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10
Mega1280 o Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53
Leonardo	ICSP-4	ICSP-1	ICSP-3	-

Tabla 6.5 Distribución de pines

6.3 Antena Bluetooth nRF8001

La antena Bluetooth nRF8001 Breakout permite establecer un enlace inalámbrico entre el dispositivo Arduino y un dispositivo con sistema operativo iOS o Android (4.3+), es decir versión de Android 4.3 o superior.

Funciona mediante la simulación de un dispositivo UART (*Universal Asynchronous Receiver – Transmitter*), produciendo un envío de datos ASCII de ida y vuelta entre los dispositivos. El UART toma los bytes de datos y transmite los bits individualmente, en la parte que recibe los bits, los reensambla y crea el byte completo en el otro extremo de la comunicación.

6.3.1 Conexión

Como hemos indicado anteriormente este funciona con el protocolo SPI. A continuación vamos a desglosar los pines que compone este dispositivo externo. En nuestro proyecto junto a este dispositivo hemos usado el Arduino MEGA 2560. Que su principal característica es la gran cantidad de pines que tiene, entre ellos hasta cuatro parejas de pines serie Tx/Rx.



Figura 6.13 nRF8001 Breakout [7]

- **SCK.** Esta es la señal de reloj, debe conectarse a la señal de reloj del *master* SPI. En nuestro proyecto como lo conectamos a un Arduino MEGA, corresponde con el pin Digital 52.
- **MISO.** En este canal se encuentran los datos de salida del *slave*, los de la antena Bluetooth dirección a la placa Arduino. En nuestro proyecto como lo conectamos a un Arduino MEGA, corresponde con el pin Digital 50.
- **MOSI.** En este canal se encuentran los datos de entrada del *slave* en este caso los que la placa Arduino envía a la antena Bluetooth. En nuestro proyecto como lo conectamos a un Arduino MEGA, corresponde con el pin Digital 51.

- **REQ.** Este es el equivalente al que en el protocolo SPI denominamos *Chip Select*, el que selecciona el dispositivo que tiene que leer/escribir en el canal. Funciona a nivel bajo. En nuestro proyecto hemos decidido ponerlo en el pin Digital 49.
- **RDY.** Este es el pin que nos indica que la antena esta lista para interpretar o recibir los datos. En nuestro proyecto hemos decidido ponerlo en el pin 21. Tiene que ser un pin de interrupción, todos los pines no tiene esa característica, en la siguiente tabla se encuentran los que sí que la contiene.

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

Tabla 6.6 Pines de interrupción

- **ACT** Es una salida que nos indica cuando el modulo está ocupado. En nuestro proyecto este no está usado.
- **RST** Reset. En nuestro proyecto hemos decidido ponerlo en el pin Digital 47.
- **3Vo** En esta salida tenemos 3.3V, puede ser usada para alimentar otros dispositivos y pueden conseguir hasta 100mA.
- **GND** Masa, esta debe ser común a la de la placa Arduino y a la resta de dispositivos.
- **VIN** Entrada de alimentación de 3 a 5 voltios. En nuestro caso lo alimentamos a 5 Vo.

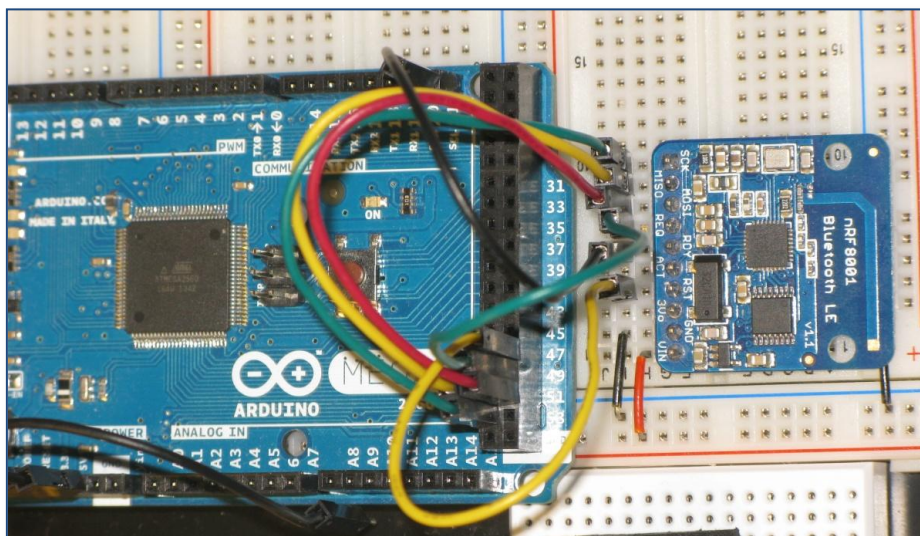


Figura 6.14 Arduino Mega – nRF8001 BLE

En la siguiente figura se puede apreciar con claridad el funcionamiento de la señal RDY, esta figura muestra cuando la placa Arduino le avisa a la antena de que tiene algo para leer, él tiene un tiempo de salvaguarda para decirle que está listo, esa señal se pone a bajo cuando realmente está preparado.

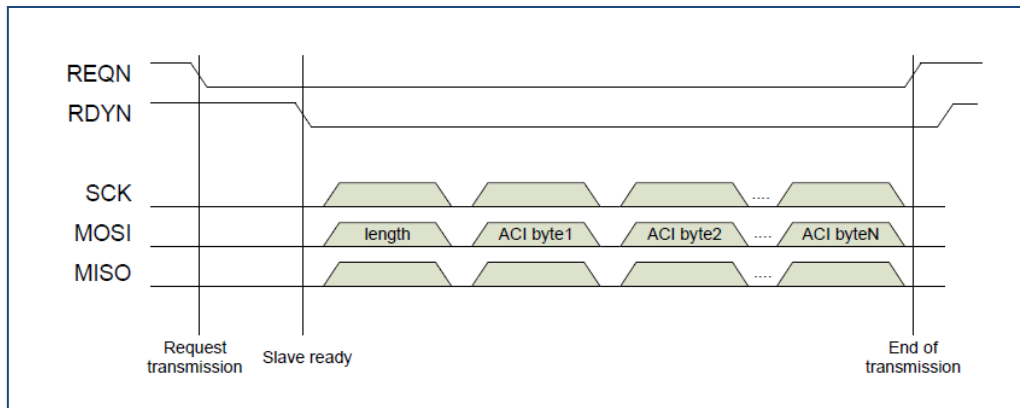


Figura 6.15 Intercambio de datos desde controlado a nRF8001

6.3.2 Sketch- Parte Comunicación

En este apartado vamos a describir la parte de programa que realiza la gestión de la comunicación, es decir, el programa que ejecuta en el Arduino MEGA 2560.

```

Parte_Comunicaciones
1  /*
2   Proyecto: Reproductor Arduino
3   Parte: Comunicación ARDUINO-MOVIL
4   Componente: Arduino MEGA - nRF8001 Breakout
5   Autor: Javier Colomer Barberá
6   Fecha: Julio 2015
7   Escuela: ETSIT
8
9   Descripción: Control de la antena nRF8001. Retransmitiendo los valores que le llegan por ella a otro Arduino.
10  Por el Serial1 pines 18 y 19. Y los datos que le llegan de este los emite al dispositivo movil.
11  */

```

Figura 6.16 Cabecera Sketch - Parte Comunicación

Lo primero que debemos añadir son las bibliotecas, en este caso la *SPI* junto con *Adafruit_BLE_UART*, estas se encargan de gran parte del trabajo pesado de la gestión de conexión, envío y recepción de datos. También del almacenamiento de los datos en un buffer de entrada para que Arduino los pueda interpretar cuando esté disponible. La propia biblioteca de la antena Bluetooth es la encargada de llamar a la de *SPI*.

A continuación viene la declaración de variables, se encuentran las tres necesarias para llamar al objeto “*Adafruit_BLE_UART*”, declarado como el objeto *BTLEserial* (en la línea 26). También están declaradas las variables de los dos LED que indican el estado de la conexión en los pines 22 y 24, junto con una variable booleana. Por último inicializamos la variables *laststatus* ha desconectado.

```

13 #include <SPI.h>
14 #include <Adafruit_BLE_UART.h>
15
16 /* INICIALIZACIÓN PROTOCOLO SPI, PARA LA ANTENA BT */
17 //Como es el Arduino Mega: CLK = 52, MISO = 50, MOSI = 51
18 #define ADAFRUITBLE_REQ 49
19 #define ADAFRUITBLE_RDY 21 // Interrupt
20 #define ADAFRUITBLE_RST 47
21
22 const int ledConect = 22; // Conectado
23 const int ledDesconect = 24; // Desconectado
24 bool reset = false;
25
26 Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
27
28 aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;
29

```

Figura 6.17 Declaración de variables – Parte Comunicación

La segunda sección es *setup()*, en este se introducen las instrucciones que solo van a ejecutarse cuando arranques la placa, una única vez.

Una de las cosas que típicamente encontraremos en esta parte son las llamadas a la instrucción *begin()*, en este caso tanto del canal *Serial* estándar (pines Digitales 0 y 1, también asociado al monitor serie que podemos encontrar en el entorno de desarrollo), como la del *Serial1* (pines Digitales 18 y 19, para comunicación con Arduino Uno). Estas llamadas son necesarias para abrir los canales de comunicación, por lo tanto, esto es imprescindible antes de realizar cualquier transmisión. El valor que le pasamos especifica la velocidad en bits/s a la que se producirá la transferencia serie de datos. El objeto *BTLESerial*, también necesita su llamada a la instrucción *begin()*, pero sin parámetro de entrada.

La instrucción *println()*, permite enviar datos desde el microcontrolador hacia el exterior. Por lo dicho anteriormente la instrucción de la línea 33 muestra en el monitor serie del PC “*Reproductor Arduino:*” y la línea 40 envía el carácter ‘C’ al Arduino Uno.

También declaro las variables *ledConect* y *ledDesconect* como variables de salida.

```
30 void setup(void)
31 {
32     Serial.begin(115200);
33     Serial.println(F("Reproductor Arduino:"));
34     pinMode(ledConect, OUTPUT);
35     pinMode(ledDesconect, OUTPUT);
36     BTLESerial.setDeviceName("ARDUINO"); /* 7 characters max! */
37
38     BTLESerial.begin();
39     Serial1.begin(115200);
40     Serial1.println('R'); //envia una 'R' para que el chip MP3 se resete.
41 }
42
```

Figura 6.18 Setup – Parte Comunicación

La siguiente sección es *loop()*, todas las instrucciones que estén dentro de su corchete de inicio y final se repetirán hasta que se interrumpa la ejecución.

En nuestro Arduino lo que haremos será una comprobación del estado de la conexión y actuaremos de diferente manera dependiendo del estado que se encuentre en cada momento.

Lo primero que nos encontramos es la llamada a la instrucción *pollACI()*, está programada dentro de la biblioteca que nos proporciona el fabricante de la Antena, y lo que hace es recordarle a la antena que está trabajando.

A continuación leemos el estado mediante la instrucción *.getState()*, y comprobamos si ha cambiado de estado, en el caso que cambie actuaremos como indica la siguiente figura. Los diferentes estados en los que se puede encontrar son:

- ***ACI_EVT_DEVICE_STARVED***: El dispositivo esta desconectado pero accesible para conectar con otros dispositivos.
- ***ACI_EVT_CONNECTED***: La conexión se ha establecido.
- ***ACI_EVT_DISCONNECT***: La conexión con otro dispositivo se ha cerrado.


```

44 void loop()
45 {
46   BTLEserial.pollACI();
47
48   // Pregunta cual es nuestra situacion actual
49   aci_evt_opcode_t status = BTLEserial.getState();
50
51   if (status != laststatus) { // Si el estado es diferente al anterior
52     if(status == ACI_EVT_DEVICE_STARTED) {
53       Serial.println(F("* Esperando conexion"));
54       digitalWrite(ledConect, LOW);
55       digitalWrite(ledDesconect, HIGH);
56     }
57     if (status == ACI_EVT_CONNECTED) {
58       Serial.println(F("* Conectado"));
59       digitalWrite(ledConect, HIGH);
60       digitalWrite(ledDesconect, LOW);
61       reset =true;
62       Serial1.println('r'); /*solo envia cuando se acaba de conectar,
63                            este devuelve el numero total de canciones*/
64     }
65     if (status == ACI_EVT_DISCONNECTED) {
66       Serial.println(F("* Desconectado"));

```

Figura 6.19 Loop – Parte Comunicación

Si el estado de la antena es conectado, compruebo si tengo datos en algún sentido.

- **BTLEserial.available()**. Si este es positivo es que tengo datos procedentes vía Bluetooth, entonces el Arduino los lee, y los reenvía al otro Arduino que tiene conectado por serie. Esto es lo que ocurre en el bucle *while* de la línea 77.
- **Serial1.available()**. Si este es positivo significa que el Arduino conectado en serie a este, le está enviando datos, lo que tendrá que realizar es, recibir esos datos, prepáralos en palabras de 20 bytes y enviar al dispositivo mediante la antena Bluetooth. Esto es lo que vemos en el bucle *while* de la línea 86.

```

74   if (status == ACI_EVT_CONNECTED)
75   {
76
77     while (BTLEserial.available()) //Recibe datos del telefono
78     {
79       char c = BTLEserial.read();
80       Serial.print(c);
81       Serial.print("\n");
82       Serial1.println(c); //Envia al otro Arduino
83     }
84
85     //Lo que recibe del otro Arduino lo envia a traves de la antena al telefono
86     while (Serial1.available()) //Recibe datos del otro Arduino
87     {
88       Serial1.setTimeout(100); // 100 millisecond timeout
89       String s = Serial1.readString();
90       uint8_t sendbuffer[20];
91       s.getBytes(sendbuffer, 20);
92
93       char sendbuffersize = min(20, s.length());
94       // Serial.print("\n");
95       //Serial.print (" Texto recibido de Arduino UNO: ");
96       Serial.print((char *)sendbuffer);
97
98       BTLEserial.write(sendbuffer, sendbuffersize); //Envio al telefono
99     }

```

Figura 6.20 Acciones cuando está conectado

Si el estado es desconectado manda una única vez una 'R' al Arduino que tenemos conectado en serie. En el otro Arduino hemos programado que esa 'R' la interprete como que tiene que hacerse un reset.

```

102
103   if (status != ACI_EVT_CONNECTED && reset == true) {
104       Serial1.println('R');           //Cuando se desconecta, envia una 'R' para que el chip MP3 se resete.
105       reset = false;
106   }
107
108 }

```

Figura 6.21 Acciones cuando está desconectado

El código fuente completo se encuentra en el siguiente enlace.

<https://goo.gl/dbIKjT>

6.3.3 Esquema Parte Comunicaciones

Teniendo esta parte finalizada ya somos capaces de intercambiar caracteres ASCII, entre el Arduino Mega y el dispositivo con sistema operativo Android, el cual tendrá que tener instalada la aplicación hecha al respecto para este proyecto.

El prototipo de esta parte de comunicaciones es el que se muestra en la siguiente figura (Figura 6.22 Montaje Arduino Mega con Antena Bluetooth nRF8001).

En los LED de estado hemos puesto una resistencia en serie de 3kΩ, para que los LED puedan durar más tiempo, ya que con la luz que proporcionan con esta resistencia es suficiente.

La masa y la alimentación deben de ser comunes a todos los componentes del circuito, tanto la placa Arduino, como la antena Bluetooth y los LED.

Abriendo el monitor serie del software de desarrollo podemos intercambiar caracteres ASCII y comprobar el buen funcionamiento de esta parte.

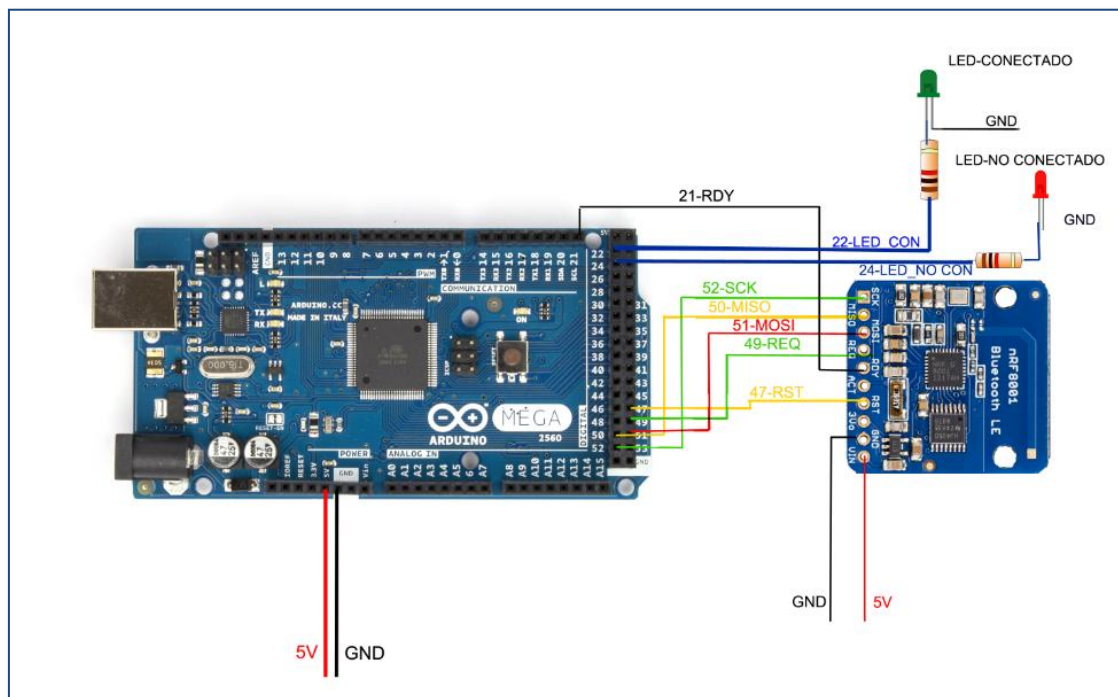


Figura 6.22 Montaje Arduino Mega con Antena Bluetooth nRF8001

6.4 MP3 Player Shield

El *MP3 Player Shield* es un dispositivo externo que se acopla al Arduino como un escudo. Este permite reproducir sonidos MP3 almacenados en la tarjeta μ SD que introduces en el zócalo del escudo, también es capaz de reproducir AAC, WMA y sonidos MIDI. La pieza central del escudo es el chip *VS1053B Audio Codec IC*.

Tiene diferentes salidas de audio un conector Jack 3.5mm o los tres pines 'R', 'L' y '- '.

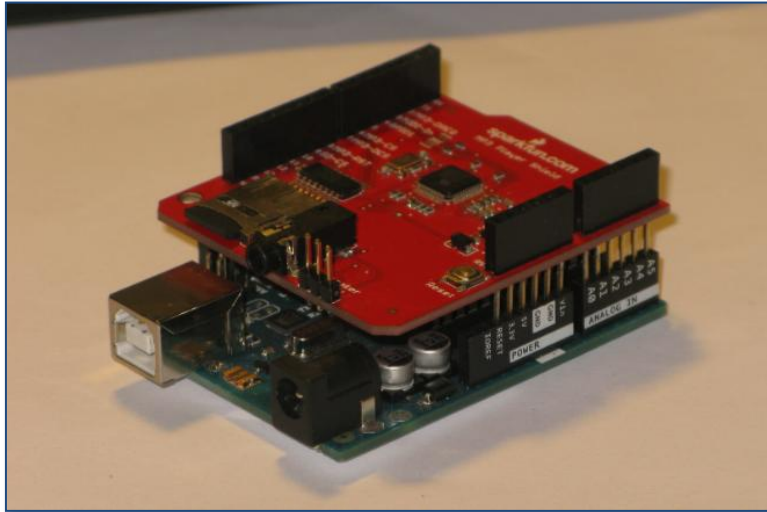


Figura 6.23 Arduino Uno – MP3 Player Shield

6.4.1 Conexionado

En la siguiente figura se pueden observar todos los pines que componen este escudo, de color azul los utilizados por el *VS1053 MP3 Codec IC*, con la etiqueta roja el usado para la tarjeta μ SD, y los morados son utilizados para ambos componentes, estos últimos son los comunes del protocolo SPI.

- **SCLK (13)**. Señal de reloj encargada de sincronizar los datos. (protocolo SPI).
- **MISO (12)**. Datos de salida del escudo en dirección al *master* (*Arduino*).
- **MOSI (11)**. Datos de entrada al escudo, procedentes del *master* (*Arduino*).
- **SD Card CS (9)**. Es el chip select de la tarjeta μ SD. Indica cuando debe leer o escribir la tarjeta en los canales MISO y MOSI.
- **VS1053 Reset (8)**. Señal que produce un reset en *VS1053*.
- **VS1053 Data CS (7)**. Avisa cuando el chip está disponible para recibir datos, o cuando está saturado.
- **VS1053 CS (6)**. Chip select del *VS1053*, indica que los datos que van por los canales MISO y MOSI son para él.
- **MIDI In (3)**. Para la gestión de sonidos MIDI, en nuestro proyecto no lo usamos.
- **Data Request (2)**. Pin de interrupción que indica al *Arduino* que el *VS1053* necesita más datos de música.

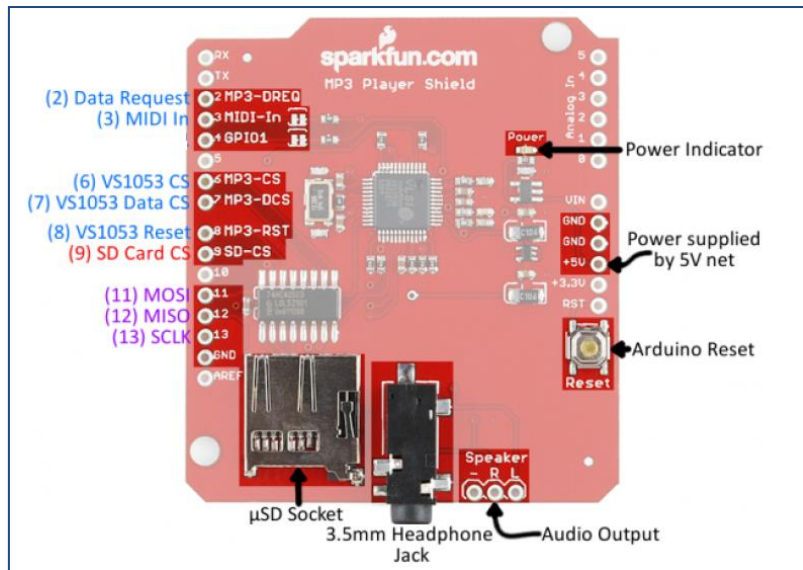


Figura 6.24 Conexión de MP3 Player Shield [11]

Quedan libres los siguientes pines:

- **Rx/Tx**, los cuales el Arduino usará para intercambiar datos con la otra placa Arduino.
- **D5 y D10**, serán usados para activar los LED de estado.
- **Pines Analógicos A0 –A5**, estos no serán usados en nuestro proyecto.

6.4.2 Puesta a punto del MP3 Player Shield

Lo primero que tenemos que comprobar es si nuestra tarjeta μ SD está en formato FAT16 o FAT32, si no es así deberemos formatear en uno de estos dos tipos.

A continuación, antes de cargar los archivos *.mp3* a la tarjeta tendremos que cambiar los nombres, la librería SDFat sólo admite nombres en formato “8.3” (nombre del archivo en 8 caracteres (punto) 3 caracteres). Hemos decidido usar el criterio de nombrar a las canciones como track001.mp3, track002.mp3, etc. Hay que tener en cuenta que los números siempre tienen que ser correlativos, porque la aplicación hace peticiones a reproducir a nombres de canciones incrementado ese número, si no son correlativos llegará una canción que hará una petición a reproducir y no existirá.

Otra cosa a hacer es etiquetar las canciones en título, artista y álbum, estos los devolverá una función de la librería del dispositivo, y serán enviados al dispositivo que esté conectado mediante Bluetooth, eso si no deben superar los 20 caracteres, en caso de que lo superen el Arduino sólo cogerá los 20 primeros caracteres.

Una vez copiados en la raíz de la tarjeta μ SD los archivos de música *.mp3*, en el nombre modificado, podemos introducir la tarjeta μ SD en el *Shield*.

6.4.3 Sketch – Parte Audio

En este apartado vamos a describir la parte de programa que realiza nuestro Arduino Uno, este se encarga de la reproducción de audio en formato mp3, este audio tiene que estar almacenado en la tarjeta μ SD del *Shield*.

```
Parte_Audio §
1  /*
2     Proyecto: Reproductor Arduino
3     Parte: Audio
4     Componente: Arduino UNO - MP3 Player Shield
5     Autor: Javier Colomer Barberá
6     Fecha: Julio 2015
7     Escuela: ETSIT
8
9     Descripción:
10    La parte de Comunicación (otro proyecto de Arduino MEGA) le envía los datos por el canal serie
11    Este los recibe, interpreta y actúa en consecuencia.
12  */
```

Figura 6.25 Cabecera sketch – Parte Audio

Como en cualquier proyecto lo primero que nos encontramos es la llamada a las librerías, en nuestro proyecto a la librería SPI, junto con las del fabricante *SdFat*, *SdFatUtil* y *SFEMP3Shield*. También declaramos los objetos de estas librerías, para más adelante poder hacer referencia a estas.

```
13
14  #include <SPI.h>
15    //Añadimos librerías SdFat
16  #include <SdFat.h>
17  #include <SdFatUtil.h>
18    //Añadimos librería MP3
19  #include <SFEMP3Shield.h>
20
21    //inicializo variables sd y MP3player
22  SdFat sd;
23  SFEMP3Shield MP3player;
24
```

Figura 6.26 Librerías y declaraciones

El siguiente punto es la declaración todas las variables globales, las que utilizamos para saber que canción tenemos que reproducir, los LEDs, variables para almacenar título, artista y álbum, etc. El listado completo de todas las variables que utilizamos lo podéis consultar en el código fuente.

En la sección del *setup()* se realizan las llamadas a la instrucción *begin()*, tanto del Serial como de la SD y del MP3Player. También incluimos el llamamiento a la función *calcula_canciones()*, está creada por nosotros, calcula el número total de canciones que hay en la tarjeta μ SD.

```

48
49 void setup(void)
50 {
51   Serial.begin(115200);
52   //Serial.println(F("Reproductor Arduino:"));
53   //Iniciación SDCard
54   if(!sd.begin(SD_SEL, SPI_HALF_SPEED)) sd.initErrorHalt();
55   // depending upon your SdCard environment, SPI_HAVE_SPEED may work better.
56   if(!sd.chdir("/")) sd.errorHalt("sd.chdir");
57
58   //Iniciación MP3 Player Shield
59   MP3player.begin();
60   MP3player.resumeDataStream();
61
62   menu('R'); //reset
63   continuo= false;
64   calculo_canciones(); //calcula el número de canciones que hay en la SDCard
65   pinMode(LED_reproduciendo, OUTPUT);
66   pinMode(LED_stop, OUTPUT);
67   digitalWrite(LED_stop, HIGH);
68   digitalWrite(LED_reproduciendo, HIGH);
69 } //final SETUP
70

```

Figura 6.27 Setup – Parte Audio

En la sección correspondiente al `loop()` hace diferentes cosas:

- Comprueba si le llegan datos por el Serial, si es así estos son comandos que provienen del otro Arduino. Entonces debe llamar a la función `menu()` con los parámetros que le lleguen, esta función la comentaremos más adelante.

```

71 void loop()
72 {
73   while (Serial.available()) {
74     comando = Serial.read(); //Todos los datos que van llegando del otro Arduino los mando como comandos a menu(comando);
75     menu(comando);
76   } //FINAL DEL WHILE
77

```

Figura 6.28 Llamamiento a la función `menu()`

- Comprueba el estado del módulo para activar los diferentes LEDs.

```

93   }else
94     if(MP3player.isPlaying()){ //si está reproduciendo
95       digitalWrite(LED_stop, LOW);
96       digitalWrite(LED_reproduciendo, HIGH);
97
98   }

```

Figura 6.29 Estado Reproduciendo

```

78
79   if(!MP3player.isPlaying() || MP3player.getState() == paused_playback){
80     //si no está reproduciendo...
81     digitalWrite(LED_reproduciendo, LOW);
82     digitalWrite(LED_stop, HIGH);
83

```

Figura 6.30 Estado Pausa/Stop

- Comprueba si ha llegado al final de una canción, entonces debe pasar a la siguiente o si ha llegado al final de la última canción se para y no sigue reproduciendo.

```

99   //Cuando termino una canción y no he llegado a la última paso a la siguiente.
100   if(!MP3player.isPlaying() && continuo == true && cancion < max_canciones){
101     Serial.print("SIGUIENTE");
102     delay(100);
103     menu('>');
104   }

```

Figura 6.31 Siguiete canción

```

83
84 // Para que enviar al movil "FINAL" debe cumplirse:
85 //no reproduciendo, ultima cancion, no este en pause o stop, es decir, se habra finalizado la cancion.
86     if(btn_stop == false && cancion == max_canciones && solo_una_vez == true)
87     {
88         Serial.print("FINAL");
89         delay(100);
90         solo_una_vez = false;
91     }
92

```

Figura 6.32 Ultima canción

6.4.3.1 Función `menu()`

Esta función recibe como parámetro de entrada un *char*, este es el comando que recibimos del otro Arduino conectado en serie. Dependiendo de lo que valga ese “comando” hacemos una cosa u otra. En este apartado vamos a comentar lo que hace en cada uno de los caso, la función completa se encuentra en el código fuente. A continuación el listado posibles comandos.

- ‘R’. Reseteamos el Shield
- ‘r’. Enviamos el número máximo de canciones que tiene la tarjeta μ SD.
- ‘0-9’. El número de canción. Para el cálculo de este hay que hacer algunas conversiones, y poner el carácter que corresponde en las unidades, decenas y centenas, antes de convertirlo a un entero. El número viene precedido por una letra ‘F’ o ‘I’, para saber dónde termina.
 - Si es ‘F’ después de calcular la canción llama a `menu('f')`.
 - Si es ‘I’ después de calcular la canción llama a `menu('i')`.
- ‘f’. Pone a play. Si anteriormente no estaba en pause ni en stop, pone a reproducir y devuelve los datos de título, artista y álbum. Si está en pause o en stop pone a reproducir pero no devuelve los atributos
- ‘i’. información. Devuelve título, artista y álbum.
- ‘s’ stop. Pone la reproducción en parada.
- ‘p’. pause. Pone la reproducción en pausa.
- ‘+’ más volumen. Sube el volumen de salida del *Shield* y devuelve los dB que esta.
- ‘-’. menos volumen. Baja el volumen del *Shield* y devuelve los dB que se encuentra.
- ‘>’. siguiente. Pasa a la siguiente canción, y devuelve el título, artista y álbum.
- ‘<’. anterior. Pasa a la canción anterior en número, y devuelve el título, artista y álbum.

6.4.3.2 Función `calculo_canciones()`

Esta recorre todos los archivos .mp3 que hay en la raíz de la tarjeta μ SD, los va abriendo y cerrando e incrementado el contador.

Esta sirve para saber cuál va a ser el número máximo de canción al cual puedo acceder siguiendo la nomenclatura `track001.mp3`.

6.4.3.3 Función `convertir()`

La llamada a esta función se realiza para descomponer a partir de un número las unidades decenas y centenas que tiene este. Así poder añadirlo al nombre de la canción en los caracteres diferenciados.

Como parámetro se le pasa el entero a descomponer.

El código fuente de esta parte del programa con todas las funciones, se puede encontrar en el siguiente enlace.

<https://goo.gl/eHCd4W>

6.4.4 Esquema Parte Audio

Teniendo la parte de Audio finalizada podemos comprobar su funcionamiento conectando unos auriculares/altavoces a la salida Jack 3.5mm, y enviando los comandos mediante el monitor serie del entorno de desarrollo.

A continuación tenemos el esquema del montaje que corresponde a esta parte, también hemos añadido unas resistencias de $1K\Omega$ antes de los LEDs.

Aunque en la siguiente figura se observa el *Shield* cableado esto no es así, ya que le hemos soldado unos pines y encaja en la parte superior de Arduino sin ningún problema.

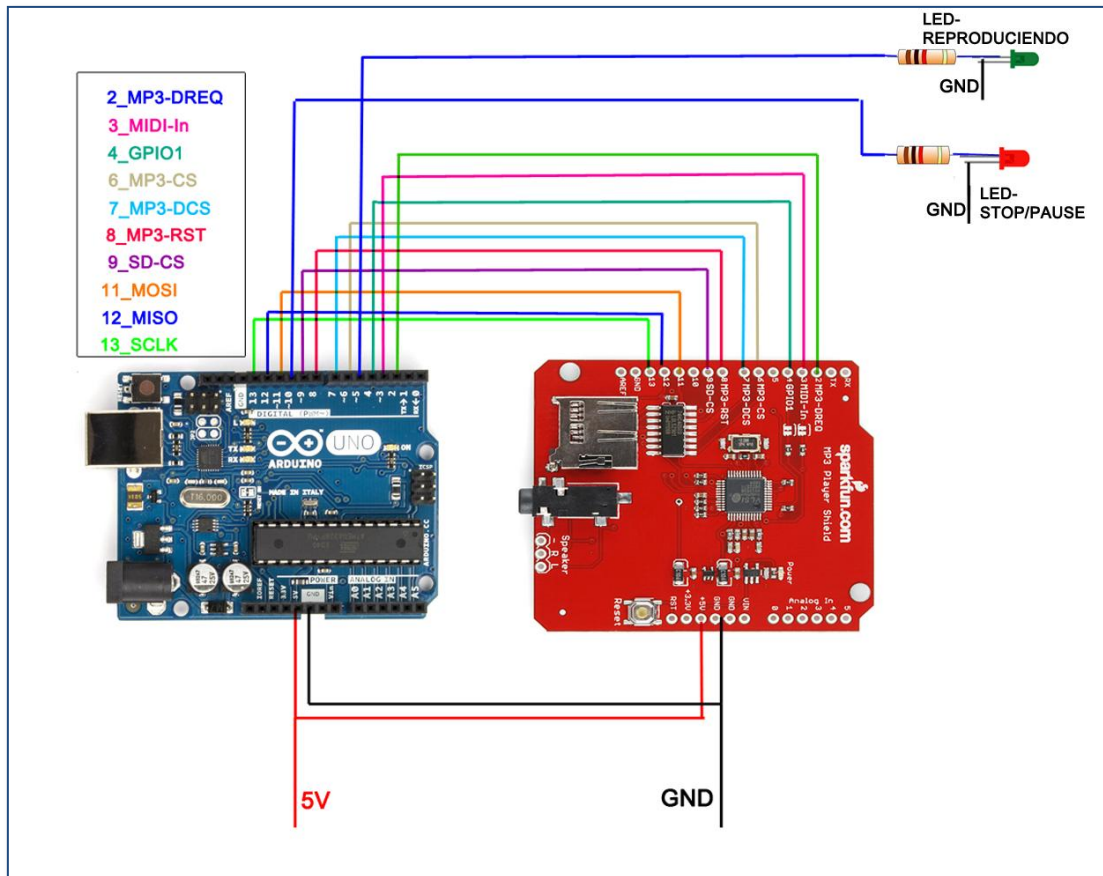


Figura 6.33 Montaje Arduino Uno con MP3Player Shield

6.5 Conjunto Parte Arduino

En este punto vamos a ver las dos partes de Arduino explicadas anteriormente, parte comunicaciones y parte de audio. Como hemos conectado en apartados anteriores estas se comunican por puertos serie, para el Arduino Mega el puerto *Serial1* (pines 18 y 19), con el Arduino Uno por el puerto *Serial* (pines 0 y 1). Los pines tienen que estar cruzados, es decir, el *18Tx1-0Rx* y el *19Rx1-1Tx*.

Otra cosa a tener la cuenta es que la **GND** (masa) tiene que ser común a los dos Arduinos y a todos los dispositivos que conectemos a este prototipo. A esta misma premisa también se debe unir la señal *Vin* que contendrá 5 voltios, mediante un único punto de alimentación.

Para la salida del audio hemos empleado las salidas ‘R’ y ‘L’, están conectadas a un amplificador el *TPA2005D1*, en un principio este amplificaba la señal al doble, pero poniéndole unas resistencias en paralelo en el zócalo preparado para lo mismo, hemos conseguido una mayor amplificación. [12]

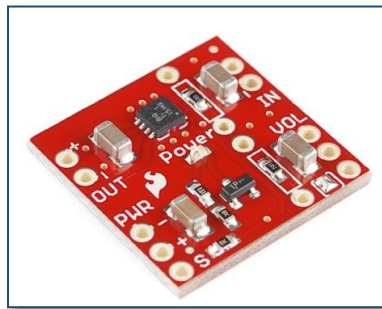


Figura 6.34 Amplificador *TPA2005D1* [12]

Al añadir la resistencia en paralelo se cumple la siguiente ecuación:

$$\text{Amplificación} = 2 * \left(\frac{150k}{R}\right)$$

En nuestro proyecto hemos introducido una resistencia de 56k Ω , esto supone una amplificación de aproximadamente 6 veces la entrada.

La salida de amplificador se conecta dos altavoces de 5 cm de diámetro de 8 ohmios y 0.5W los cuales radian el sonido hacia el exterior, como los que se muestran en la *Figura 6.35*.



Figura 6.35 Altavoz 8 Ohmios 0.5W

En el siguiente *Figura 6.36 Esquema completo Reproductor Arduino*, se encuentra todo el conexionado de prototipo final del reproductor.

En este podemos observar como al tener un *Shield* que se acopla a los pines de la placa Arduino hemos ahorrado mucho en espacio, y en el cableado de los mismos, podríamos haber hecho lo mismo en la antena Bluetooth, poniéndole una placa *Proto Shield*.

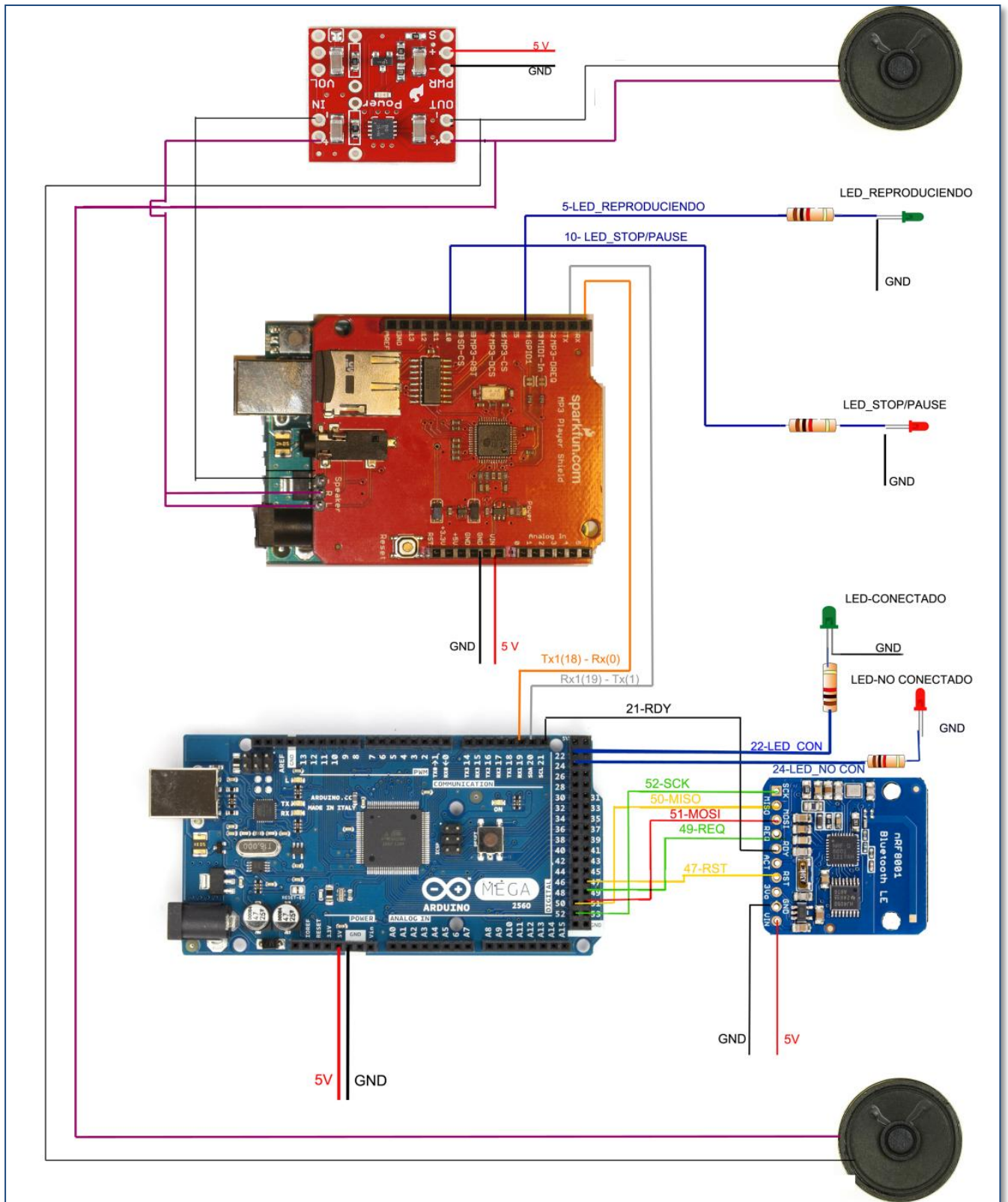


Figura 6.36 Esquema completo Reproductor Arduino

Capítulo 7. Reproductor Arduino

En este capítulo vamos a ensamblar el funcionamiento global del proyecto, juntando la aplicación Android con el montaje final de Arduino.

Vamos a empezar por explicar todos los casos de uso, es decir, las cosas que van a ocurrir en cualquier punto del funcionamiento. Y como responde la aplicación a estos sucesos.

7.1 Casos de uso

En todos los casos de uso vamos a ver el intercambio de paquetes entre el dispositivo con sistema operativo Android y los dos Arduinos. El que aparece enmarcado en amarillo es el empieza la acción en cada uno de los casos.

7.1.1 Establecimiento de la conexión

Cuando en el dispositivo Android nos disponemos a conectarnos, se produce un intercambio de paquetes para establecer la conexión, todo esto es transparente a nuestra gestión, ya que esto lo realizan las librerías.

Una vez establecida la conexión, el Arduino Mega avisa al Arduino Uno que se ha establecido una conexión, este calcula el número total de canciones que tiene la tarjeta μ SD, y le devuelve ese número para que el dispositivo Android pueda inicializar cual va a ser la canción máxima.

También cabe recordar que el conjunto de botones de la aplicación solo serán activos cuando se haya producido la conexión, si estas desconectado solo está habilitado el botón de conectar.

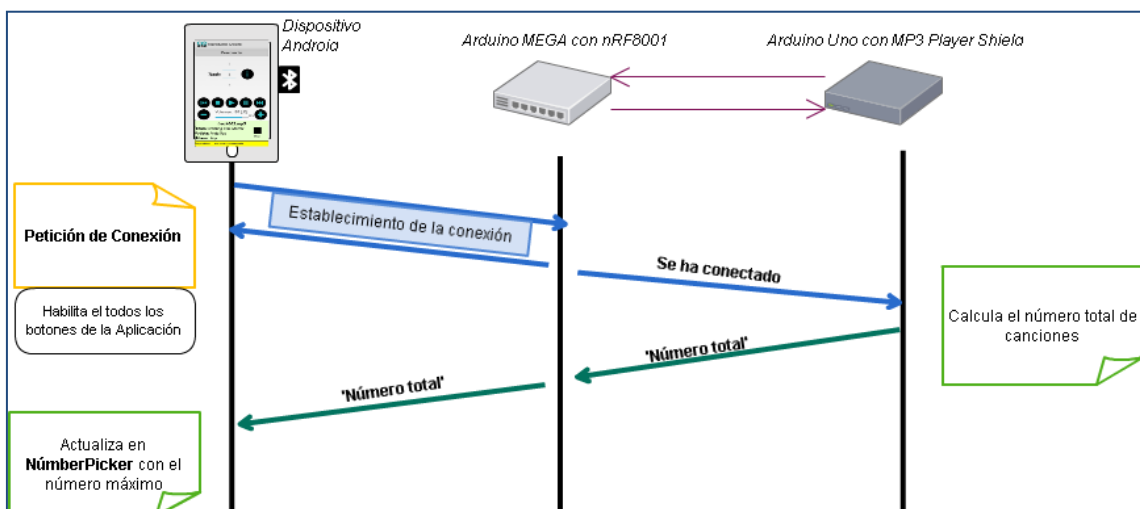


Figura 7.1 Establecimiento de la conexión

7.1.2 Desconexión

Cuando se pulsa el botón de “Desconectar”, la comunicación con el Arduino Mega se cierra, y este avisa al Arduino Uno, y le ordena que se resete, así paramos el funcionamiento de MP3 Player Shield.

Esto también ocurre cuando nos separamos demasiado del Arduino y la conexión se rompe inesperadamente. O por algún u otro fallo de comunicación.

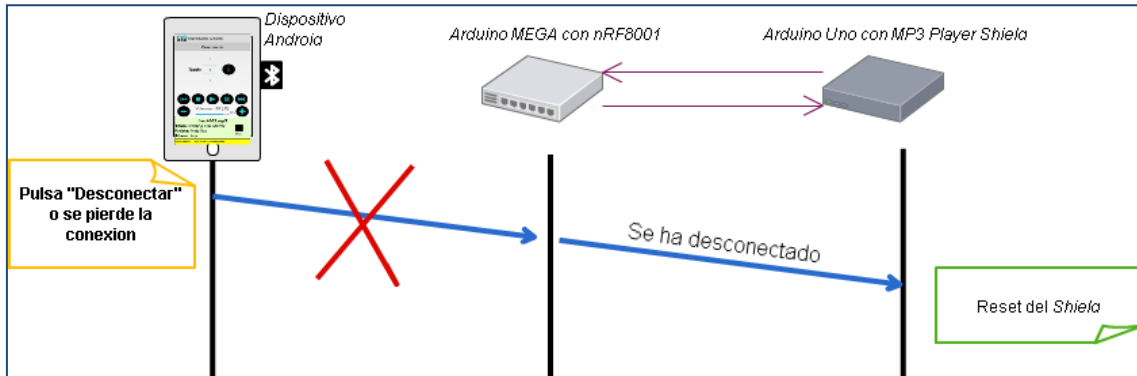


Figura 7.2 Desconexión

7.1.3 Poner a reproducir

Este caso se inicia cuando se pulsa el botón “play” en el dispositivo Android, este coge el número que tiene en el *Numberpicker* le añade una ‘F’ y lo envía. Se activan las variables para la recepción de los atributos.

El Arduino Uno pone a reproducir el MP3 Player Shield, y le devuelve los atributos de título, artista y álbum al dispositivo Android para que pueda actualizar.

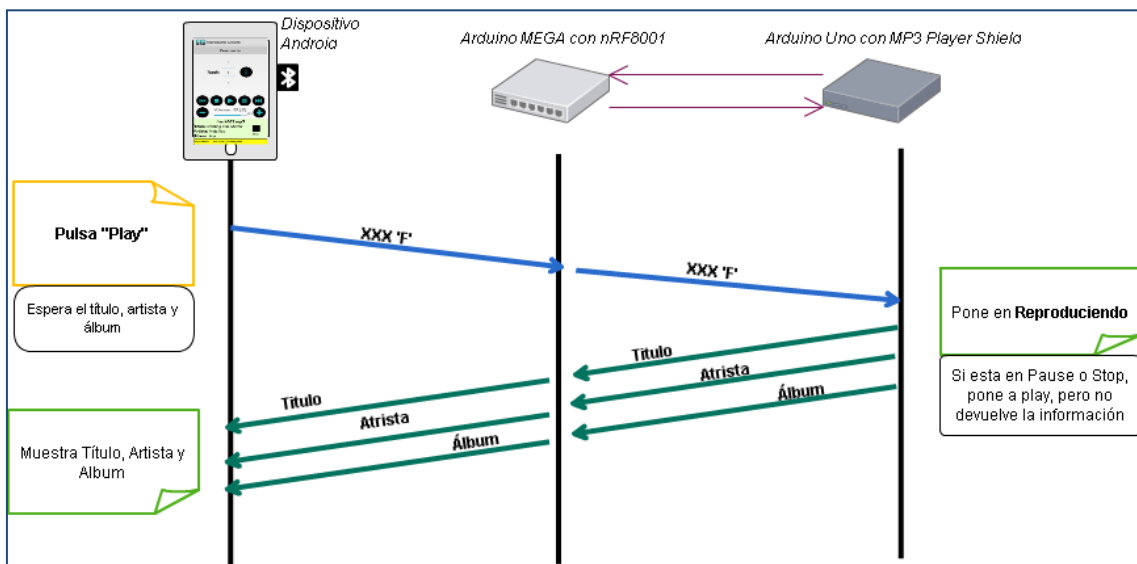


Figura 7.3 Poner a reproducir

7.1.4 Petición de información

Cuando se solicita la información sobre una canción, se envía el número de canción sobre la que solicitas la información, esto precedido de una 'I', el número lo coge del *NumberPicker*. El Arduino Uno le devuelve los atributos de la canción (título, artista y álbum). Esto sólo sucederá si la canción que requerimos la información no es la que esta reproduciéndose.

Para realizar esto el Arduino Uno tendrá que poner play durante unos pocos segundos, inmediatamente pasara a stop, el play es necesario porque para conocer los atributos de una canción tiene que ser la que esta seleccionada. Pero de esto ni nos percataremos, porque no llega a sonar.

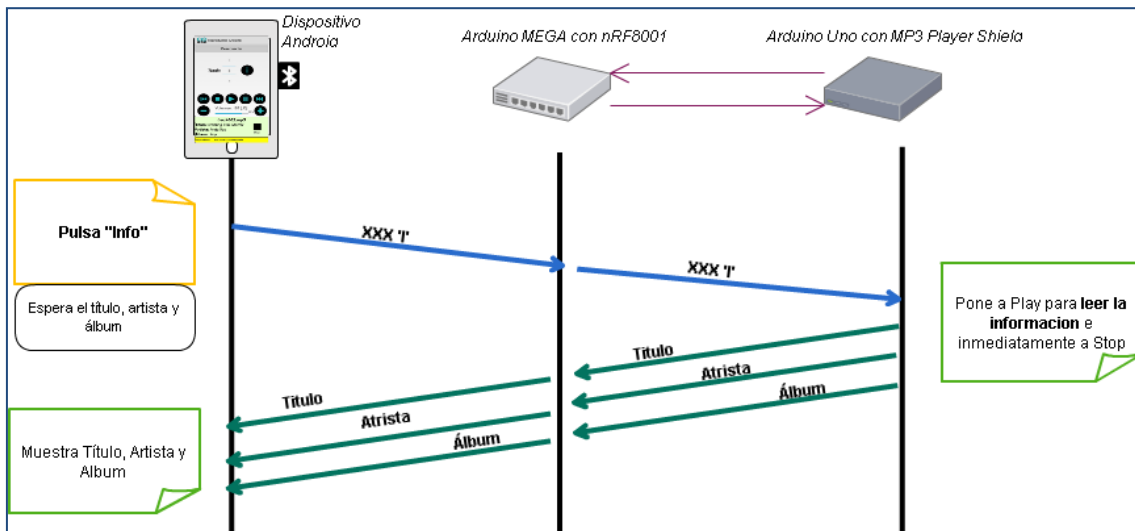


Figura 7.4 Solicitud de información

7.1.5 Poner en pausa la reproducción

Cuando se pulse el botón de "pause", enviamos el comando 'p', y cuando este llegue al Arduino Uno la reproducción quedara pausada.

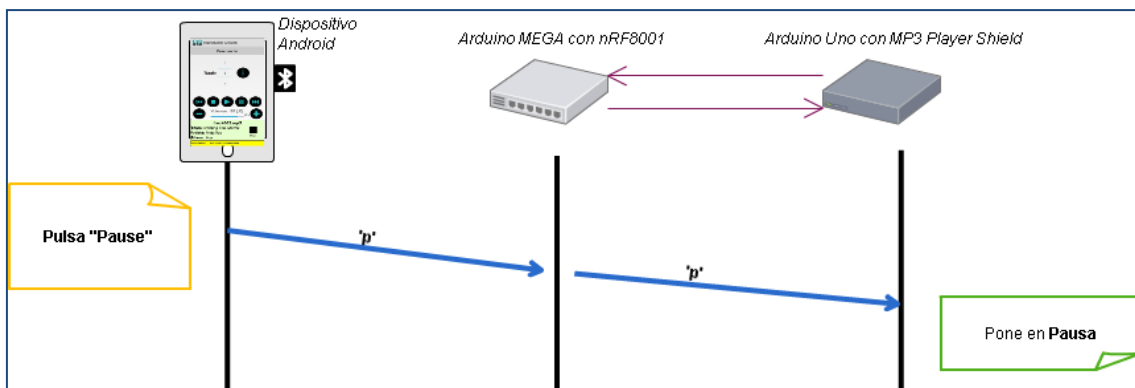


Figura 7.5 Pausar la reproducción

7.1.6 Poner en parada la reproducción

Cuando se pulse el botón de “stop”, se envía el comando ‘s’ pasara de un Arduino a otro y la reproducción quedara parada.

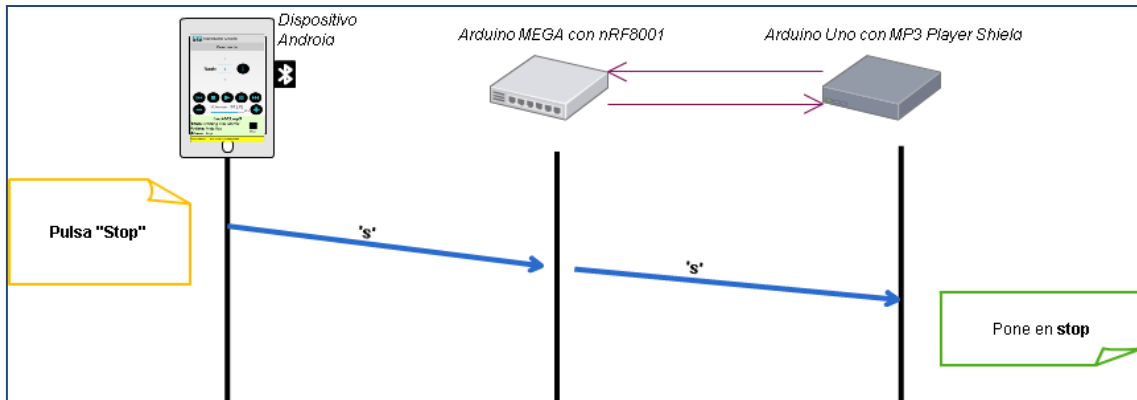


Figura 7.6 Parar la reproducción

7.1.7 Petición de pasar a la siguiente/anterior canción

Al pulsar el botón de “siguiente”, el dispositivo Android envía el comando ‘>’, este llega hasta el Arduino Uno, este interpreta el comando en pasar a la siguiente canción. Y le devuelve los atributos de la canción que pasa a reproducir.

En el caso de pulsar el botón de “anterior” realiza lo mismo pero decrementando el número de canción a reproducir.

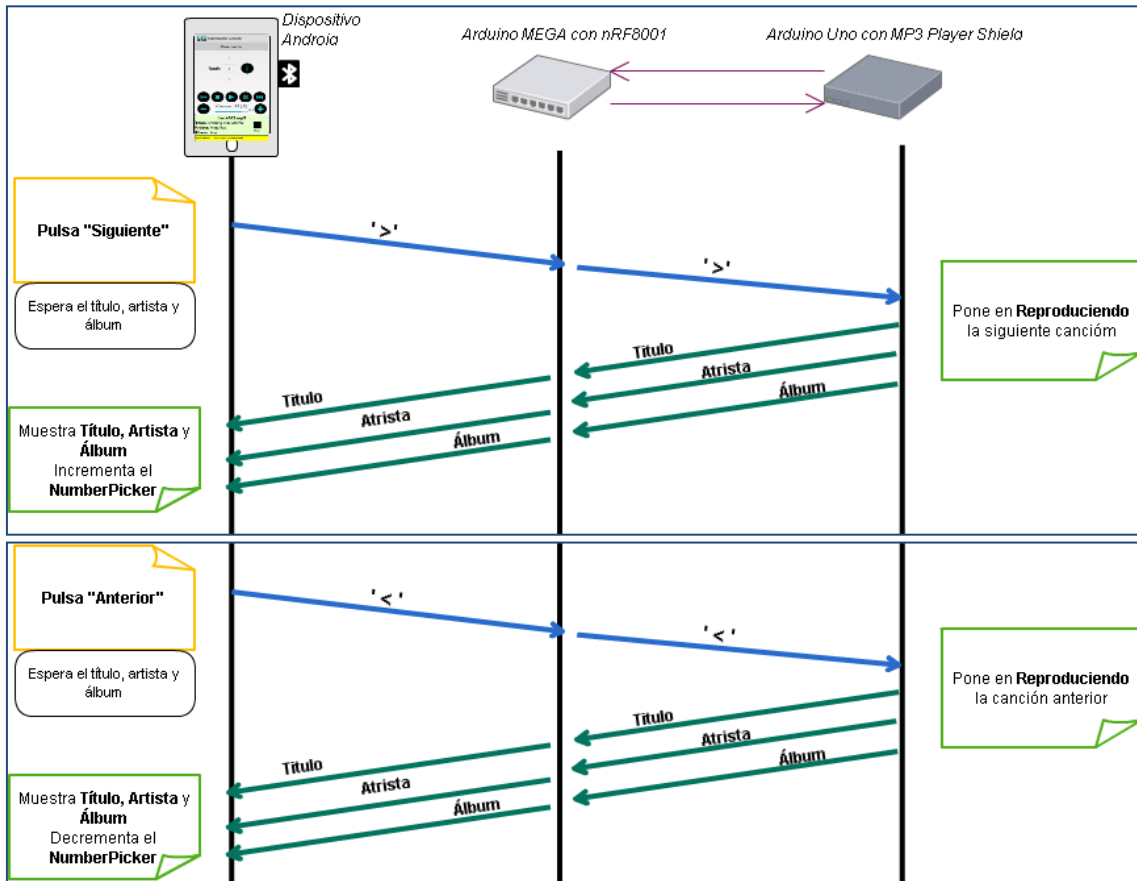


Figura 7.7 Avance o retroceso de canciones

7.1.8 Petición de incrementar/decrementar el volumen

Al pulsar los botones de más volumen o menos, se envía un '+' o '-' respectivamente. Esto llega al Arduino encargado del audio y lo interpreta en lo que corresponda, aumentar o decrementar. Este devuelve el valor del volumen en dB el cual el dispositivo Android recibe e interpreta el porcentaje de volumen, teniendo en cuenta que -1dB es el máximo que puede dar.

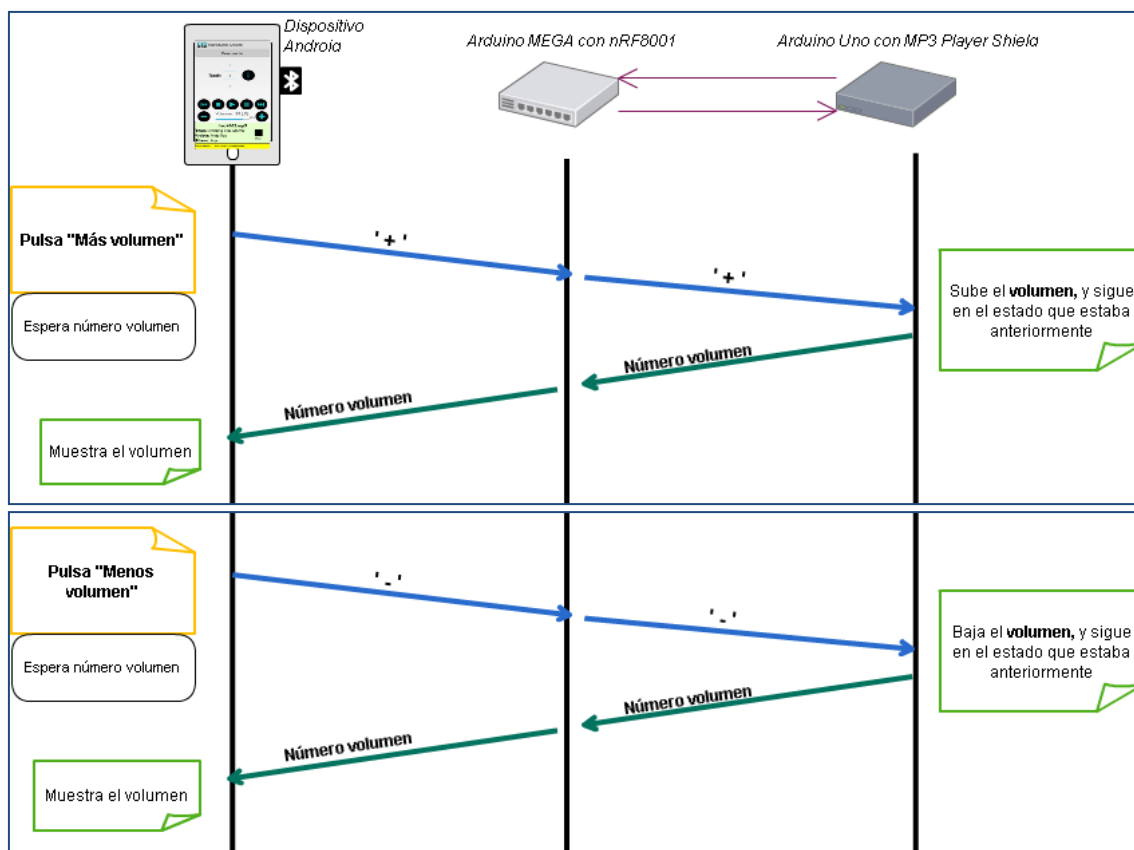


Figura 7.8 Gestión del volumen

7.1.9 Reproducción continua

Cuando una canción finaliza automáticamente pasa a reproducir la siguiente canción, entonces tiene que avisar al dispositivo Android que esto ha sucedido, para que incremente en *NumberPicker*, y se ponga a la espera de los atributos de la nueva canción.

Esto lo hace enviando el *String* "SIGUIENTE" entonces el Android sabe que ha cambiado de canción y que a continuación va a recibir los atributos de la nueva canción.

Así hasta que llegue a la última canción, cuando llegue a esta última sucederá lo que se explica en el siguiente caso.

Como se puede apreciar en la siguiente figura, en este caso la comunicación la empieza el Arduino, por eso su recuadro está en amarillo.

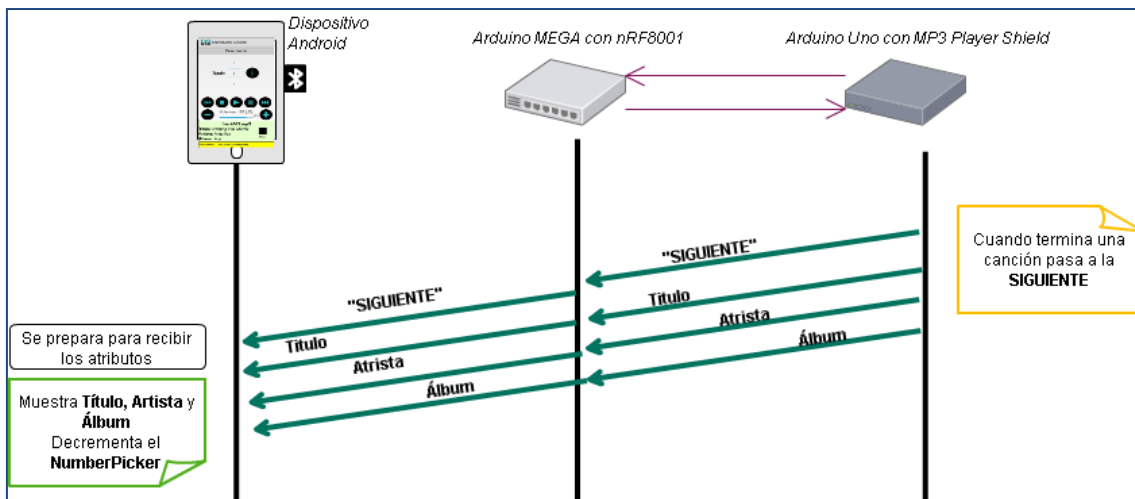


Figura 7.9 Reproducción continua

7.1.10 Finalizado todas las canciones

Como hemos comentado anteriormente, cuando llega a la última canción no sigue reproduciendo. Pero tiene que avisar al dispositivo Android que ha finalizado de reproducir. Esto lo hace enviando el *String* "FINAL".

Para enviar esto debe comprobar que se encuentre en la última canción y que no esté reproduciendo, pero si no está reproduciendo también debe de comprobar que las causas no sean por estar en *pause* ni en *stop* sino porque ha finalizado la canción.

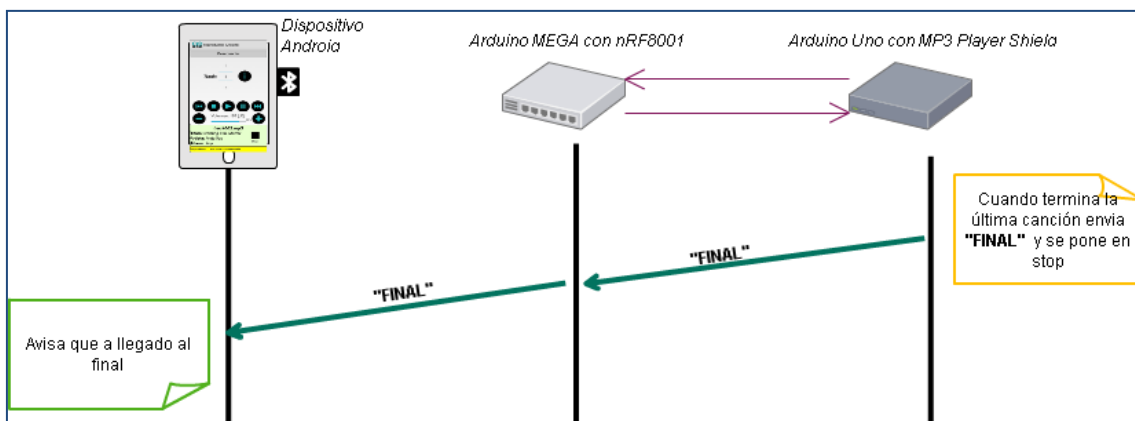


Figura 7.10 Última canción terminada

7.2 Repositorio

En el siguiente enlace estará almacenada la aplicación Android durante todo el 2016, para que cualquiera pueda descargar dicha aplicación y probar su funcionamiento.

También podréis encontrar los programas tanto del Arduino Mega como del Arduino Uno, para que así se pueda realizar el montaje completo y probar el correcto funcionamiento del proyecto.

[Enlace de programas y documentación](https://goo.gl/ivgR5r)

<https://goo.gl/ivgR5r>



7.3 Apariencia del Proyecto

En la siguiente figura se puede observar el montaje final de nuestro proyecto.

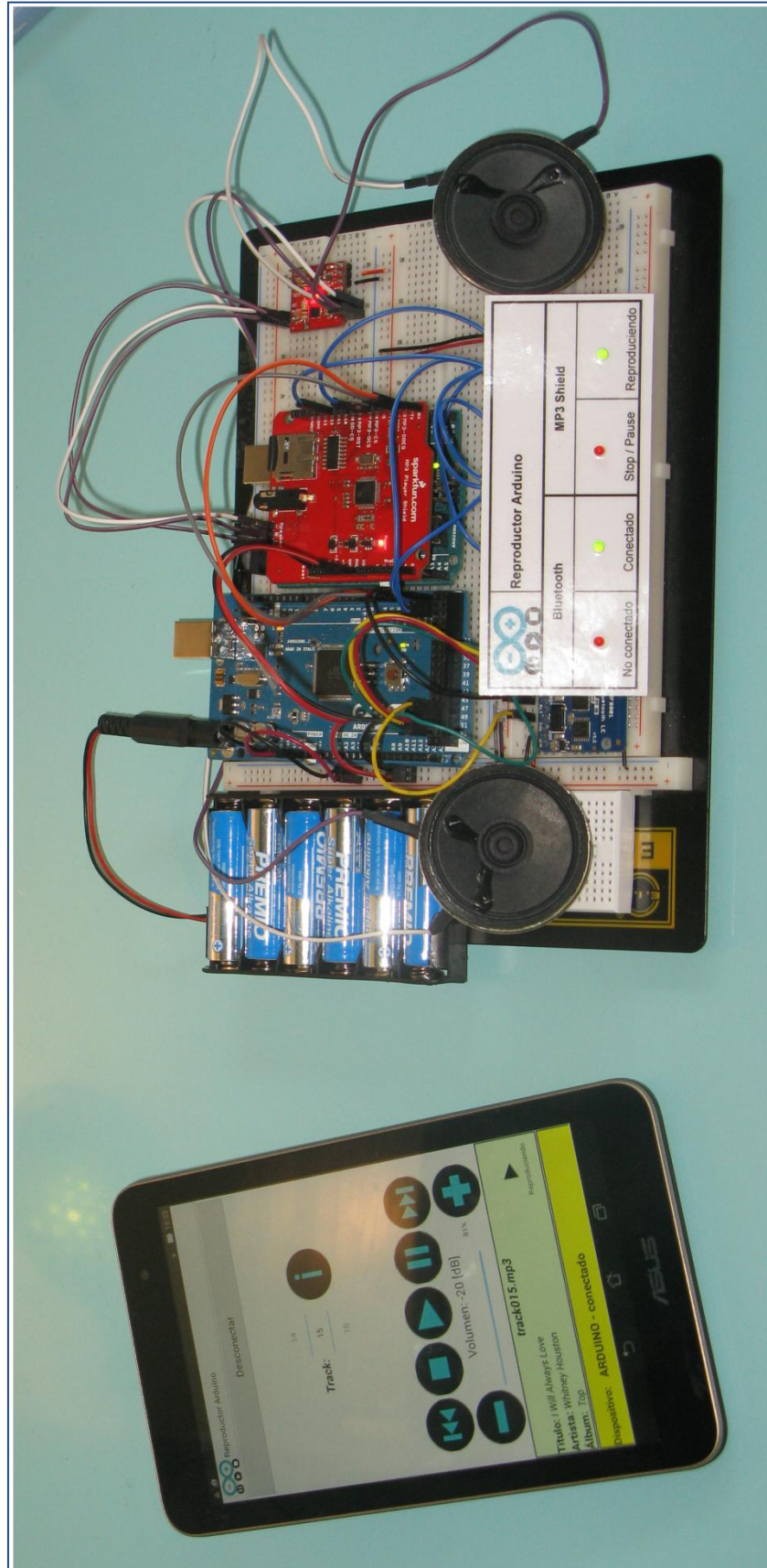


Figura 7.11 Proyecto

Capítulo 8. Presupuesto

Para la realización del proyecto se han empleado diferentes bienes, tanto económicos para conseguir los componentes hardware, como el tiempo empleado en la realización.

En la siguiente tabla se enumeran los bienes materiales que se han necesitado junto con su precio aproximado en el mercado.

Componente	Cantidad	Precio orientativo (la unidad)
Arduino Uno R3	1	24,95 €
Arduino Mega 2560 R3	1	47,19 €
MP3 Player Sheild	1	31,85 €
Bluetooth nRF8001	1	27,88 €
Altavoz 5cm 8 Ohm - 0.5W	2	2,90 €
Amplificador TDA2005D1	1	7,60 €
Cables / pines	10	0,30 € Aprox.
Resistencias / LEDs	4	1,00 € Aprox.
Total de gastos economicos		152,27 €

Tabla 8.1 Tabla de materia con precios

Otro tema a tener en cuenta en este apartado es el tiempo empleado para la realización del proyecto. Se han utilizado aproximadamente 600 horas, incluyendo los percances que hemos tenido. Si valoramos las horas de trabajo como si de un empleo se tratara y pudiéramos remunerarlas a 6€ por hora, esto sumaría un total de 3600€.

El tiempo de la realización para futuros proyectos quedaría más reducido, ya que en este hemos empleado bastante tiempo en el aprendizaje.

Juntando los bienes materiales y de las horas trabajadas, el proyecto en completo podría estar valorado en 3.800€ aprox.

El tener solo estos costes es posible gracias a que el coste del software de programación es libre y gratuito, ya que en muchos casos una parte importante del presupuesto debería asociada a la compra de licencias para software, cosa que en Android y Arduino no sucede.

Capítulo 9. Propuestas futuras

Una vez finalizado el proyecto, se pueden ver como este aun tienen diferentes campos en los cuales se podría ir mejorando la aplicación. A continuación voy a nombrar algunas de las cosas que se le podrían añadir a este.

En la parte de conexión, podríamos añadir nuevos métodos de conexión, ya que usamos placa Arduino Mega, y sigue teniendo puertos UART de comunicaciones libres.

Por ejemplo, podríamos añadir es el módulo Wi-Fi (*WIFI Serial Transceiver Module-ESP8266*), este se comunica por serie y no por protocolo SPI, es decir evitaríamos problemas de incompatibilidad por uso de todo el canal. También tendríamos que añadir a la aplicación Android código para permitir conectar con Wi-Fi. Con esta modificación ampliaríamos el radio de alcance del Arduino, es decir, podría estar controlado desde un sitio más alejado.



Figura 9.1 Antena Wi-Fi ESP8266

Otra ampliación con respecto a las comunicaciones sería el añadirle el *Shield Ethernet* y dejar el Arduino conectado a la red escuchando en una dirección IP, y desde tu dispositivo móvil poder manejarlo a través de Ethernet. En esta ampliación el proyecto podría estar controlado desde cualquier punto con conexión a internet.

En lo que respecta a la parte de reproducción una posible ampliación del proyecto sería realizar una segunda pantalla en la aplicación Android simulando un teclado de piano, y el *MP3 Player Shield* mediante comandos MIDI, reproducir los sonidos del piano. O hacer otro uso de los sonidos MIDI que contiene la tarjeta.

Capítulo 10. Conclusiones

En líneas generales podemos decir que el objetivo principal del trabajo se ha cumplido, y la valoración de este es positiva, ya que superando algunos contratiempos hemos creado un Reproductor de Audio para Arduino.

Para la realización de éste se han necesitado al menos nociones previas de los diferentes lenguajes de programación empleados, tanto de Android como de Arduino, e incluso algunos conocimientos de la electrónica.

Uno de los aspectos más positivos ha sido el aprender a actuar ante un imprevisto en la realización de este, buscando alternativas y posibles soluciones para el correcto funcionamiento, tomando la decisión que creemos que es la más apropiada. Esto hace referencia a lo ocurrido con las colisiones en el protocolo SPI.

Si hubiera que rediseñar de nuevo el sistema, muy probablemente cambiaría algunas cosas, como la elección de los dispositivos externos de Arduino. Elegiría uno con protocolo SPI y otro que usara otro protocolo diferente como puede ser el Serie por UART.

En conclusión, la realización del TFG ha sido una experiencia positiva, que me ha permitido aprender a programar en diferentes entornos y llevar a cabo la realización de un proyecto de principio a fin, tanto en aspectos de planificación como de implementación.

Bibliografía

Libros consultados para la creación de Trabajo Fin de Grado:

- [1] LEE, W. 2012. *Android 4: desarrollo de aplicaciones*. 1da. ed. ANAYA Multimedia/WROX. Ficha disponible en Internet: <http://www.anayamultimedia.es/libro.php?id=2957992>
- [2] TOMÁS, J. 2015. *El gran libro de Android*. 4da. ed. MARCOMBO, S.A. Ficha disponible en Internet: http://www.marcombo.com/El-gran-libro-de-android_isbn9788426718327.html
- [3] TOMÁS, J. et al. 2015. *El gran libro de Android avanzado*. 2da. ed. MARCOMBO, S.A. Ficha disponible en Internet: http://www.marcombo.com/El-gran-libro-de-android-avanzado_isbn9788426720788.html
- [4] TORRENTE, O. 2015. *Arduino: curso práctico de formación*. 1da. ed. Alfaomega, RC Libros. Ficha disponible en Internet: <http://www.alfaomega.com.mx/default/arduino-curso-practico-de-formacion-4791.html>
- [5] RIBAS, J. 2015. *Desarrollo De Aplicaciones Para Android - Edición 2016 (Manuales Imprescindibles)*. 1da. ed. ANAYA Multimedia. Ficha disponible en Internet: <http://www.anaya.es/fichaGeneral/ficha.php?obrcod=3273953>
- [6] RIBAS, J. 2015. *Arduino práctico (Manuales Imprescindibles)*. 1da. ed. ANAYA Multimedia. Ficha disponible en Internet: <http://www.anayamultimedia.es/libro.php?id=3273803>

Páginas web de consulta:

- [7] ADAFRUIT EDUCATORS (<http://www.adafruit.com/educators>). Documentación acerca de la antena Bluetooth nRF8001. En: *Adafruit* — 26 Jun 2015 22:05. Disponible en Internet: <https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout>.
- [8] ANDROID TEAM. Documentación oficial sobre las librerías de Android. En: *Android 5.1 r1* — 24 Jun 2015 22:05. Disponible en Internet: <https://developer.android.com/reference/packages.html>
- [9] ARDUINO TEAM (<https://www.arduino.cc/en/Main/AboutUs>). Página oficial de Arduino. En: *Arduino* — 26 Jun 2015 22:05. Disponible en Internet: <http://www.arduino.cc/>
- [10] SPARKFUN EDUCATION DEPARTMENT (<https://learn.sparkfun.com/contact>). Tutoriales para los productos de *sparkfun*.. En: *sparkfun* — 26 Jun 2015 22:05. Disponible en Internet: <https://learn.sparkfun.com/tutorials>.
- [11] SPARKFUN EDUCATION DEPARTMENT (<https://learn.sparkfun.com/contact>). Información sobre el MP3 Player Shield de *sparkfun*.. En: *sparkfun* — 26 Jun 2015 22:05. Disponible en Internet: <https://learn.sparkfun.com/tutorials/mp3-player-shield-hookup>.
- [12] SPARKFUN EDUCATION DEPARTMENT (<https://learn.sparkfun.com/contact>). Información del modulo amplificador de *sparkfun*.. En: *sparkfun* — 26 Jun 2015 22:05. Disponible en Internet: <https://www.sparkfun.com/tutorials/392>.
- [13] TUTORIALSPPOINT.COM. Ejemplos de implementación de los métodos. En: *TUTORIALSPPOINT.COM* — 26 Jun 2015 22:05. Disponible en Internet: <http://www.tutorialspoint.com/android/>