PROYECTO FINAL DE GRADO

# Diseño de aplicación móvil para la comunicación inalámbrica de señales audiovisuales

Grado en Ingeniería de Sistemas Audiovisuales


**Proyectista** **Alberto Esteban Pérez**
**Director** **Raúl Fernández García**

**Junio 2013**

# DISEÑO DE APLICACIÓN MÓVIL PARA LA COMUNICACIÓN INALÁMBRICA DE SEÑALES AUDIOVISUALES

Autor: Alberto Esteban Pérez teby182@gmail.com

Director: Raúl Fernández García

Presidente: Ignacio Gil Galí

Vocal: Ignasi Esquerra Llucia

# AGRADECIMIENTOS

En primer lugar me gustaría dar las gracias a mis padres y mi hermana, por toda la paciencia que han tenido y el ánimo que me han dado durante toda la trayectoria de la carrera. Gracias a ellos estoy aquí.

También quiero agradecer a mis amigos y en especial a Juan, que me ha ayudado mucho a la hora de desarrollar este proyecto, sin él no hubiera sido posible.

Finalmente quiero dar gracias a todos los profesores que me han formado durante estos años, y en especial al director del proyecto, Raúl Fernández García, por el tiempo y la ayuda prestada.

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1. INTRODUCCIÓN

La evolución de la tecnología en los últimos años ha crecido de manera exponencial desde la aparición de los ordenadores de sobremesa, pero esta evolución todavía ha sido más rápida con la aparición de los primeros smartphones hace unos años. Hace diez años, nadie creía que podríamos llegar a tener los ordenadores en la palma de nuestra mano, hemos pasado de llamar y poca cosa más a tener el smartphone en nuestro día a día, pudiendo desde navegar por internet, a entrar en las redes sociales, o enviar y recibir archivos entre miles de personas de todo el mundo.

Por otra parte, no sólo la tecnología es la que avanza, sino que el mundo de Internet ha evolucionado desde el punto de buscar una información en una web a estar todos conectados con la creación de las redes sociales y tener siempre información disponible. Dados estos dos puntos, es normal que cada vez estemos más interconectados entre nosotros.

## 1.1 MOTIVACIÓN

La idea de este proyecto surgió por varias circunstancias. A punto de finalizar la carrera me apetecía realizar un PFC que me forzará a aprender cosas nuevas y a aplicar parte de lo aprendido durante estos años. Gracias a un par de asignaturas de la carrera, despertó en mí una afición por los módulos del hardware libre Arduino. Y viendo que la forma más fácil y documentada que hay para crear aplicaciones para Smartphones, es Android, me decanté por integrarlo en el proyecto.

Si mezclamos todo esto, quise realizar algo de lo que hubiera poca documentación, de hecho, me estuve informando por muchos foros de temas relacionados y no encontré mucho. Por eso quiero hacerlo, para que la gente que pueda tener inquietudes con el tema le sea más fácil guiarse.

## 1.2 OBJETIVOS

El principal objetivo del presente proyecto es el diseño de una aplicación Android que mediante un módulo Arduino conectado por Ethernet a un router, permita la transmisión

de archivos audiovisuales inalámbricamente. El sistema contara con la aplicación Android jugando el roll de cliente y el módulo Arduino de servidor.

### 1.2.1 OBJETIVOS DEL PROYECTO

La aplicación que se quiere llevar a cabo con el presente proyecto, se puede enfocar a muchos ámbitos, desde el turismo, la publicidad, o incluso la seguridad. Las ventajas que obtendríamos serían tener un sistema privado de transferencia de archivos, seguro y a un bajo coste, ya que, hoy en día, las placas Arduino están a un precio muy atractivo.

Para su realización, los pasos a seguir son los siguientes:

- Programar el sketch del módulo Arduino para que actúe como un servidor, leyendo de la SD y comunicándose con el cliente.
- Diseño de la aplicación Android: su apariencia.
- Programar la aplicación Android para que se comunique con el servidor y guarde datos en la SD y/o muestre los datos por pantalla.
- Diseñar un protocolo de envío y recepción para que cliente y servidor se entiendan en la transferencia.

### 1.2.2 OBJETIVOS PERSONALES

- Aprendizaje del lenguaje de programación JAVA.
- Aprendizaje en el diseño de aplicaciones Android.
    - Utilización del IDE NetBeans en conjunto con la SDK de Android
- Profundizar en el lenguaje de programación Arduino (Processing)
- Afrontar un problema que combine hardware + software.
- Profundizar en protocolos de transmisión TCP/IP

# 2. ESTADO DEL ARTE

Es necesaria una fase inicial de análisis para determinar el alcance, objetivo, requisitos del proyecto. También habrá que hacer una selección de hardware y herramientas de software del que se hará uso. Por ello, en esta primera parte, se va a abordar la fase previa del desarrollo. Se pondrá atención en protocolos de transmisión y en diferentes *Open Source Hardware* que podrían operar con dichos protocolos. Esto nos ayudará durante el análisis y desarrollo de nuestra idea y así llegar a la decisión más eficiente y económica.

## 2.1. PROTOCOLOS DE TRANSMISIÓN

Los protocolos principalmente son las reglas q se deben de seguir para poder realizar una operación o transmisión correctamente. Y funcionan de la siguiente manera:

Cuando se intenta hacer una transferencia de datos los protocolos se encargan de separar en partes pequeñas toda la información y encapsularlas en grupos para posteriormente proceder a enviarlas. Cuando se logra enviar el paquete de datos, el receptor debe enviar un mensaje de respuesta para verificar que la transmisión se haya realizado correctamente. Si el paquete de datos enviado no llega completo o llega dañado el receptor manda un mensaje de error para que el emisor reenvié el paquete de datos cuando el proceso se completa el emisor procede a enviar el segundo paquete de datos y así sucesivamente hasta terminar de enviar todos los datos. El receptor al terminar de recibir los datos los empieza a agruparlos para poder reconocer los datos enviados.

Una vez hecha esta breve introducción, se explicará los diferentes protocolos de transmisión que hay y los que más podrían ser de utilidad.

### 2.1.1. BLUETOOTH

Protocolo de transmisión inalámbrica que permite enviar y recibir datos y voz entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz, cuando se encuentran en un área reducida. Ofrecen la posibilidad de crear

pequeñas redes inalámbricas. Está diseñado especialmente para equipos de bajo consumo como tablets, smartphones, PDAs, etc.

Bluetooth ofrece unas velocidades que pueden llegar, con la versión BLUETOOTH 4.0, hasta los 24 Mbits/s.[1]

### 2.1.2. ZIGBEE

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

El ámbito donde se esta tecnología cobra más fuerza es en domótica. La razón de ello son diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo.
- Su topología de red en malla.
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

#### Topologías de red

ZigBee permite tres topologías de red [2]:

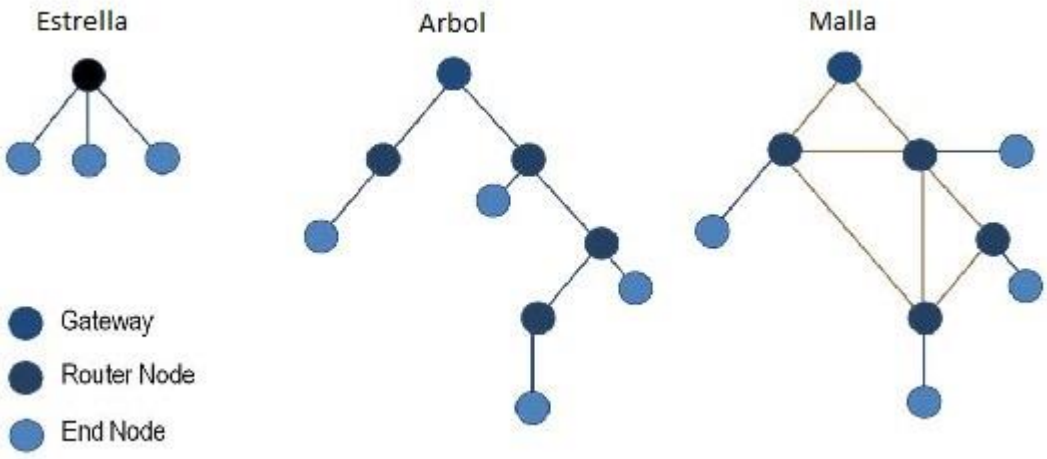



Figura 2.1: Tipos de topologías

- Topología en estrella: el coordinador se sitúa en el centro.
- Topología en árbol: el coordinador será la raíz del árbol.
- Topología de malla: al menos uno de los nodos tendrá más de dos conexiones.

La topología más interesante (y una de las causas por las que parece que puede triunfar ZigBee) es la topología de malla. Ésta permite que si, en un momento dado, un nodo del camino falla y se cae, pueda seguir la comunicación entre todos los demás nodos debido a que se rehacen todos los caminos. La gestión de los caminos es tarea del coordinador.

### 2.1.3. TCP/IP

TCP/IP es un conjunto de protocolos. La sigla TCP/IP significa "Protocolo de control de transmisión/Protocolo de Internet" y se pronuncia "T-C-P-I-P". Proviene de los nombres de dos protocolos importantes del conjunto de protocolos, es decir, del protocolo TCP y del protocolo IP.

En algunos aspectos, TCP/IP representa todas las reglas de comunicación para Internet y se basa en la noción de dirección IP, es decir, en la idea de brindar una dirección IP a cada equipo de la red para poder enrutar paquetes de datos. Debido a que el conjunto de protocolos TCP/IP originalmente se creó con fines militares, está diseñado para cumplir con una cierta cantidad de criterios, entre ellos [3]:

1. Dividir mensajes en paquetes.
2. Usar un sistema de direcciones.
3. Enrutar datos por la red.
4. Detectar errores en las transmisiones de datos.

La velocidad que nos aporta este protocolo es la que nuestra conexión a internet pueda soportar.

## 2.2. HARDWARE

### 2.2.1. ARDUINO

Arduino es una plataforma de hardware libre, basada en una placa con un micro controlador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.[4]

El hardware consiste en una placa con un micro controlador Atmel AVR y puertos de entrada/salida.4 Los micro controladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (boot loader) que corre en la placa.[5]

Figura 2.2: Placa Arduino UNO

Al ser Open Source Hardware, tanto su diseño como su distribución es libre, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

Especificaciones

El micro controlador Arduino Diecimila, Arduino UNO (figura 2.2) y Arduino Mega están basados en Atmega168, Atmega 328 y Atmega1280.

| | **Atmega168** | **Atmega328** | **Atmega1280** |
|---|---|---|---|
| **Voltaje operativo** | 5 V | 5 V | 5 V |
| **Voltaje de entrada recomendado** | 7 - 12 V | 7 - 12 V | 7 - 12 V |
| **Voltaje de entrada límite** | 6 - 20 V | 6 - 20 V | 6 - 20 V |
| **Pines de entrada y salida digital** | 14 (6 proporcionan PWM) | 14 (6 proporcionan PWM) | 54 (14 proporcionan PWM) |
| **Pines de entrada analógica** | 6 | 6 | 16 |
| **Intensidad de corriente** | 40 mA | 40 mA | 40 mA |
| **Memoria Flash** | 16KB (2KB reservados para el bootloader) | 32KB (2KB reservados para el bootloader) | 128KB (4KB reservados para el bootloader) |
| **SRAM** | 1 KB | 2 KB | 8 KB |
| **EEPROM** | 512 bytes | 1 KB | 4 KB |
| **Frecuencia de reloj** | 16 MHz | 16 MHz | 16 MHz |

Tabla 2.1: Especificaciones de Arduino Diecimila, UNO y MEGA

Una de las características más reseñables que aporta este hardware, es su capacidad para ser transformado con otros fines, como dotar a la placa Arduino de conexión Bluetooth, sensores de distancia, etc. Esto es posible gracias a placas conocidas como Shields. Los Shields se conectan encima de las placas Arduino, conectando los pines de ambos entre sí. En la figura 2.3. podemos ver un ejemplo.

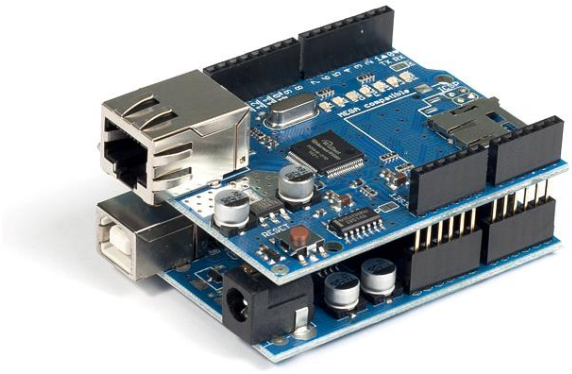


Figura 2.3: Conexión de un Shield a una placa Arduino

A continuación explicaremos los Shields a estudiar, para decidir más tarde que protocolo nos conviene más junto con las ventajas que aportan estas nuevas placas añadidas.

### 2.2.2. SHIELDS ARDUINO

**ETHERNET SHIELD**

La Arduino Ethernet Shield (figura 2.4) permite a una placa Arduino conectarse a internet. Está basada en el chip Ethernet Wiznet W5100. El Wiznet W5100 provee de una pila de red IP capaz de TCP y UDP. Soporta hasta cuatro conexiones de sockets simultáneas.

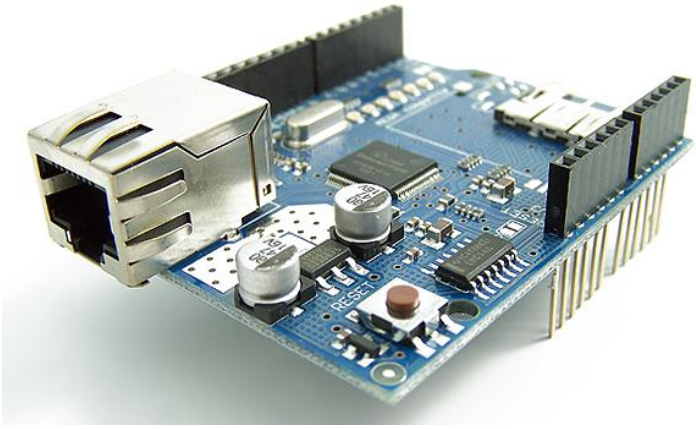


Figura 2.4: Ethernet Shield

Lo más destacado es que dispone de una ranura para tarjetas de memoria microSD.

La placa Arduino se comunica con el módulo W5100 y la micro-SD utilizando el bus SPI (mediante el conector ICSP). Esto se encuentra en los pines digitales 11, 12 y 13 en el modelo UNO y en los pines 50, 51 y 52 del modelo MEGA. En ambas placas, el pin 10 es utilizado para seleccionar el W5100 y el pin 4 para la micro-SD. Estos pines no pueden ser utilizados para otros fines mientras la Ethernet Shield esté conectada [6].

**WIRELESS SHIELD**

Este Shield tiene un zócalo para módulos como Xbee, Bluetooth, RFID o módulos WiFi. Lleva un regulador de 3.3V, adaptadores activos y un área para la creación de prototipos.


Figura 2.5: Wireless Shield

Como se puede ver en la figura 2.5, viene con una ranura microSD, que se puede utilizar con la biblioteca SD para guardar y cargar datos.

La alimentación es tomada del pin $V_{IN}$ del Arduino y regulada a 3,3V para alimentar el módulo wifi. Para comunicarse con el módulo se utiliza el protocolo SPI utilizando los pines digitales 10 a 13 (CS, MOSI, MISO y SCLK) [7].

Este Shield es indispensable para poder utilizar otros módulos que utilicen diferentes protocolos de transmisión. A continuación explicaremos los diferentes módulos:

- MODULO BLUETOOTH

Módulo Bluetooth para Arduino que puede ser conectado en el Shield WIFI y obtener una comunicación serie entre el ordenador y una placa Arduino a través del protocolo Bluetooth.

Figura 2.6: Modulo Bluetooth

**Especificaciones:**

- Chip Bluetooth: eUnistone 31308/2
- Versión: Bluetooth 2.0 + EDR (Configurable BT 1.2)
- TX Potencia: 2.5dBm
- RX Sensibilidad:-86dBm
- Antena: 2dBi
- Gama al aire libre: 60 m LOS (línea de visión)
- Rango interior: 40 m NLOS (sin línea de visión)

**Opciones especiales:**

- Mínimo y máximo control del umbral RSSI
- Salto de frecuencia adaptable
- Seguridad: Autenticación, "Pairing" y cifrado
- Almacena 5 dispositivos de confianza diferentes en su memoria interna
- Operación Master o Slave

- MODULO XBEE

El módulo de XBee proporciona una conectividad serie IEEE 802.11. Ofreciendo una relación consumo/coste muy baja. El XBee Wi-Fi crea nuevas oportunidades inalámbricas para la gestión de energía, procesos y automatización de fábricas, redes de sensores inalámbricos y más.


Figura 2.7: Modulo Xbee

- SPI flexible e interfaz serial UART

- Soporte para aplicaciones de baja potencia

- 802.11n proporciona hasta 65 Mbps de velocidad de datos

**Características [8]:**

- Interfaz de datos serial UART hasta 1 Mbps, SPI de hasta 3,5 Mbps
- Método de configuración API o comandos AT
- Banda de frecuencia ISM de 2,4 GHz
- Entradas ADC (4) de 12 bits
- 10 Pines Digital I / O
- Temperatura de funcionamiento -40 ° C a +85 ° C

**Redes y Seguridad [8]:**

- Cifrado WPA-PSK y WPA2-PSK
- Canales 14 canales
- LAN inalámbrico
- Estándar 802.11b/g/n

**Velocidades de datos [8]:**

- 802.11b: 1, 2, 5.5, 11 Mbps
- 802.11g: 6, 9, 12, 18, 24, 36, 48, 54 Mbps
- 802.11n: 6.5, 13, 19.5, 26, 39, 52, 58.5, 65 Mbps

- MODULO WIFI

Este módulo se basa en el protocolo de transmisión WIFI e incorpora el estándar 802.11 b / g, procesador de 32 bits, TCP/ IP, reloj en tiempo real, la unidad de administración de energía y sensores analógicos interfaz.


Figura 2.8: Modulo WiFi

**Características [9]:**

- Conectividad directa a Internet
- Conectividad punto a punto a cada nodo sin la necesidad de perfiles personalizados
- Basado en 802.15.4
- Ultra bajo consumo de energía: el modo de suspensión 4uA, 38 mA activa
- A bordo de la pila TCP / IP incluye DHCP, UDP, DNS, ARP, ICMP, el cliente HTTP, cliente FTP y TCP
- Firmware configurable
- Potencia de transmisión: 0 dBm a 12 dBm
- Interfaces de hardware: TTL UART
- Velocidad de datos de hasta 464Kbps
- Soporta Adhoc e infraestructura de redes
- 8 pines E / S digital
- Entradas de sensor analógicas
- Reloj en tiempo real para sellado de tiempo, auto-sleep, y los modos de auto-wakeup
- Acepta fuente de alimentación regulada 3.3VDC
- Configuración UART o más de interfaz inalámbrica (a través de Telnet) utilizando simples comandos ASCII
- Durante la actualización del firmware del aire (FTP)
- Autentificación WiFi : WEP, WPA-TKIP, WPA2-AES

## 2.3. PLATAFORMAS DE MOVIL

Las plataformas móviles son la base para cualquier dispositivo móvil. Proporcionan el SDK, herramientas y el sistema operativo que permiten el desarrollo de aplicaciones para esa plataforma. Por lo general, las propias plataformas tienen su propio modelo de distribución.

El desarrollador de la plataforma. Por ejemplo Apple iOS tiene AppStore, Google/Android tiene Play Store y Windows Phone tiene Windows Marketplace. Las aplicaciones móvil se desarrollan para una diversidad de plataformas móviles como Android, iOS (iPhone), Symbian (Nokia), Windows Mobile (Microsoft), Blackberry (SonyEricsson), etc. Recientemente, Symbian ha sido reemplazado por Windows Mobile desde que su popularidad disminuyó en el mercado móvil.

Desde la introducción de iOS (2007) y Android (2008) como plataformas móviles para smartphones, la popularidad ha ido creciendo en torno a estas dos sistemas operativos como base para el desarrollo de aplicaciones móviles. Apple lanzó iOS como plataforma para sus propios dispositivos (iPhone, iPod Touch). En contraste, Android fue liberado como código abierto por la Open Handset Alliance. Actualmente, Android cuenta con el apoyo de varios fabricantes como Samsung, Sony Ericsson, HTC, Toshiba y LG entre otros. [10]

En este proyecto me centraré solo en las dos plataformas más punteras que existen hoy en día, iOS y Android.

### 2.3.1. ANDROID

Android es una plataforma móvil lanzado por la Open Handset Alliance, que consisten en una pila de software compuesto por un sistema operativo, middleware y la clave de AP-8.

El desarrollo del software está ligado a la utilización de la máquina virtual Dalvik (Android Runtime) que permite el uso de Java como lenguaje de programación. La mayoría de las bibliotecas que son compatibles con JDK pueden ser desplegadas en un dispositivo Android. Sin embargo, algunos de ellos pueden presentar problemas de compatibilidad con el compilador, y por lo tanto no son capaces de ejecutarse. [11]

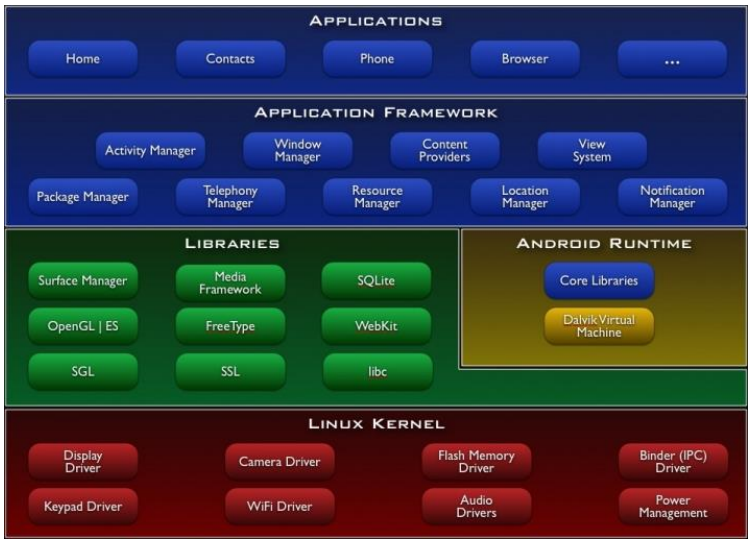



Figura 2.9: Arquitectura Android.

La arquitectura de Android se compone de varias capas que se basan en un núcleo Linux como se muestra en la figura 2.9. La capa de aplicación incluye un conjunto de aplicaciones por defecto en el sistema operativo, como el calendario, contactos, etc. El marco de aplicación constará de los servicios predeterminados para la gestión de los recursos de hardware (sensores, pantalla, etc.), software (alarmas, servicios de fondo) y la integración con recursos externos (sistemas de información de ubicación, notificación de servicios, etc.)

A pesar de que las aplicaciones de Android se distribuyen principalmente en el Android Market, estas se pueden distribuir libremente en Internet, una vez que se envasan en APK (Android Application Package) [13]. Sin embargo, si la aplicación no se comprar en el Android Market, se corre el riesgo de adquirir malware para Android.

### 2.3.2. iOS

iOS es una plataforma móvil creado por Apple y se implementa en sus dispositivos móvil (iPhone, iPod Touch). Debido a que es una tecnología propietaria, la mayor parte de su funcionalidad básica junto con el hardware, no es accesible para el desarrollador.

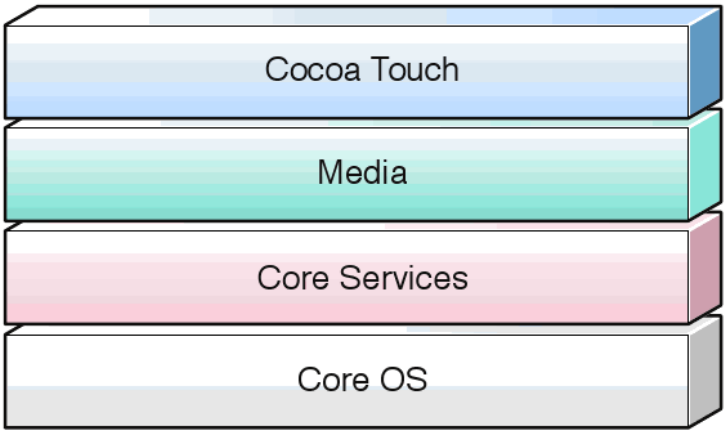



Figura 2.10: Arquitectura iOS.

iOS se compone de varias capas de software.

La arquitectura iOS se muestra en la figura 2.10. La complejidad de cada capa está relacionada con las líneas de código necesarias para alcanzar el objetivo en la aplicación móvil. En general, cuanto mayor sea el nivel, menos esfuerzo requerirá la creación de la aplicación móvil.

La capa Cocoa Touch [12] es la más alta de la plataforma iOS, escrito en lenguaje Objective-C. La capa proporciona servicios tales como servicio de notificaciones, entre otros. La capa Media proporciona capacidades para la reproducción audio y video, junto con las capacidades gráficas para animaciones. La capa de Core OS se encuentra en la parte inferior de la pila de iOS y, como tal, se encuentra directamente en la parte superior del hardware del dispositivo. La capa proporciona una variedad de servicios que incluyen bajo nivel de creación de redes, acceso a los accesorios externos y servicios del sistema, tales como la gestión de memoria, gestión de sistemas.

La distribución de las aplicaciones se dispone exclusivamente a través de la AppStore de iPhone. El desarrollador que desee publicar aplicaciones debe presentarlos para una inspección al centro de desarrollo de iOS. Una vez que se ha completado la revisión y la aplicación cumple los requisitos obligatorios establecidos por Apple, se publica la aplicación en la AppStore.

# 3. IMPLEMENTACIÓN DEL PROYECTO

## 3.1. DECISIÓN Y SELECCIÓN

Después de estudiar con detenimiento lo que nos puede aportar hoy en día la tecnología, es hora de escoger que protocolo, Hardware y SDK es el más óptimo para el desarrollo de nuestro proyecto.

Con el fin de poder dar un uso a este proyecto y poder definir mejor las necesidades del mismo, he inventado una supuesta necesidad que podrían tener los museos de todo el mundo. Trata sobre una aplicación móvil que el usuario utiliza por todo el museo, descargando información sobre los diferentes cuadros que hay y seleccionando, el propio usuario, el que quiera ver. Las ventajas de esta aplicación son varias, pero una de las más importantes es la comodidad que aporta al no tener que cargar con las Audio-guías que alquila el propio museo y así aprovechar que hoy en día la mayoría de la gente lleva consigo un Smartphone.

Una vez conocidas las necesidades del proyecto, se puede escoger con mayor acierto el protocolo de transmisión, hardware y SDK necesarios.

**TRANSMISIÓN**

El protocolo que es más acertado para este proyecto, según mis estudios, es el TCP/IP. El TCP/IP aprovecha toda la red global de Internet. Otro punto a favor, es la forma en que transmite y gestiona los paquetes de datos que se envían, ya que es capaz de detectar errores de envío y solucionarlos. También, la velocidad de transferencia, solo está limitada por nuestro móvil, que actualmente la velocidad de 3G pueden ser superiores a los 3 Mbits/s [13]. Y por último, uno de los datos más concluyentes es el alcance entre dispositivos, superior a los otros protocolos mencionados anteriormente. El TCP/IP, al ser el protocolo que gestiona todo Internet, brinda a cada equipo que esté conectado, una IP, por lo que el alcance entre dispositivos podría decirse que es infinito, en cambio, el BLUETOOTH tiene un alcance máximo entre dispositivos de 100 metros sin obstáculos de por medio[1].

## HARDWARE

Los factores que se han tenido en cuenta para la elección del hardware son los siguientes:

- Coste
- Fácil implementación
- Potencia

Una vez estudiado estos factores y sabiendo ya el protocolo que se utilizará, el Hardware más acertado es ARDUINO UNO. El coste de una de estas placas es de 20€, un coste bajo, teniendo en cuenta todo lo que nos puede ofrecer.

La implementación, como hemos dicho anteriormente es mediante programación con un lenguaje muy parecido al C. Además, está el IDE de Arduino con librerías y mucha documentación.

La potencia de esta placa es muy normal, pero es suficiente para la función que tiene que desempeñar, que es enviar audios, imágenes o textos de no más de 1 o 2 MB.

Sabiendo que lo que se utilizará como protocolo es TCP/IP, nos aísla un par de posibilidades. El Shield Ethernet o el Shield Wifly + Modulo WIFI. Si nos basamos en el primer factor, el coste, sin duda escogeremos el Shield de Ethernet, con un coste de 29€, mientras que si cogemos el pack WIFI, el coste sería la suma del Shield Wifly (19,90€) más el modulo WIFI (40€), un total de 59,90€ . También existe un Shield WIFI, con todo integrado, pero su precio asciende hasta los 69€. Sabiendo que el funcionamiento de los dos Shields es el mismo y en los dos casos dispondríamos de la ranura de microSD, la elección es el Shield de Ethernet. [14]

Para la realización del proyecto, dispondremos de un Router que será donde conectemos nuestra placa de Arduino + Ethernet.

## PLATAFORMA MÓVIL

A la hora de escoger la plataforma en que se desarrollaría la aplicación para Smartphones, el único factor clave ha sido la accesibilidad de cada una.

El SDK de iOS, queda descartado, ya que para adquirir el kit de desarrollo hace falta hacer un pago a Apple si después se quiere publicar. Por lo que la aplicación se realizará con Android, que es totalmente libre a la hora de publicar la App.

## 3.2. SOFTWARE NECESARIO

- ARDUINO: para la programación de la placa Arduino, es necesario descargar su propio IDE. Este IDE, está basado en Processing, que es un lenguaje de programación de código abierto basado en Java, de fácil utilización, y sirve para la producción de proyectos multimedia e interactivos de diseño digital.

- ANDROID: habitualmente se suele utilizar Eclipse, que es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma, en el que principalmente se utiliza Java. Por una mera cuestión de gustos, se escogió el IDE de programación NetBeans [15], ya que me resulta mucho más agradable para programar que Eclipse, además el editor es rápido, lo que hace que me sienta muy a gusto con él. El lenguaje de programación es Java. Se puede conseguir gratuitamente sin restricción de uso. Un añadido al IDE de NetBeans, es la instalación del SDK de Android, que se puede descargar gratuitamente de la página web [11]. Necesitaremos este plug-in para poder desarrollar la aplicación móvil y testearla en un móvil Android virtual, así como sus librerías y documentación.

## 3.3. EXPLICACIÓN DEL PROYECTO

Como se ha comentado en los objetivos, el proyecto si realiza basándose en una posible aplicación para Museos de todo el mundo, que facilite el uso de las nuevas tecnologías al cliente.

La idea es que cuando el cliente quiera información sobre un cuadro que está viendo en ese justo momento, utilice su móvil para descargar un video, sonido de la historia del cuadro o imagen con un texto explicativo.

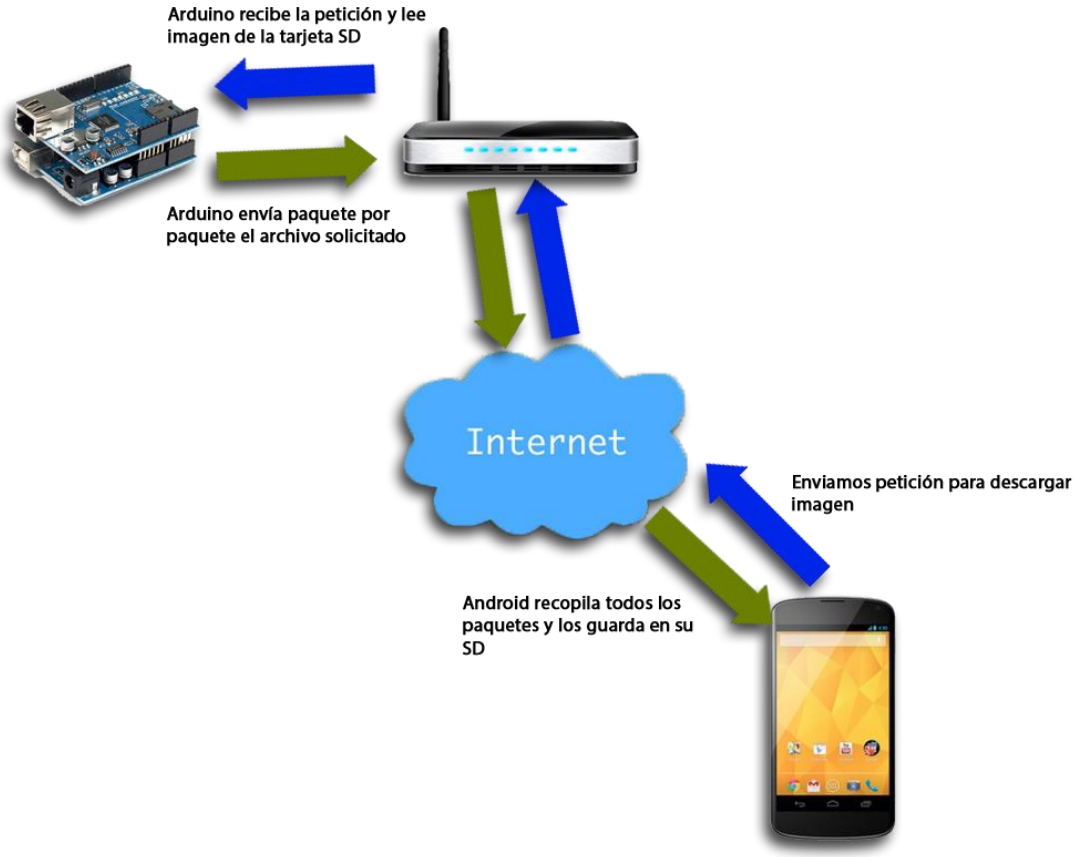



Figura 3.1: Diagrama explicativo de la idea del proyecto

El modo de funcionamiento es sencillo. Como se puede observar en la figura 3.1, el móvil del cliente, envía una petición a un servidor (que será nuestro Arduino), y este cuando la recibe, lee del disco duro (micro-SD) y envía al cliente el archivo demandado.

Para realizar este proyecto, se ha centrado en cuatro partes:

1. Diseño de protocolo de transferencia
2. Implementación del servidor en Arduino
3. Diseño de la aplicación móvil Android
4. Implementación del cliente en Android

### 3.3.1. DISEÑO DEL PROTOCOLO DE TRANSFERENCIA

El reto más importante que hay en este proyecto, es lograr la transferencia de archivos entre el servidor y el móvil. Para ello, hace falta que se entiendan perfectamente para que no haya perdidas de información.

En este apartado explicaré brevemente como se llevará a cabo este entendimiento del que hablamos y en los siguientes apartados se irá profundizando en la explicación.

Una forma de conseguir que dos programas se transmitan datos, basada en el protocolo TCP/IP, es la programación de sockets [16]. Como podemos ver en la figura 3.2, un socket no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador. Ambos programas deben conectarse entre ellos con un socket y hasta que no esté establecida correctamente la conexión, ninguno de los dos puede transmitir datos. Esta es la parte TCP del protocolo TCP/IP, y garantiza que todos los datos van a llegar de un programa al otro correctamente. Se utiliza cuando la información a transmitir es importante, no se puede perder ningún dato y no importa que los programas se queden "bloqueados" esperando o transmitiendo datos. Si uno de los programas está haciendo otra función y no atiende la comunicación, el otro quedará bloqueado hasta que el primero lea o escriba los datos.

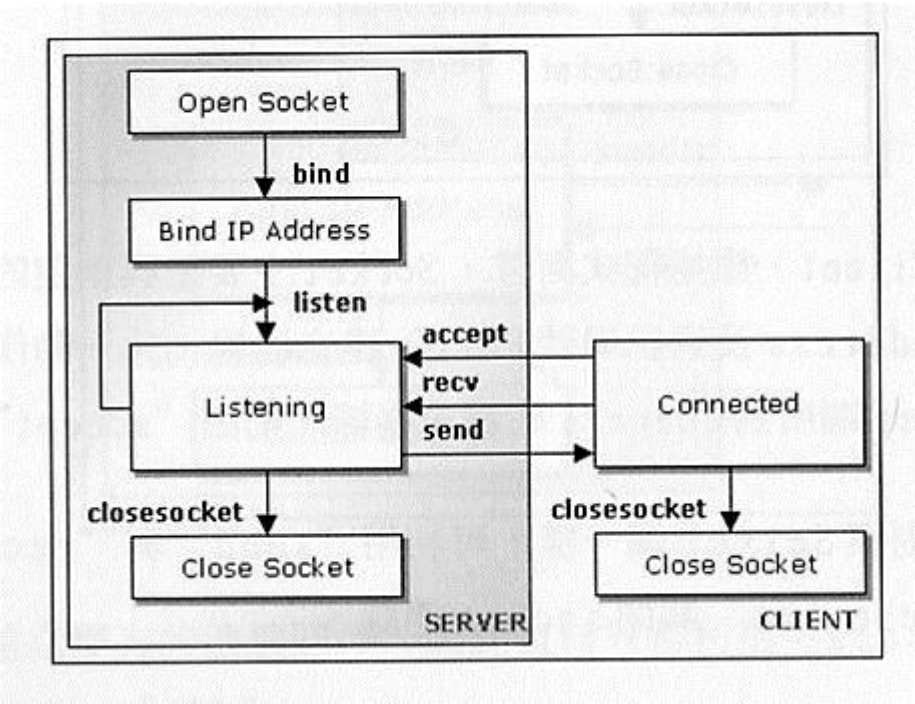



Figura 3.2: Funcionamiento de un Socket

Los pasos que seguirá el protocolo de transferencia serán los siguientes:

Primero, el móvil y el servidor Arduino, han de establecer una conexión. Una vez realizada con éxito, el usuario seleccionará un cuadro y la aplicación enviará una petición al servidor, donde este la recibirá. Como se trata de una transferencia mediante el protocolo TCP/IP, hay que aprovechar que trabaja con envío de datos en paquetes. Por lo que el servidor tiene que ser capaz de leer byte a byte los archivos de la tarjeta SD y almacenarlos en buffers para su envío. Lo mismo ocurre con la aplicación móvil, debe ser capaz de recibir esos paquetes e ir leyéndolos byte a byte y guardándolos en un buffer, que más tarde, será el archivo deseado. Como podemos ver en la Figura 3.1, el servidor, bajo una petición, fragmenta en paquetes el archivo, lo envía, y el móvil vuelve a juntar esos paquetes para volver a construir el archivo.

Una vez finalizada la transferencia, el link que une al cliente y al servidor, se rompe para dejar paso a una futura conexión.

### 3.3.2. IMPLEMENTACIÓN DEL SERVIDOR EN ARDUINO

Antes de comenzar con la explicación, voy a hacer una breve introducción al tipo de programación que utilizan las placas Arduino.

#### 3.3.2.1. PRIMEROS PASOS EN ARDUINO

Lo primero que tenemos que hacer para comenzar a trabajar con el entorno de desarrollo de Arduino es configurar las comunicaciones entre la placa Arduino y el PC. Para ello deberemos abrir en el menú "Tools" la opción "Serial Port". En esta opción deberemos seleccionar el puerto serie al que está conectada nuestra placa. En Windows, si desconocemos el puerto al que está conectado nuestra placa podemos descubrirlo a través del Administrador de dispositivos (Puertos COM & LPT/ USB Serial Port).

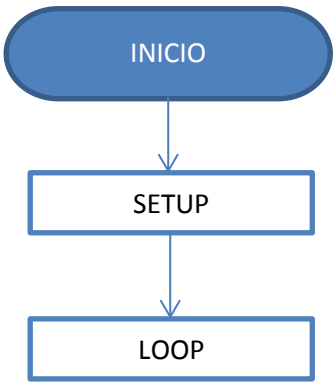



Figura 3.3: Diagrama de flujo del programa principal

La estructura básica de programación de Arduino es bastante simple y divide la ejecución en dos partes: setup y loop. Setup() constituye la preparación del programa y loop() es la ejecución. En la función Setup() se incluye la declaración de variables y se trata de la primera función que se ejecuta en el programa. Esta función se ejecuta una única vez y es empleada para configurar el pinMode (p. ej. si un determinado pin digital es de entrada o salida) e inicializar la comunicación serie. La función loop() incluye el código a ser ejecutado continuamente (leyendo las entradas de la placa, salidas, etc.). [5]

```
void setup() {
   inMode(pin, OUTPUT);  // Establece 'pin' como
salida
}
void loop() {
   digitalWrite(pin, HIGH);    // Activa 'pin'
   delay(1000);          // Pausa un segundo
   digitalWrite(pin, LOW);     // Desactiva 'pin'
   delay(1000);
}
```

Como se observa en este bloque de código cada instrucción acaba con ; y los comentarios se indican con //. Al igual que en C se pueden introducir bloques de comentarios con /* ... */.

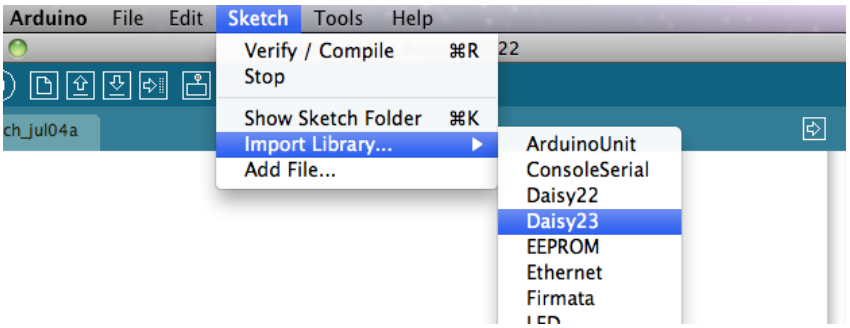



Figura 3.4: IDE Arduino

Una parte importante del mundo Arduino son las librerías que existen. Las Librerías proveen funcionalidad extra a nuestro sketch, por ejemplo: al trabajar con hardware o al manipular datos. Para usar una librería dentro de un sketch, se selecciona desde *Sketch > Import Library* (Importar Librería), como se puede observar en la figura 3.4.

En nuestro caso, necesitaremos dos librerías, la de Ethernet y la de SD. La librería de Ethernet permite que la placa Arduino pueda conectarse a Internet utilizando el Ethernet Shield y la librería de SD permite el control total (lectura, escritura, borrado, etc) de la ranura SD que incorpora el Ethernet Shield.

#### 3.3.2.2. DECLARACIÓN DEL BUFFER

Como se ha dicho antes, se necesita un buffer para poder optimizar al máximo el envío de datos, con el fin de conseguir la máxima velocidad de transferencia. Con Arduino UNO tenemos una memoria de 2KB de SRAM [17]. Hay que tener en cuenta que cada carácter que se muestre por pantalla estará utilizando un byte, por lo que no es difícil llegar a 1KB. Si a esto le sumamos que tenemos que reservar memoria para el BUFFER, debemos escoger muy bien el tamaño del mismo para que no se quede sin memoria.

El tamaño del Buffer será de 225 Bytes, ya que es una cifra considerable y no ponemos en riesgo el programa.

Para la declaración del Buffer, debe hacerse fuera de las dos partes principales del código de Arduino nombradas anteriormente.

```
byte buffer[225];
```

La palabra byte que podemos ver en el código anterior es una variable de tipo Byte. La variable Byte almacena un número sin signo de 8 bits, desde 0 a 255 valores.

#### 3.3.2.3. DECLARACIÓN DEL SERVIDOR

Para que Arduino actúe como servidor hace falta declararlo y configurarlo con su dirección MAC, IP y el puerto por el que se comunicará.

```
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

IPAddress serverIP(192,168,1,128);

int serverPort=8888;

EthernetServer server(serverPort);
```

Como se puede observar, el código es bastante sencillo. Lo único que hay que explicar es la última línea, que su función es la de inicializar la librería Ethernet.

### 3.3.2.4. SETUP

Como se puede observar en la Figura 3.5, en este apartado se configura la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Su principal función es para comunicarse con el ordenador, y se utiliza una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200.

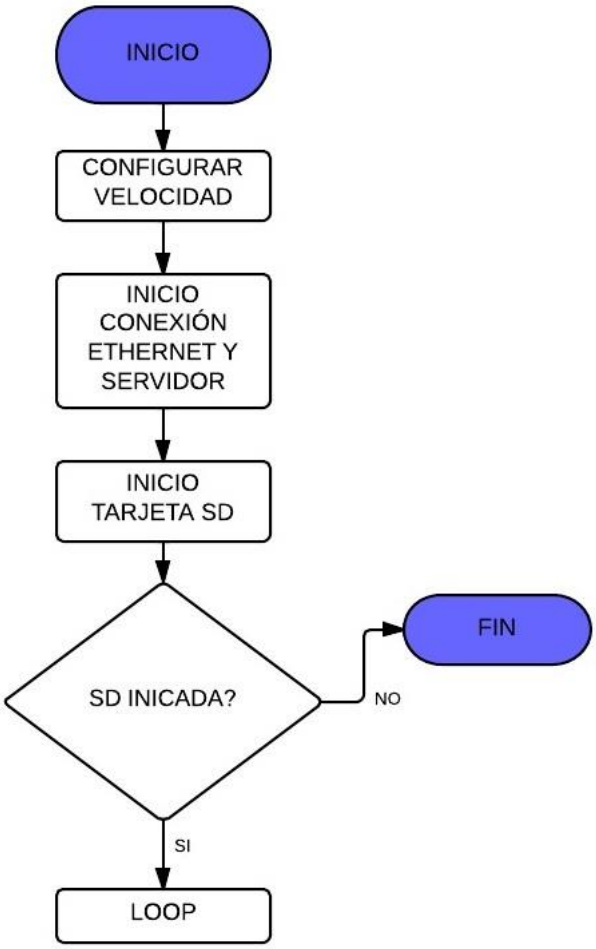



Figura 3.5: Diagrama de flujo de la función Setup

Justo después se inicia, gracias a las librerías de Ethernet, la conexión Ethernet y el Servidor con la información que se han declarado anteriormente (IP, dirección MAC y puerto).

La SD es lo que se inicia a continuación. Para acceder a ella, se utiliza el PIN 4, por ello se comprueba que está en correcto funcionamiento y lo muestra por pantalla. Si no funcionara, el programa terminaría y habría que buscar el error de la tarjeta SD.

Si todo ha salido correctamente, el código pasará a ejecutar el programa principal, el Loop.

### 3.3.2.5. LOOP

En esta fase del código es donde se ejecuta la función principal del programa del Servidor. Nos basaremos en el diagrama de flujo de la figura 3.6.

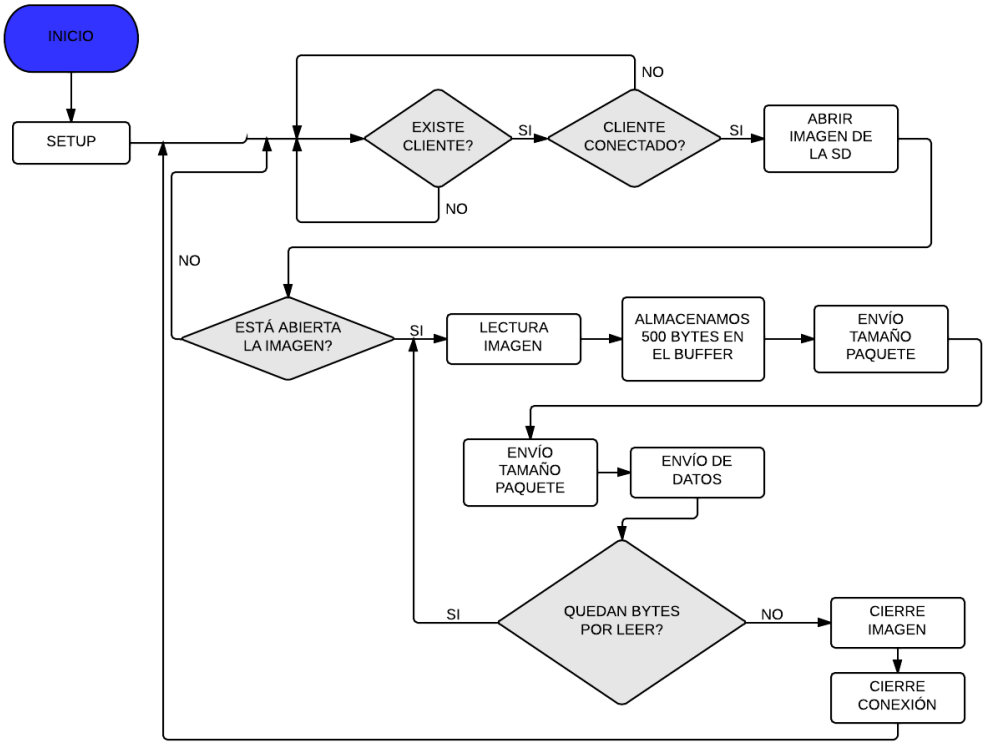



Figura 3.6: Diagrama de flujo de la función Loop

Lo primero que se comprueba es si hay algún cliente entrante queriendo conectarse. Ya que se trata de una función LOOP, si no existiera ningún cliente entrante, lo seguiría esperando mientras la placa Arduino siga encendida, ya que es un bucle infinito.

Una vez el cliente ha enviado una petición y se ha logrado hacer la conexión, el programa trabajará bajo una condición "*while*" para que siga trabajando mientras el cliente está conectado.

El cliente enviará un dato para identificar la imagen a descargar por el servidor. A continuación el programa abre la imagen alojada en la tarjeta SD y comprueba si se ha abierto correctamente. Si no lo estuviera, el programa cerraría la conexión y volvería al bucle inicial esperando a otro cliente y mostrando por pantalla el error.

Si la imagen se ha abierto correctamente entrará en un bucle para ir leyendo 225 bytes y almacenarlos en un buffer para su envío, así hasta que no quede ningún byte por enviar. Un dato importante a tener en cuenta es que todos los paquetes serán de 225

bytes excepto el último, ya que se puede dar el caso de que sean menos y el cliente debe saber de qué tamaño será el paquete para que no haya errores. Por ello, antes de enviar los datos, se envía también, el tamaño del buffer.

Finalmente, cuando la imagen haya sido enviada, enviará al cliente otra vez el tamaño del buffer, pero esta vez siendo cero para que haga de discriminador. Cerraremos la imagen y finalizaremos la conexión con el cliente, así volveremos a estar a la escucha de futuras peticiones de clientes.

### 3.3.3. DISEÑO DE LA APLICACIÓN ANDROID

Sabemos, que el sistema operativo es el encargado de pausar, parar o destruir nuestra aplicación según las necesidades de recursos del dispositivo, aun así, nosotros como desarrolladores debemos aprender a controlar todos estos eventos para hacer nuestras aplicaciones robustas y mejorar el rendimiento de los teléfonos.

Entre el código que nos encontramos, en Android, se usa el termino de Actividad ("*Activity*" en inglés) para denominar a un tipo de clases java que heredan de Activity. Una actividad, como su propio nombre indica, es algo que el usuario puede hacer y en nuestro teléfono móvil son las pantallas que visualizamos en las aplicaciones. El símil que se me ocurre ahora mismo, con el ejemplo anterior, es el de actividad física, que es un conjunto de movimientos del cuerpo que obtienen como resultado un gasto de energía. En Android una actividad es lo mismo, es un conjunto de acciones (tocar la pantalla para apretar un botón, para escribir con el teclado, etc.) que son una iteración directa con el usuario y que afectan a una parte de la aplicación. [11]

Una actividad se caracteriza por tener un ciclo de vida. Un ciclo de vida, al igual que el ciclo vida de un animal: nace, crece, come, se reproduce y muere; también, es semejante en una actividad. De la actividad de correr tendría el siguiente ciclo de vida: estiramos antes de empezar a correr, comenzamos a correr, corremos a tope, en cierto momento decidimos hacer una pausa para hacer flexiones, luego podemos decidir continuar con la carrera, y en otro momento dejamos correr para otro día. El siguiente diagrama de la figura 3.7 (modificación del oficial para realizar el ejemplo, diagrama oficial que mostramos más adelante) muestra cómo se desarrolla el ciclo de vida de la actividad de correr y así poder entender mejor la vida de una Activity (Diagrama disponible en el Anexo D en su tamaño real).

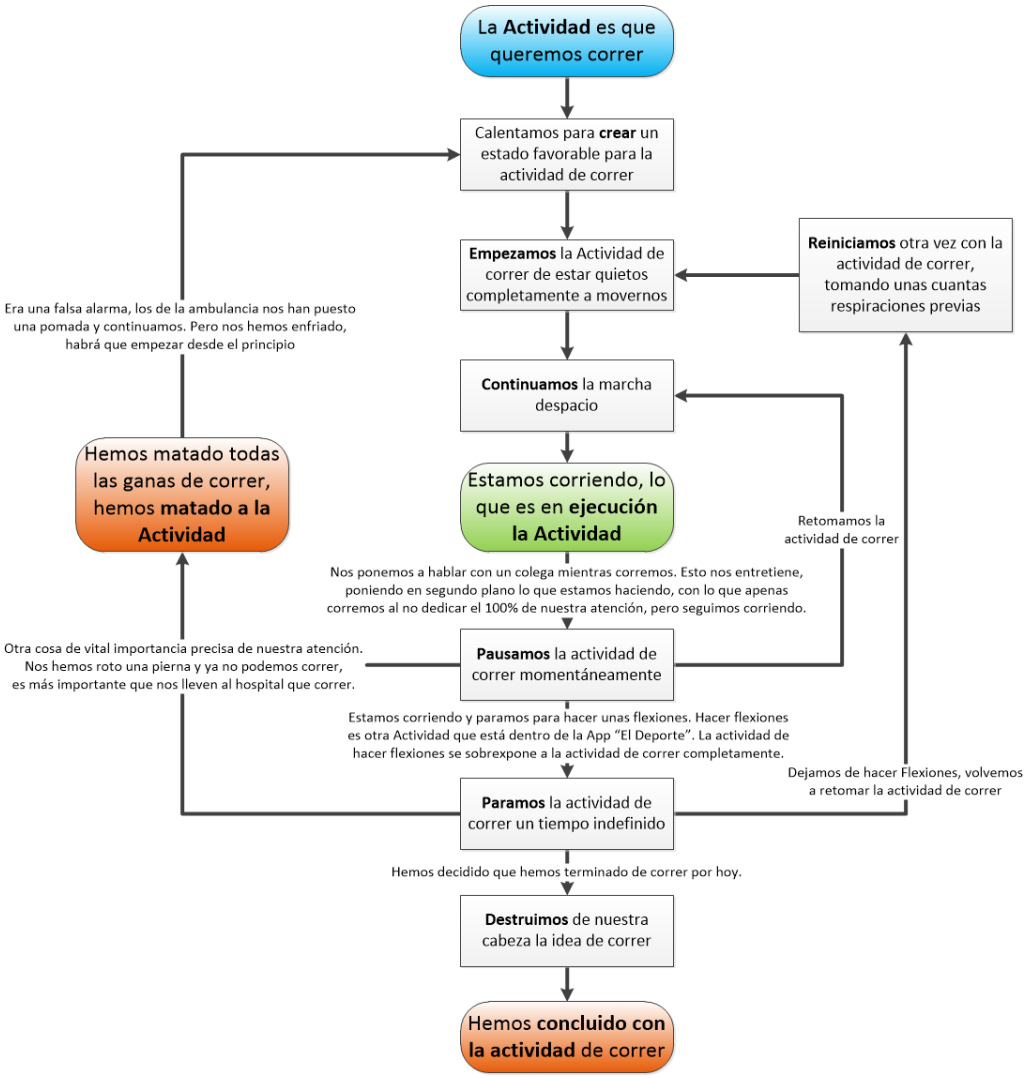



Figura 3.7: Diagrama del ciclo de vida de una "Activity" (modificado)

Si pensamos en todo lo que podemos hacer con nuestro móvil teniendo la aplicación en marcha, y lo comparamos con la actividad de correr podemos ver que son idénticas:

- Arrancar la actividad: Pasará por Crear, Empezar y Continuar, para llegar a la ejecución normal.
- Usar de manera normal la actividad: estamos en la actividad en ejecución.
- Una ventana emergente se ha abierto: Pasará por Pausar.
- Cambiar a otra actividad o bloquear el móvil: Pasará por Pausar y Parar.
- Apagar el móvil: Pasará por Pausar, Parar y Destruir.

Entendido esto veamos el ciclo oficial de una actividad real de un Smartphone:


Figura 3.8: Diagrama oficial del ciclo de vida de una "Activity" [13]

El diseño de la aplicación creada para este proyecto es sencillo y fácil de utilizar, ya que la dificultad principal del proyecto reside en el envío y entendimiento entre la aplicación y la placa Arduino, que se explica en el próximo apartado.

La aplicación constará de tres "activities":

1. Pantalla principal: se trata del menú principal de la aplicación, donde está el título y los botones para trasladarse a las siguientes pantallas.

2. Pantalla de Cuadros: en esta pantalla, se visualizan los títulos de los cuadros que el cliente puede visualizar, descargando del servidor Arduino. Además, contiene el botón de descarga.
3. Pantalla "About": es donde se explica el funcionamiento de la aplicación y el nombre del autor.

Como se ha dicho anteriormente, el código de las activities creadas para esta aplicación, contienen el método "*onCreate*". Al método "*onCreate*" se le pasa por parámetro una variable que se llama "*savedInstanceState*" de tipo Bundle. Un Bundle son datos que le pasamos a esta actividad para inicializar su estado. En este ejemplo no lo usamos, pero si pensamos por ejemplo en un juego, en el que se pide en una actividad el sexo del personaje, y pasamos a la siguiente actividad, en el que se ve nuestro personaje con el sexo elegido; el sexo como variable hombre o mujer ha tenido que ser pasado de algún modo de una actividad a otra, se hace con Bundle.

El *onCreate* requiere iniciar al padre antes de todo, de ahí la línea: `super.onCreate(savedInstanceState);`

En la línea siguiente establecemos que Layout queremos asociar a esta actividad, como es la actividad "Cuadros", es justo asociarle el layout "Cuadros", se hace como se muestra en: `setContentView(R.layout.cuadros);`

Existe otro método normal y corriente llamado *OnClickListener* que es el que se ejecuta cuando hacemos clic en el botón con el texto "CUADROS".

Este método lo único que hace es abrir la otra actividad "Cuadros". Se hace con algo llamado "Intent". Un Intent es declarar la intención de lo que se quiere hacer, en este caso la intención es ejecutar la actividad Cuadros –aunque podría decir que además de abrir otra actividad lo que queremos es también pasar ciertos datos como en el ejemplo anterior del juego con el sexo. Dicho de otro modo, un Intent es un paquete con todo lo que queramos configurar la nueva "activity" a ejecutar. Solo falta decir que el "*this*" es el contexto de la actividad actual. Esto lo explica el código:

```
Intent intent = new Intent(this, Cuadros.class);
```

Después de declarar la intención hay que lanzar la actividad con: `startActivity(intent);`

Para llegar a la siguiente actividad y Layout asociado necesitamos su código. Creamos otra nueva actividad que llamamos “Cuadros”.

### 3.3.4. IMPLEMENTACIÓN DEL CLIENTE EN ANDROID

La parte del código que se explica a continuación es la que se encarga de realizar la transferencia y recibirla como cliente. La explicación se basará en el diagrama de flujo de la figura 3.9.

Este código se ejecutara cuando el usuario presione el botón del cuadro que quiere visualizar. Justo en ese momento, el programa creará un socket entre el servidor y el cliente para poder transferir los datos, y lo mantendrá abierto hasta el final del envío o cuando se produzca algún error de comunicación.

Una vez establecida la conexión, el programa crea un ArrayList donde guardaremos el archivo entrante paquete a paquete. Un ArrayList es un Array normal pero dinámico [18]. Las razones por la que he utilizado un ArrayList en lugar de un típico Array, son las siguientes:

- Un ArrayList colecciona objetos, un array colecciona datos de un mismo tipo.

- Su tamaño. Es decir, cuando declaras un array debes indicar su tamaño al instanciarlo. Y si quieres incluir más valores para los que en origen se creó, deberás redimensionarlo, pero al hacer esto los valores introducidos con anterioridad se pierden. Esto no sucede con los ArrayList porque uno de sus constructores, los construye con tamaño indeterminado, por lo que no hay límite.

- El catálogo de métodos que tiene un objeto de ArrayList, de los que obviamente carece un array, ya que si se necesitan deberán programarse. Por ejemplo, a los arrays le faltan la funcionalidad de ordenar descendente y ascendentemente, de eliminar un ítem según un valor coleccionado, de limpiar su contenido a través de una función, sin tener que realizar un bucle. Para saber si un array cuenta con un dato tendrás que recorrerlo entero. Sin embargo un ArrayList te devuelve el objeto que en la colección tiene ese valor.

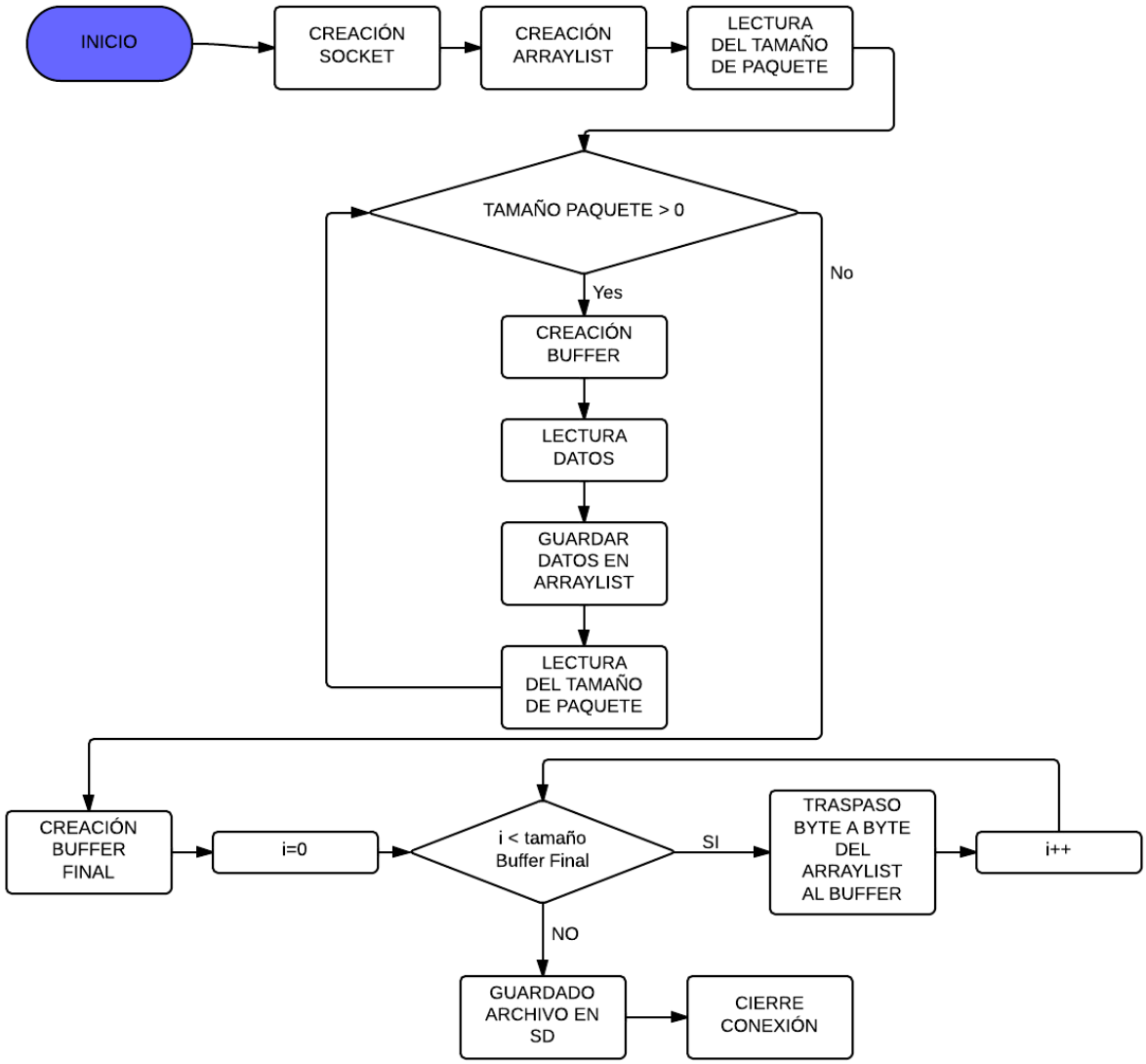



Figura 3.9: Diagrama de flujo del cliente

Este tipo de Array facilita mucho la gestión de los datos que se envían y su almacenamiento.

Como se ha explicado en el apartado de implementación del servidor, antes de enviar los datos en paquetes, el servidor debe enviar el tamaño de dicho paquete, para que el cliente cree un buffer de bytes del mismo tamaño. Esto se realiza, como se ha dicho antes, porque el último paquete no tiene por qué ser de 225 Bytes.

Un problema importante que encontré a la hora del entendimiento entre Arduino y Android fue la cantidad de bytes que utilizan para sus variables. Por ejemplo, Arduino utiliza 2 bytes para las variables de enteros int y Android utiliza 4 bytes. Para solucionarlo se ha tenido que utilizar la función "Union" [19]. Las uniones son un tipo especial de estructuras que permiten almacenar elementos de diferentes tipos en las mismas posiciones de memoria aunque no simultáneamente.

Un ejemplo sería el siguiente:

Supongamos que en nuestro ordenador, int ocupa cuatro bytes, char un byte y double ocho bytes. La forma en que se almacena la información en la unión del ejemplo sería la siguiente:

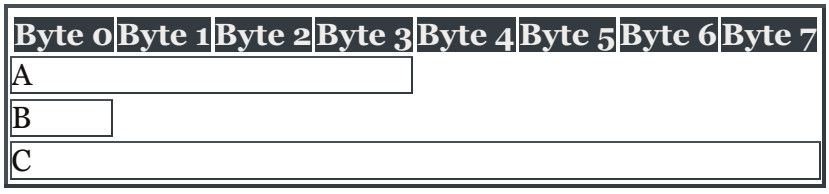



Figura 3.10: Almacenamiento de la función Union.

Por el contrario, los mismos objetos almacenados en una estructura normal tendrían la siguiente disposición:


Figura 3.11: Almacenamiento sin Union.

Utilizando uniones se ha relacionado el int con un Array de bytes. Con lo que una vez en Java, el código debe de ser capaz de decodificar esos 2 bytes en un int.

Una vez solucionado este problema el código entra en una condición, que mientras el tamaño del buffer sea mayor que 0, se creará un vector de ese tamaño, leerá los datos y se pasarán los datos del vector al ArrayList.

Cuando se reciban todos los datos, el código pasará los datos del ArrayList a un buffer final con el que, gracias a una función importada de una librería de Java, lo guardaremos en la SD del teléfono móvil con el formato deseado (jpg, txt, wav, etc…).

Y finalmente, cuando Arduino envíe el 0 discriminador, el código de Android lo entenderá como que ha terminado de transmitir y cerrará la conexión.

# 4. VALIDACIÓN DE LA APLICACIÓN

En este apartado se demostrará el correcto funcionamiento de la aplicación y diferentes pruebas de rendimiento realizadas.

## 4.1. INTERFAZ GRÁFICA

Como se comenta en el apartado de diseño de la aplicación Android, el programa consta de tres "activities", la primera es el menú principal, después se bifurca en dos más, una pantalla donde estarán los archivos para descargar y otra de información sobre la aplicación. A continuación se mostrarán las diferentes pantallas, así como el escritorio del teléfono móvil.

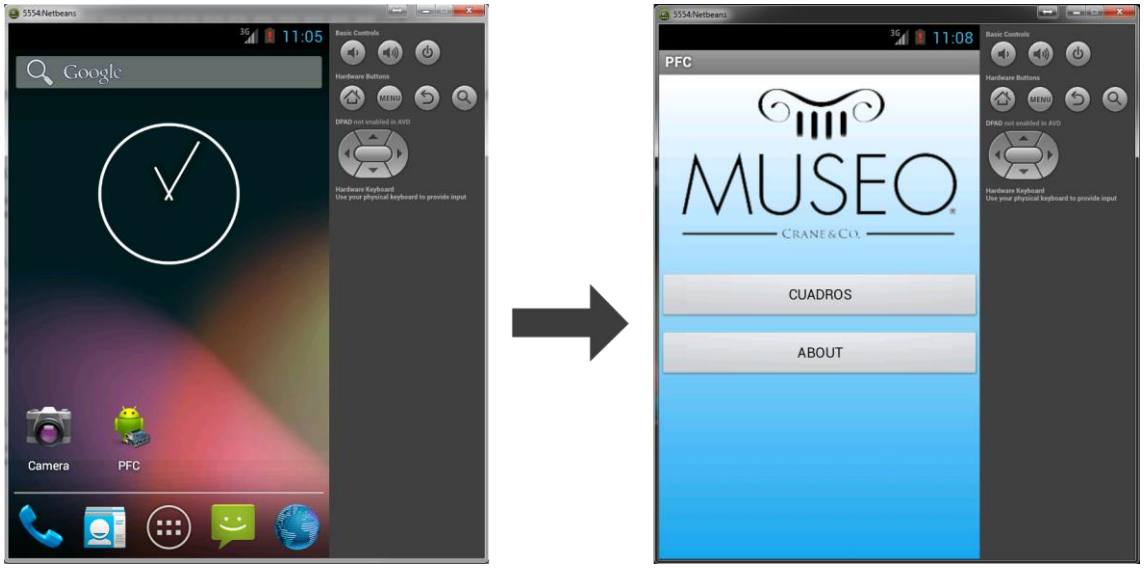



Figura 4.1: Escritorio y menú principal de la aplicación

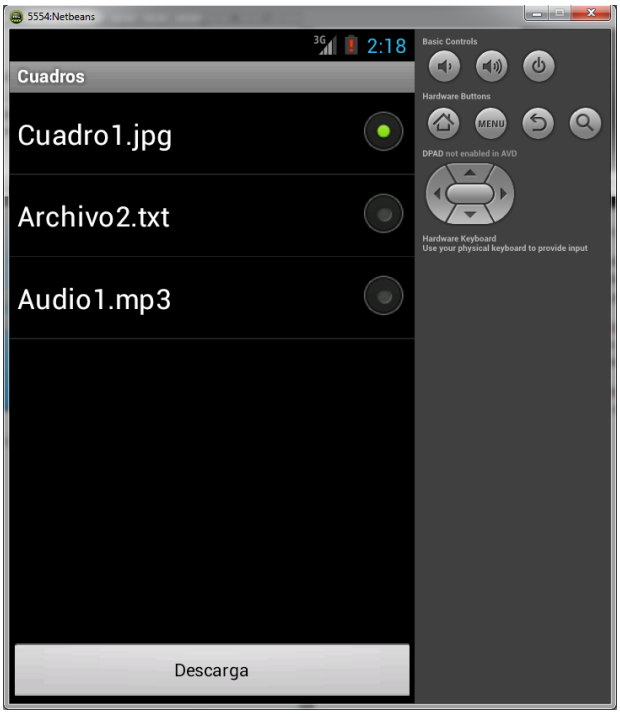



Figura 4.2: Pantalla Cuadros

En esta pantalla (Figura 4.2) se pueden encontrar los archivos para que el usuario los descargue. Se han presentado diferentes archivos con diferentes tamaños para, posteriormente, hacer las respectivas pruebas.

El funcionamiento de esta Activity es sencillo, el usuario deberá seleccionar un archivo de la lista y pulsar el botón "Descarga". Cuando finalice la transferencia, el programa mostrara la imagen al usuario y la guardara un la tarjeta SD.

El programa realiza *feedbacks* (comentarios del proceso que está siguiendo en cada momento) al usuario, para que en ningún momento piense que el programa se ha colgado.


Figura 4.3: Pantalla "About"

Esta pantalla del programa (Figura 4.3), sirve para mostrar el funcionamiento del proyecto, así como su título y el autor.

## 4.2. TEST

Los test que se han realizado para probar el funcionamiento de la aplicación, han sido sobre el tiempo que tarda en transferirse un archivo. Hay que añadir, que los envíos son en paquetes de Bytes, por lo que no importa si es un archivo de imagen, texto, video o audio, sino el tamaño que tienen.

En el código de la aplicación de Android, se ha implementado un contador, para que, una vez finalizada la transferencia, muestre el tiempo transcurrido desde que el usuario presiona el botón hasta que se recibe el archivo.

A continuación se mostrarán las imágenes con los diferentes casos y sus correspondientes tiempos de transferencia.

### 4.2.1. PRIMER TEST. IMAGEN

Como se puede observar en la imagen de la figura 4.4, el tiempo que ha tardado ha sido de 25.987 segundos. Este tiempo ha sido para una imagen con un peso de 432 kB.

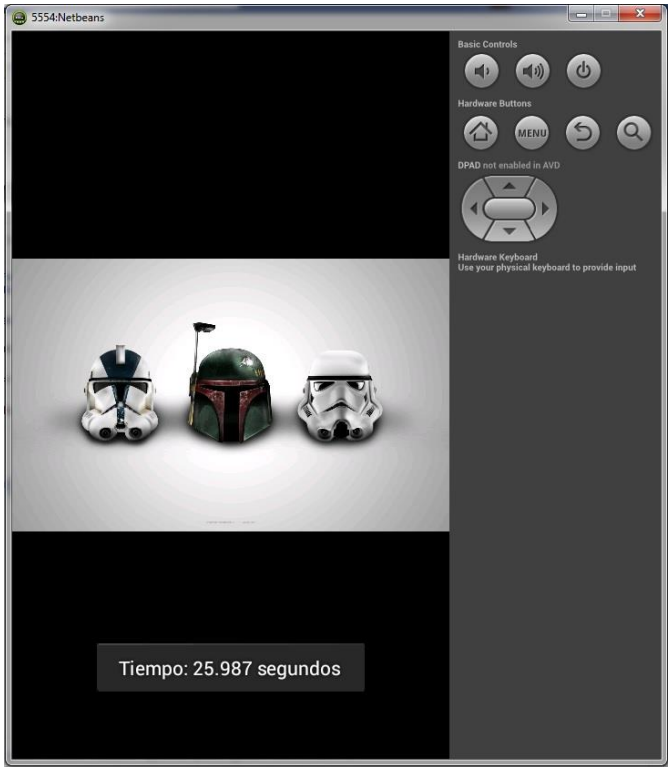



Figura 4.4: Tiempo transferencia de una imagen

### 4.2.2. SEGUNDO TEST. TEXTO

Para la segunda prueba, se trata de enviar un archivo de texto. El peso del archivo es de 1.73 kB, y ha tardado 0.254 segundos.


Figura 4.5: Tiempo transferencia de un texto

### 4.2.3. TERCER TEST. AUDIO

El archivo de audio tiene un tamaño de 1.57 MB y ha tardado 90.437 segundos (Figura 4.6).


Figura 4.6: Tiempo transferencia de un audio

## 4.3. RESULTADOS

Los resultados de tiempo y velocidad obtenidos tras una serie de pruebas de envío/recepción de archivos con diferentes tamaños son los siguientes:

| Archivo | Tamaño del archivo | Tiempo de transferencia | Velocidad |
|---|---|---|---|
| **Imagen** | 432 kB | 25.987 segundos | 16.62 KBps |
| **Texto** | 1.73 kB | 0.254 segundos | 6.81 KBps |
| **Audio** | 1.57 MB | 90.437 segundos | 17.36 KBps |

Tabla 4.1: Tiempos y velocidad de transferencia

# 5. CONCLUSIONES

## 5.1. CONCLUSIONES TÉCNICAS

Una vez analizados los resultados finales obtenidos de las transferencias, se puede considerar que se han cumplido los objetivos más importantes del proyecto, centrados principalmente en tres aspectos: implementar un servidor en una placa Arduino UNO + Ethernet Shield, implementar un cliente con la creación de una aplicación Android para teléfono móvil y ser capaz de gestionar la transferencia de un archivo lo más óptimamente posible entre el servidor y el Smartphone mediante el protocolo TCP/IP.

Como se ha podido observar en la tabla 4.1, los tiempos de transferencia no son del todo idóneos, ya que si los comparamos con el tiempo que tardan hoy en día en transferirse archivos de un ordenador a otro, son demasiado altos. Igualmente, se puede mirar desde otro punto de vista según el uso que se le vaya a dar a este proyecto. En este caso, como se ha dicho desde un principio, se basaba en una supuesta aplicación para museos, en la que el cliente pudiese descargar información adicional de los cuadros si la necesitaba. Por lo que, si el cliente final quiere descargase un audio o una imagen con texto, el archivo no tiene por qué ser de un tamaño superior al que vemos en la Tabla 4.1. Por ejemplo, para visualizar imágenes en un teléfono móvil no hace falta que el archivo sea muy grande, con una fotografía de 200 kB, la resolución es más que aceptable, y si contamos con los métodos de compresión y codificación que existen, el sonido tampoco es problema, ya que una voz que explique información sobre el cuadro, no requiere ser estéreo, ni tener 16 bits y mucho menos utilizar 44000Hz de frecuencia de muestreo, con ser mono, tener 8 bits de resolución y 16 KHz de frecuencia de muestreo, se puede entender perfectamente al locutor.

El principal motivo de estos tiempos tan elevados, es el cuello de botella que genera la placa Arduino UNO. Debido a su corta memoria SRAM, no es posible hacer un Buffer de envío mayor de 225 Bytes, así que, aunque contemos con la mejor velocidad de internet, Arduino nos limitará la velocidad de transferencia.

## 5.2. APLICACIONES DEL PROYECTO

La aplicación natural por la cual se ha desarrollado este proyecto es la de tener un sistema de transferencia de datos entre un servidor y el teléfono móvil por el coste económico más pequeño posible, por ello se han utilizado componentes "Open Source

Hardware". Así se demuestra que no hace falta un equipo con especificaciones técnicas altas y un precio desorbitado.

El presente proyecto también puede servir como un sistema domótico de vigilancia, ya que incorporándole una cámara digital, se podría tener una vivienda en todo momento monitorizada desde un teléfono móvil.

Otra posible aplicación, en el caso hipotético de que se mejoraran los componentes de "Open Source hardware" a un precio modesto, podría ser la transferencia de datos (imagen, video o sonido) vía "streaming". Para esta posible aplicación, Arduino debería estar dotado de un procesador más potente y un mayor tamaño de SRAM.

## 5.3. MEJORAS

En el caso de que este proyecto pueda ser objeto de mejoras, estas se tendrían que centrar en profundizar e implementar con más detalle una mejora de la transmisión de datos. Para ello, sería interesante mejorar los siguientes puntos:

- En el presente proyecto, esta implementado de forma que solo pueda conectarse un cliente cada vez que quiere descargar un archivo, es decir, mono-hilo. Por ello propongo que se implemente la transmisión por un socket multi-hilo.

- El diseño de la aplicación para móvil puede ser mejorada en apariencia y optimización de código, dando también más funcionalidades para amenizar el contacto con el usuario final.

- Durante la implementación de este proyecto, surgió una nueva placa Arduino por un precio muy similar al Arduino UNO, llamada Arduino DUE. Esta placa cuenta con 96kB de SRAM, muy por encima de los 2kB del Arduino UNO, un procesador de 32 bits ARM y muchos más pines de entradas y salidas. Esta nueva placa solucionaría el problema del cuello de botella que se nombraba anteriormente, y solo por 10 € más en el presupuesto final. [20] [21]

- Implementar la misma idea de proyecto, pero en este caso que Arduino haga la función de un servidor web y el usuario final se conecte a una página web donde visualizar imágenes, videos, etc.

# 6. REFERENCIAS


[1] Houda Labios, Hossam Afifi, Constantino de Santis. *Wi-Fi Bluetooth, ZigBee and WiMax*. Springer-Verlag. New York Inc. 2006

[2] ZigBee Alliance. *Specifications*. [En línea]. EEUU: 2013.

Disponible en: <http://www.zigbee.org/Specifications/ZigBee/Overview.aspx>

[3] Valera Guerrera, I. *Transmisión de datos en Internet*. [En línea]. Santiago de Compostela: Marzo 2008.

Disponible en: <http://www.monografias.com/trabajos5/datint/datint.shtml>

[4] Shiffman, Daniel. *Interview with Casey Reas and Ben Fry*. [En línea]. 23 septiembre 2009.

Disponible en: <http://rhizome.org/editorial/2009/sep/23/interview-with-casey-reas-and-ben-fry/>

[5] Maik Schmidt, *Arduino: A Quick-Start Guide*. The Pragmatic Programmers.

4 de febrero 2011.

[6] Arduino. *Arduino Ethernet shield*. [En línea]. Italia.

Disponible en: <http://arduino.cc/en/Main/ArduinoEthernetShield>

[7] Arduino. *Arduino Wireless shield*. [En línea]. Italia.

Disponible en: <http://arduino.cc/en/Main/ArduinoWirelessShield>

[8] Libelium. *Xbee 802.15.14 SMA Module*. [En línea]. Coocking Hacks. Disponible en: <http://www.cooking-hacks.com/index.php/xbee-802-15-4-sma-module.html>

[9] Libelium. *WiFi Module for Arduino*. [En línea]. Coocking Hacks. Disponible en: <http://www.cooking-hacks.com/index.php/shop/arduino/wireless/wifi-module-for-arduino-roving-rn-xvee-xbee-compatible.html>

[10] Universidad Politécnica de Valencia. *Comparativa con otras plataformas*. [En Línea]. Valencia.

Disponible en: <http://www.androidcurso.com/index.php/recursos-didacticos/tutoriales-android/31-unidad-1-vision-general-y-entorno-de-desarrollo/99-arquitectura-de-android>

[11] Ed Burnette. *Hello, Android*. The Pragmatic Programmers; Edición: 3rd revised edition. 3 de agosto de 2010

[12] Apple Inc. *Cocoa Touch*. [En línea]. Disponible en:

<https://developer.apple.com/technologies/ios/cocoa-touch.html>

[13] Josh. *Velocidad de internet móvil*. [En línea]. Test Velocidad. Disponible en: <http://testvelocidad.eu/velocidad-internet-movil>

[14] Libelium. *Precios de dispositivos*. [En línea]. Coocking Hacks.

Disponible en: <http://www.cooking-hacks.com/>

[15] NetBeans Team. *NetBeans*. [En línea].

Disponible en: <https://netbeans.org/>

[16] Jorge V. *Sockets en Java (Cliente y Servidor)*. [En línea]. Abril 2007. Disponible en: <http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.UZuYTrVM98E>

[17] Arduino. *Arduino UNO*. [En línea]. Italia. Disponible en:

<http://arduino.cc/en/Main/ArduinoBoardUno>

[18] Oracle Dev. Team. *Función ArrayList*. [En línea]. Disponible en:

<http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html>

[19] Cplusplus. *Función Union*. [En línea]. Disponible en:

<http://www.cplusplus.com/doc/tutorial/other_data_types/>

[20] Arduino. *Arduino DUE*. [En línea]. Italia. Disponible en: <http://arduino.cc/en/Main/ArduinoBoardDue>

[21] Libelium. *Precio Arduino DUE*. [En línea]. Disponible en: <http://www.cooking-hacks.com/index.php/arduino-due.html>

# ANEXO A. CÓDIGO ARDUINO

En este anexo se incluyen todos los códigos de las funciones del Servidor en Arduino. Se han incluido los comentarios para que se entienda mejor que es lo que hace cada parte del programa.

## A.1 DECLARACIÓN DE VARIABLES

```
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>

//Declaración de variables

// Creamos una variable de tipos File, que será el archivo
File FileFlor;
//Declaramos el buffer con 225 bytes
byte buffer[225];
int TAM_MAX = 255;
int zero = 0;

// Introducir la dirección MAC, IP y numero de PUERTO para el SERVER

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
IPAddress serverIP(192,168,1,128);
int serverPort=8888;

// Iniciamos la libreria del server ETHERNET
EthernetServer server(serverPort);
```

## A.2 FUNCIÓN SETUP

```
void setup()
{
  Serial.begin(9600);
  // Iniciamos la conexion ethernet y el server:
  Ethernet.begin(mac, serverIP);
  server.begin();
  Serial.println("Server en MARCHA");
  Serial.println("Initializing SD card...");
  // En el shield de Ethernet, CS es el pin 4. Esta definido como
      salida por defecto.
  // Aunque no se utilice como pin CS, el hardware del pin SS
  // (10 en la mayoria de placas Arduino, 53 en la MEGA) se debe dejar
      como salida
  // o la librería SD no funcionará.
  pinMode(10, OUTPUT);
```

```
  // Comprobamos si funciona correctamente la tarjeta SD.
  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
}
```

## A.3 FUNCIÓN LOOP

```
void loop()
{
  // Escucha de clientes
  EthernetClient client = server.available();

  if (client) {
    while (client.connected()) {
      // Mientras el cliente este conectado se ejecutara el siguiente
      código
      if (client.available()) {

        //Leemos la id del archivo que quiere descargar el usuario
        String Nombre;
        int id;
        id=client.read();

      //La relacionamos con los archivos guardados en la SD
      if (id==0){
          Nombre = "starwars.jpg";
          Serial.println("Ha escogido la imagen del cuadro.");
        }

        if (id==1){
          Nombre = "archivo2.txt";
          Serial.println("Ha escogido el texto del cuadro.");
        }

        if (id==2){
          Nombre = "dinero.mp3";
          Serial.println("Ha escogido el audio del cuadro.");
        }

        //Mostramos por pantalla la id y el nombre del archivo
        Serial.println(id);
        Serial.println(Nombre);

        //Cambiamos la variable de string a char
        char filename[15];
        Nombre.toCharArray(filename, 15);

        //Abrimos imagen
        FileFlor = SD.open(filename);

        if (FileFlor) {
          Serial.println("Abriendo archivo...");
          Serial.println("Transfiriendo al cliente..... ");
```

```
      }
      else {
        Serial.println("error opening image");
      }
      //Utilizamos la función "UNION" para unir un entero con un
      //Array de bytes (2).
      //Esto se realiza para enviarlo como Array de Bytes, ya que
      los int de Arduino y Java no se entienden.
      union{
        int i;
        byte b[2];
      }
      u;

      if (FileFlor) {

        int i = 0;
        //Mientras haya bytes disponibles por leer se ejecutará el
        siguiente codigo
        while (FileFlor.available()>0) {

          buffer[i] = FileFlor.read();

          if(i==TAM_MAX-1){
            //Union entre el entero y el array de bytes nombrado
            anteriormente.
            u.i = TAM_MAX;
            //Enviamos el tamaño que tiene el paquete para la
            gestión del mismo en el código Java.
            client.write(u.b,2);

            //Envio buffer con datos
            client.write(buffer,TAM_MAX);
            Serial.println("Transfiriendo al cliente..... ");
            i=-1;
          }
          i++;
        }

        if(i>0){
          //Se envía el último paquete que es el que tendrá
          diferente tamaño
          u.i = i;
          client.write(u.b,2);
          client.write(buffer,i);
        }

        //Enviamos que el tamaño es 0, para que se salga del bucle
          en el cliente.
        u.i = zero;
        client.write(u.b,2);
      }

      Serial.println("Transferencia hecha");
      FileFlor.close();
      //Cerramos la conexión
      client.stop();
      Serial.println("Conexion terminada");
      Serial.println("Esperando siguiente conexion...");
    }
  }
```

```
  }
  //Delay de espera para que el cliente reciba los datos
  delay(1);
}
```

# ANEXO B. CÓDIGO ANDROID

En este anexo se mostrará el código que se ha utilizado para crear la aplicación, así como la función que permite conectarse al servidor de Arduino. También se han incluido los comentarios sobre las funciones que se utilizan.

## B.1 PRIMER ACTIVITY. MENÚ PRINCIPAL

```
package pfc.trasnferencia;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class PFC extends Activity {
      //Declaración de Botones
    Button Boton1;
    Button Boton2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Creo el botón que implementa la acción de trasferencia para
          ir al siguiente activity (CUADROS)
        Boton1 = (Button) findViewById(R.id.Boton1);
        //Creo el botón que implementa la acción de trasferencia para
          ir al siguiente activity (ABOUT)
        Boton2 = (Button) findViewById(R.id.Boton2);
        Boton1.setText("CUADROS");
        Boton2.setText("ABOUT");

        //Acciones cuando se presiona el botón. Se carga el activity
          correspondiente
        Boton1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(PFC.this, Salas.class);
                intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
                startActivity(intent);
            }
        });
        Boton2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(PFC.this, About.class);
                intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
                startActivity(intent);
```

```
            }
        });
    }
}
```

## B.2 SEGUNDO ACTIVITY. SELECCIÓN DE CUADRO A DESCARGAR

```
package pfc.trasnferencia;


import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;


public class Salas extends Activity {

//Declaro Listview, ArrayAdapter y el boton
//para descargar
    private ListView ListaObjetos;
    private ArrayAdapter<String> adapter;
    private String cadena;
    private Button Boton;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
//Cargamos el Layout del Activity Cuadros
        setContentView(R.layout.salas);

//Llenamos la lista con los objetos a descargar usando un vector
//Declaro el array y lo relleno para pasarlo al adapter
        final ArrayList<String> Lista = new ArrayList<String>();
        Lista.add("Cuadro1.jpg");
        Lista.add("Archivo2.txt");
        Lista.add("Audio1.mp3");


//Creo el objeto Listview que contendra los objetos de la lista
        ListaObjetos = (ListView) findViewById(R.id.ListaObjetos);
```

```
//Construyo el adaptador y le paso de parametro el
//array con los objetos
        adapter = new ArrayAdapter<String>
            (this, android.R.layout.simple_list_item_single_choice,
Lista);

//Linko el adaptador con el listview y le asigno propiedades
        ListaObjetos.setAdapter(adapter);
        ListaObjetos.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        ListaObjetos.setItemChecked(0, true);

//Creo el boton que implementa la accion de trasnferencia
        Boton = (Button) findViewById(R.id.Boton);
        Boton.setText("Descarga");

        Boton.setOnClickListener(new View.OnClickListener() {
             public void onClick(View v) {
                   try {

//Función para crear marcador de tiempo y luego calcular tiempo
//transcurrido
            Date interestingDate = new Date();

//Creamos el socket con la ip y puerto donde se tiene que conectar
            Socket cs = new Socket("192.168.1.128", 8888);

//Implementamos las salidas y entradas del programa
DataOutputStream Out = new DataOutputStream(cs.getOutputStream());
DataInputStream In = new DataInputStream(cs.getInputStream());

//Enviamos al Server el archivo que se desea descargar
            int id = ListaObjetos.getCheckedItemPosition();
            Out.write(id);

//Creamos el ArrayList donde se guardará el archivo al recibirlo
            ArrayList<Byte> buffer;
            buffer = new ArrayList<Byte>();

//Creamos un vector de tipos Byte llamado data.
           byte[] data;

//Implementamos codigo para decodificar los 2 bytes que
//envía Arduino, siendo el tamaño del paquete entrante
            int n = 2;
            byte[] chunk = new byte[n];
            In.read(chunk);
            short value = 0;
// Obtenemos 2 bytes, unsigned 0..255
            int low = chunk[0] & 0xff;
            int high = chunk[1] & 0xff;
            value =  (short) (high << 8 | low) ;


//Tamaño del vector
            int tam = value;

            while(tam>0){

            //Mientras el tamaño sea mayor que cero,
            //se ejecutará el siguiente código
            //Se crea vector con el tamaño recibido
```

```
                    data = new byte[tam];
              //Se mete el paquete en el vector data
                    In.read(data);
                   for(int i=0; i<tam;i++){
                    //Almacenamos los datos en el ArrayList
                       buffer.add(data[i]);
                           }

                    //Se vuelve a leer el tamaño del paquete

                    In.read(chunk);
                    value = 0;
                    // Obtenemos 2 bytes, unsigned 0..255
                    low = chunk[0] & 0xff;
                    high = chunk[1] & 0xff;
                    value =  (short) (high << 8 | low) ;
                    tam = value;
                    }

//Se cierran entradas, salidas de datos y el socket
                In.close();
                Out.close();
                cs.close();
                toastShow("Descargado");

//Calculamos tiempo transcurrido
float tiempo=(float) ((new Date()).getTime() -
interestingDate.getTime());

//Se traspasan los datos del ArrayList a un Buffer final
                byte[] archivo =new byte[buffer.size()];
                for(int i=0; i<buffer.size();i++){
                   archivo[i]=buffer.get(i);
                }

               toastShow("Guardando archivo...");
               toastShow("Tiempo: " + tiempo/1000 + " segundos");

//Utilizando la siguiente función, convertimos el buffer en el archivo
//deseado asignándole el nombre con el formato

save_file(archivo,Lista.get
(ListaObjetos.getCheckedItemPosition()));

//Esta función se utiliza para, una vez descargado el archivo,
//se reproduzca/muestre con el visor por defecto del móvil Android
Intent intent = new Intent();
intent.setAction(android.content.Intent.ACTION_VIEW);
File file = new
File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/"
+ Lista.get(ListaObjetos.getCheckedItemPosition()));

if (ListaObjetos.getCheckedItemPosition() == 0) {
      intent.setDataAndType(Uri.fromFile(file), "image/*");
      startActivity(intent);
}
if (ListaObjetos.getCheckedItemPosition() == 1) {
      intent.setDataAndType(Uri.fromFile(file), "text/*");
      startActivity(intent);
}
if (ListaObjetos.getCheckedItemPosition() == 2) {
```

```
      intent.setDataAndType(Uri.fromFile(file), "audio/*");
      startActivity(intent);
}

                  } catch (IOException ex) {
                        //Mostramos por pantalla el nombre del archivo
toastShow(ex.toString());
Logger.getLogger(PFC.class.getName()).
log(Level.SEVERE, null, ex);
                  }
              }
          });
    }
}
```

Durante la ejecución del código anterior, se llama a dos funciones: save_file y toastshow.

### B.2.1 FUNCIÓN SAVE_FILE

```
    private void save_file(byte[]file,String Nombre){

    boolean sdDisponible = false;
    boolean sdAccesoEscritura = false;

    //Comprobamos el estado de la memoria externa (tarjeta SD)
    String estado = Environment.getExternalStorageState();

    if (estado.equals(Environment.MEDIA_MOUNTED))
    {
        sdDisponible = true;
        sdAccesoEscritura = true;
    }
    else if (estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
    {
        sdDisponible = true;
        sdAccesoEscritura = false;
    }
    else
    {
        sdDisponible = false;
        sdAccesoEscritura = false;
    }
        try{
                  //Se introduce la ruta de la ubicación de la SD
            File ruta_sd = Environment.getExternalStorageDirectory();

                  //Se crea un nuevo fichero con el nombre
            File f = new File(ruta_sd.getAbsolutePath(), Nombre);


            FileOutputStream fout =new FileOutputStream(f);

                  //Mostramos por pantalla el tamaño del archivo
                  guardado
```

```
            toastShow(Integer.toString(file.length));

                  //Se escribe el archivo
            fout.write(file);
            fout.close();
            toastShow ("guardado");
        }catch (Exception ex){
            Log.e("Ficheros", "Error al escribir fichero a tarjeta
SD");
        }
    }
```

### B.2.2 FUNCIÓN TOASTSHOW

```
    private void toastShow(String text){
        Toast toast = Toast.makeText(this, text, Toast.LENGTH_SHORT);
        toast.setGravity(0,0,200);
        toast.show();
    }
```

## B.3 TERCER ACTIVITY. PANTALLA DE INFORMACIÓN SOBRE EL PROYECTO

```
package pfc.trasnferencia;

import android.app.Activity;
import android.os.Bundle;

public class About extends Activity {

    @Override

      //Solo se necesita cargar el nuevo Layout para esta
      //Activity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.about);
    }
}
```

# ANEXO C. CÓDIGO XML PARA ANDROID

## C.1 CÓDIGO XML PARA EL LAYOUT DEL PRIMER ACTIVITY

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background">

<ImageView
        android:id="@+id/imagen2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true"

        android:scaleType="fitCenter"
        android:src="@drawable/logo"
        />

  <Button android:id="@+id/Boton1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="CUADROS"
        android:textSize="20sp"
        android:layout_centerVertical="true"
        android:layout_marginTop="10dp"/>

  <Button android:id="@+id/Boton2"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textSize="20sp"
          android:layout_centerHorizontal="true"
          android:layout_marginTop="10dp"
          android:text="ABOUT" />
</LinearLayout>
```

## C.2 CÓDIGO XML PARA EL LAYOUT DEL SEGUNDO ACTIVITY

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FF000000">

   <ListView android:id="@+id/ListaObjetos"
        android:layout_width="fill_parent"

        android:layout_height="10dip"
        android:layout_weight="1" />
```

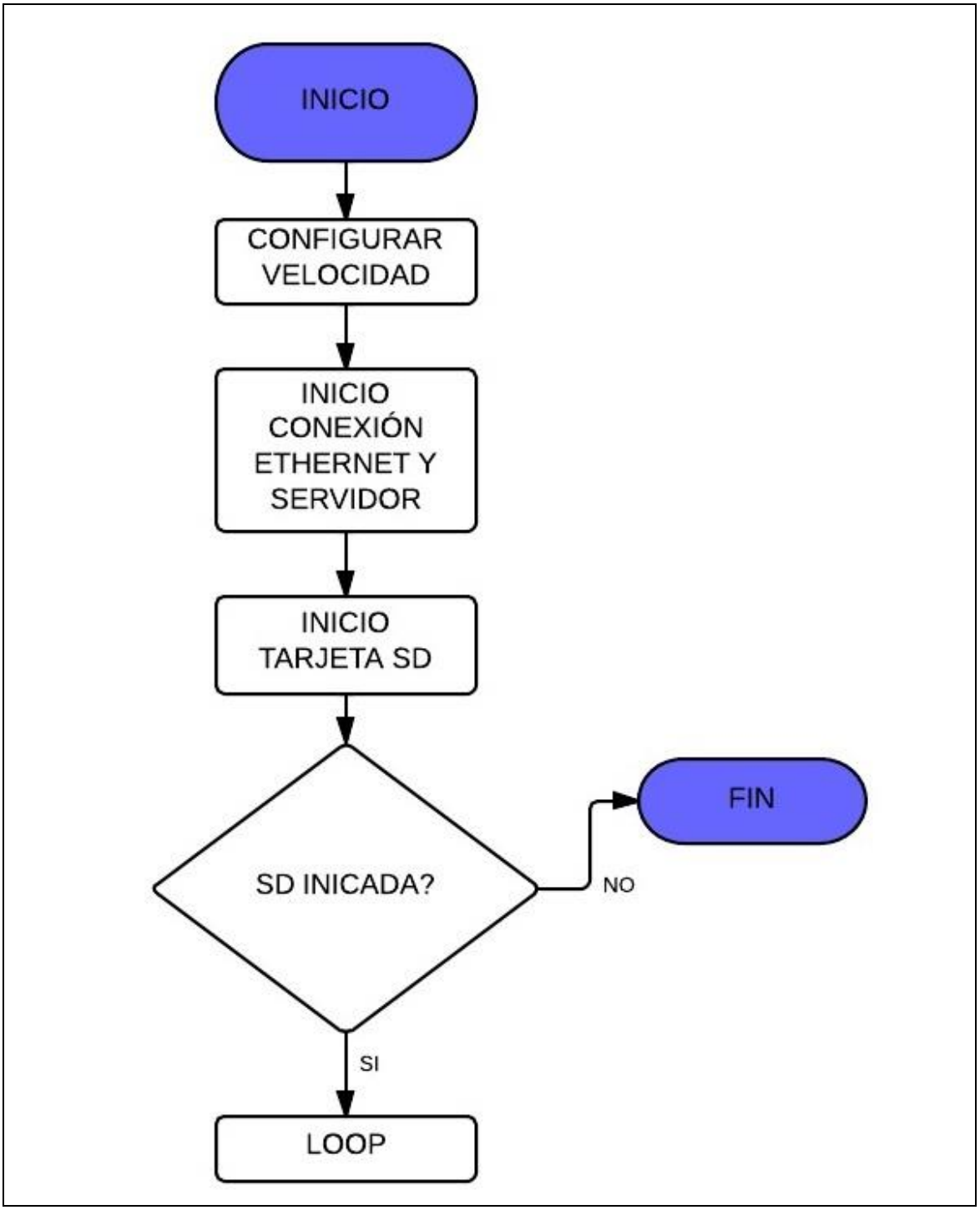
```
  <Button android:id="@+id/Boton"
     android:layout_gravity="center"
     android:layout_height="wrap_content"
     android:layout_width="fill_parent" />

</LinearLayout>
```

## C.3 CÓDIGO XML PARA EL LAYOUT DEL TERCER ACTIVITY

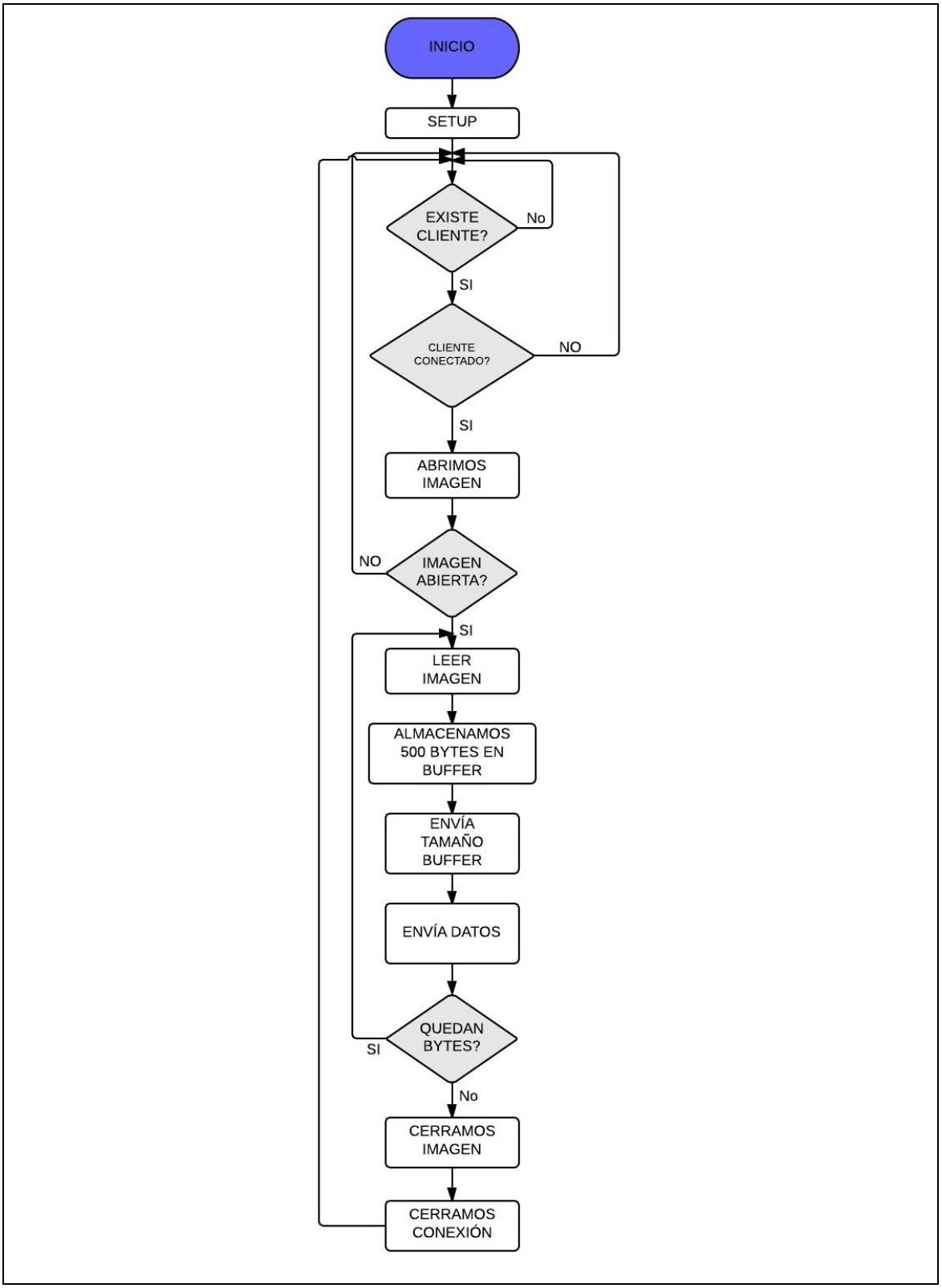
```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:background="@drawable/background">
    <TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_marginTop="20dp"
          android:gravity="center_horizontal"
          android:textColor="#000000"
          android:text="PROYECTO FINAL DE GRADO"/>

    <TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_marginTop="20dp"
          android:gravity="center_horizontal"
          android:textColor="#000000"
          android:textStyle="bold"
          android:textSize="20sp"
          android:text="Diseño de aplicación móvil para la
comunicación inalámbrica de señales audiovisuales"/>

    <ImageView
        android:id="@+id/imagen2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true"
        android:scaleType="fitCenter"
        android:src="@drawable/explicacion"/>

    <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:textColor="#000000"
            android:text="Autor: Alberto Esteban Pérez"/>
</LinearLayout>
```

# ANEXO D. DIAGRAMAS DE FLUJO

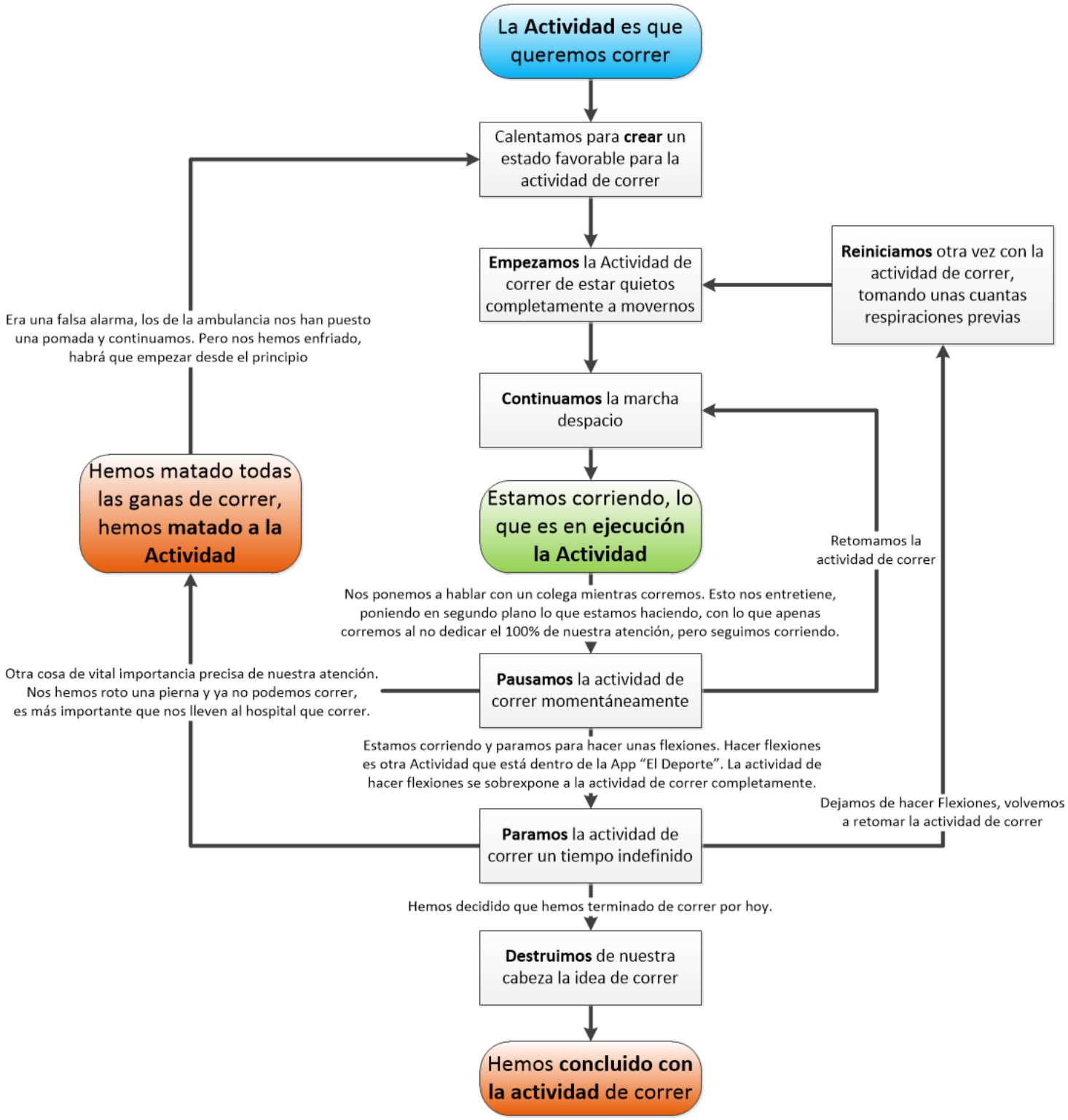
## D.1 DIAGRAMA DE FLUJO DE LA FUNCIÓN SETUP



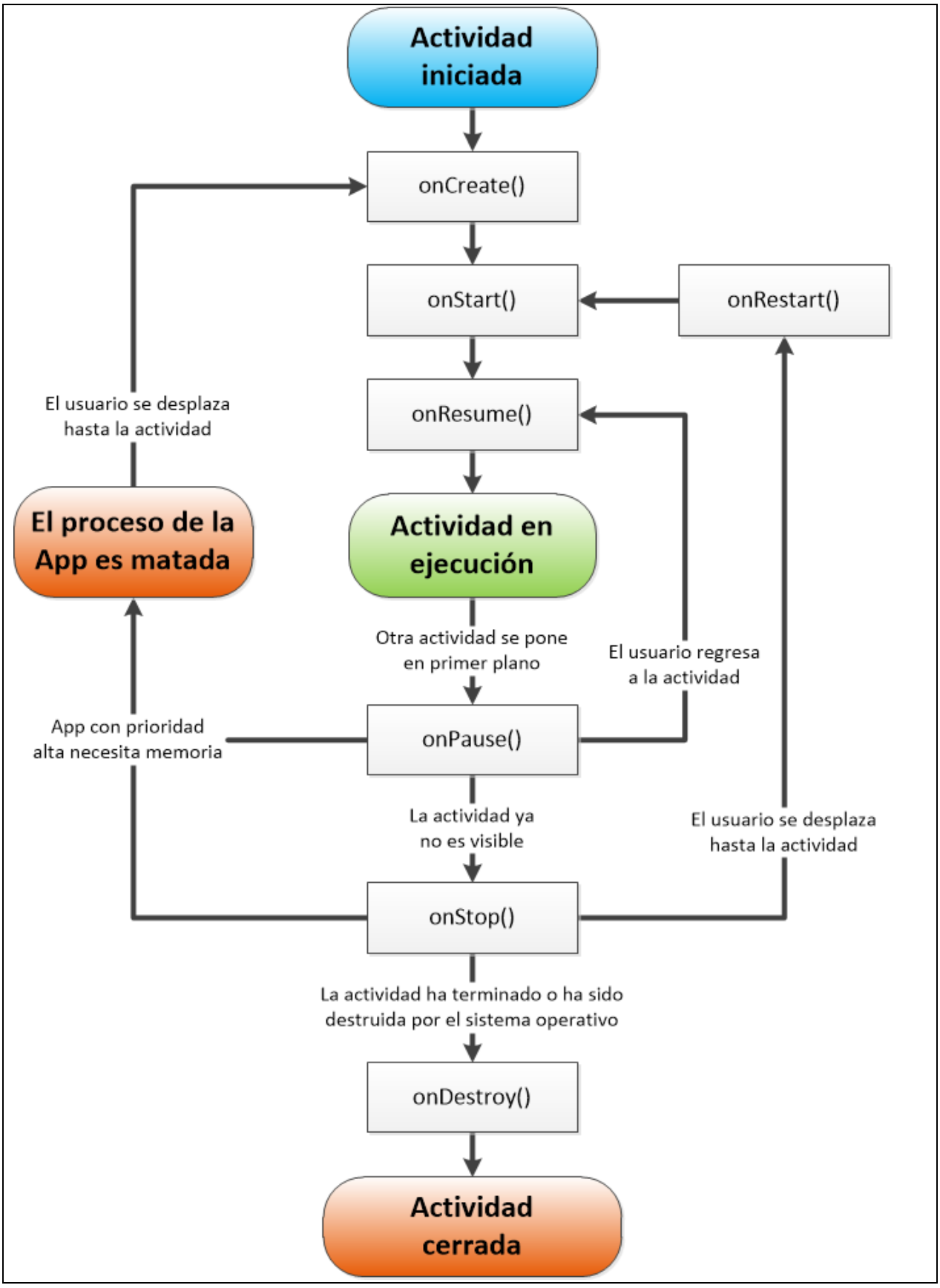
## D.2 DIAGRAMA DE FLUJO DE LA FUNCIÓN LOOP



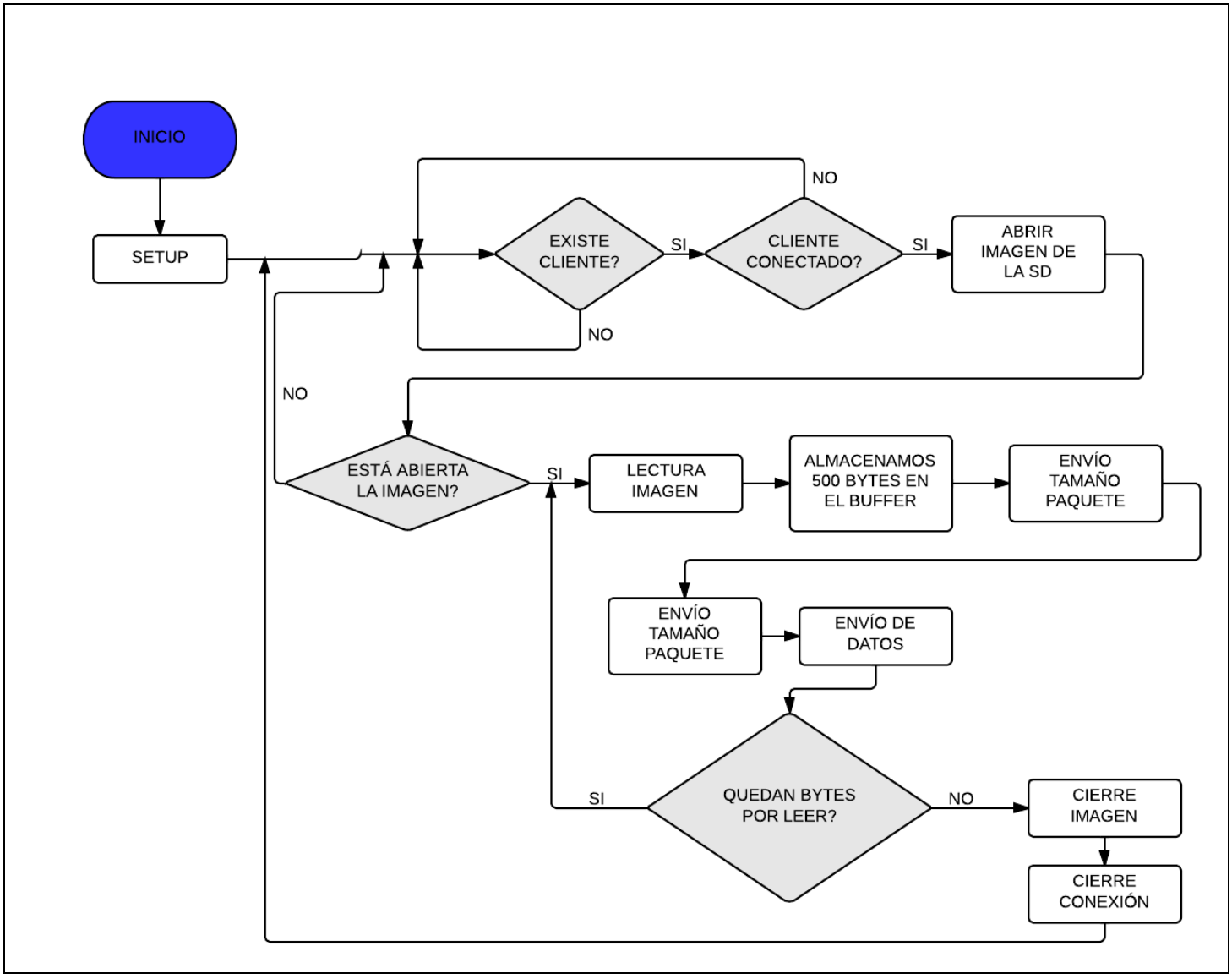

## D.3 DIAGRAMA DE FLUJO DE UNA ACTIVITY (MODIFICADO)




## D.4 DIAGRAMA DE FLUJO DE UNA ACTIVITY (OFICIAL)




D.5 DIAGRAMA DE FLUJO DE LA FUNCIÓN DEL CLIENTE


INICIO
SETUP
EXISTE CLIENTE?
SI
CLIENTE CONECTADO?
SI
NO
NO
ABRIR IMAGEN DE LA SD
NO
ESTÁ ABIERTA LA IMAGEN?
SI
LECTURA IMAGEN
ALMACENAMOS 500 BYTES EN EL BUFFER
ENVÍO TAMAÑO PAQUETE
ENVÍO TAMAÑO PAQUETE
ENVÍO DE DATOS
QUEDAN BYTES POR LEER?
SI
NO
CIERRE IMAGEN
CIERRE CONEXIÓN