Universidad de Valladolid

Escuela de Ingenierías Industriales

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de aplicación Android para localización de dispositivos Bluetooth en interiores

Autor : Postigo Gómez, Victor Manuel
Tutor : Pablo Gómez, Santiago de

Julio 2018

Contenido

1 Resumen

1.1 Resumen

Desarrollo de un programa orientado a objetos en Java para generar una aplicación Android que trate de localizar dispositivos Bluetooth activos dentro de un área determinada. El resultado de la localización debe ser un punto en el plano de trabajo, el cual se podrá determinar de forma aproximada con la intensidad de la señal Bluetooth medida por varios dispositivos “baliza”, y se calcularán las ubicaciones desde un dispositivo “maestro” que mostrará los resultados.

1.2 Palabras clave

Bluetooth Android Localización Java API

2 Absract

2.1 Abstract

Development of an object-oriented program in Java to generate an Android application that tries to locate active Bluetooth devices within a certain area. The result of the location must be a point in the work plane, which can be determined approximately with the intensity of the Bluetooth signal measured by several "beacon" devices, and the locations will be calculated from a "master" device that will show the results.

2.2 Key words

Bluetooth Android Location Java API

3 Descripción del proyecto

3.1 Objetivo del proyecto

El objetivo de este proyecto consiste en la localización de dispositivos que se encuentren usando servicios de la comunicación Bluetooth dentro de un espacio interior como puede ser una vivienda, un supermercado, una oficina o cualquier otro tipo de habitáculo donde pueda ser de utilidad el hecho de localizar un dispositivo Bluetooth.

Se pretende obtener una ubicación aproximada de los dispositivos en función de la intensidad de señal Bluetooth con la que estos hayan sido detectados.

Para el desarrollo de un sistema capaz de llevar a cabo todo tipo de interacciones con las comunicaciones Bluetooth se realizará una aplicación para el sistema operativo móvil Android, el cual dispone de herramientas y características que permitirán detectar, identificar y establecer conexiones con otros dispositivos que se encuentren consumiendo recursos de este tipo de comunicaciones.

Por tanto, los objetivos del proyecto serán:

- Aprender y conocer los fundamentos de la programación de aplicaciones Android con el uso de las herramientas y APIs disponibles.
- Localizar dispositivos que se encuentren dentro de un área determinada utilizando los recursos de la comunicación Bluetooth.
- Conectar y realizar transmisiones con comunicación Bluetooth.

3.2 Enfoque preliminar

Con las herramientas disponibles en las APIs de Android destinadas a las comunicaciones Bluetooth es posible desarrollar un programa, ya sea en lenguaje de programación Java o C++, ambos diseñados para la programación orientada a objetos, que sea capaz de detectar la presencia de un dispositivo que tenga activadas sus comunicaciones Bluetooth y su posicion dentro de un entorno cerrado, siempre y cuando este dispositivo desee ser encontrado por

cualquier otro. Estas herramientas de Android no solo permiten detectar, sino que se pueden identificar estos dispositivos obteniendo el nombre del mismo y su dirección de acceso al medio inequívoca, o del inglés *Medium Access Control* (MAC).

Con el uso de las herramientas de Android y de dispositivos que utilicen este sistema operativo, se podrá desarrollar un sistema de localización por Bluetooth de forma más rápida y sencilla que teniendo que recurrir al diseño de dispositivos hardware específicos para la tarea a realizar, lo que conllevaría un estudio más laborioso y una inversión de tiempo mayor.

Para obtener una localización de los dispositivos no solo bastará con la identidad del dispositivo, sino que se podrá aprovechar la intensidad de la señal emitida por el dispositivo a localizar, esta intensidad de la señal o RSSI (*Received Signal Strength Indicator*) varía en función de la distancia a la que se encuentre el dispositivo detectado del dispositivo detector, conociendo esto se podrá obtener una expresión con la que poder obtener información de la distancia.

Para poder ubicar el dispositivo dentro de un plano determinado se necesitará de varios dispositivos detectores, ya que únicamente con un detector se obtiene una distancia radial y el dispositivo emisor de la señal se podría encontrar en cualquier punto de la circunferencia definida por dicho radio.

De esta forma, si en lugar de un detector se añadiera otro más, dispuesto a una distancia concreta del primero, se obtendrían dos puntos en el plano definidos por los puntos de corte entre las circunferencias obtenidas por cada uno de ellos como se muestra en la Figura 1.

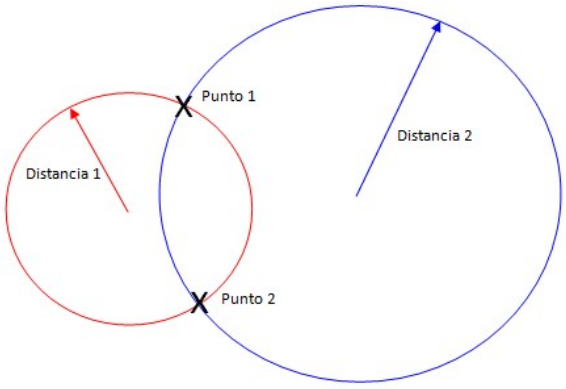

Figura 1 : Puntos obtenidos con dos detectores

Ahora solo restaría el descartar uno de los dos puntos ya que solo en uno de ellos se encontrará el dispositivo a localizar, para ello se plantean dos variantes:

1. Añadir una baliza más que permita determinar el punto, ya que en función de la distancia obtenida en esta tercera baliza, evocará a un único punto de los dos encontrados por las dos primeras.
   Para ubicar posteriormente el punto en el espacio será necesario conocer la posición de los centros de las circunferencias que serían las posiciones exactas de

las tres balizas en el espacio, como se indica en la Figura 2, para obtener un resultado concreto de las ecuaciones de intersección de circunferencias.

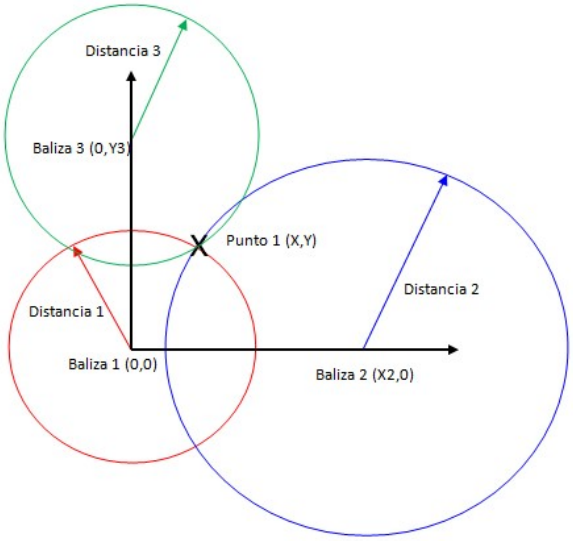

Figura 2 : Punto obtenido empleando tres balizas

2. Sin necesidad de añadir ninguna baliza más, con las dos que se tenían en la situación inicial, es posible discriminar uno de los dos puntos obtenidos por la intersección de las dos circunferencias en función del semiplano en que se encuentre cada uno de ellos.
   Si se sitúa una baliza en el origen y la otra se define en un punto del eje de abscisas, de tal manera que entre ambas se defina una frontera y quede definido también indirectamente el eje de ordenadas.
   De esta forma se hace posible la definición de una región de interés dentro del plano.

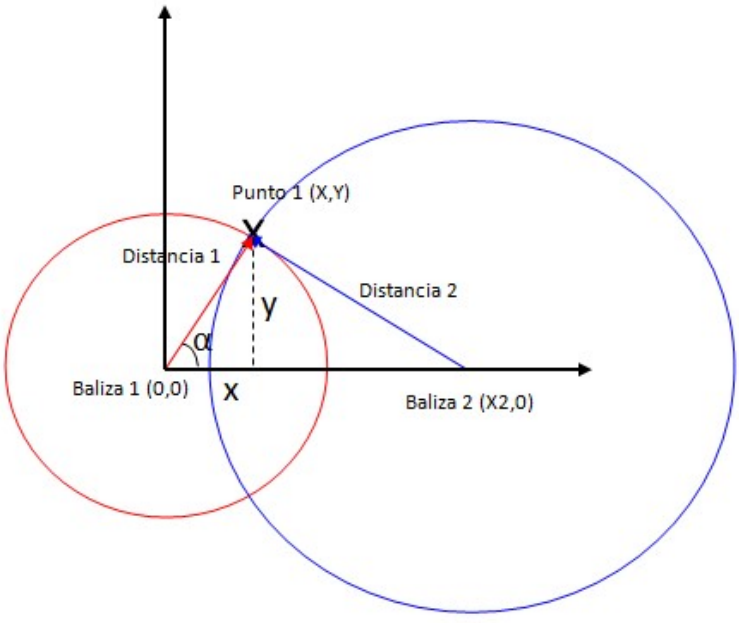

Figura 3 : Punto obtenido con dos balizas y discriminación de área

Con estas opciones, la primera de ellas es más robusta en cuanto a la localización del dispositivo objetivo pero requiere de tres dispositivos dentro del sistema que se encuentren buscando, y tratándose de dispositivos Android no es tan sencillo disponer de ellos.

Por tanto, aunque menos robusto, se decantará por la segunda opción de dos balizas localizadoras ya que es el método más sencillo, que define la posición en función de si se encuentra en la región de interés o no, y permite profundizar en el estudio en las comunicaciones entre balizas y en el desarrollo de la aplicación Android.

3.3 Aplicaciones

Teniendo en cuenta las especificaciones y alternativas al problema, los sistemas de localización de dispositivos Bluetooth en interiores pueden ser de utilidad para aplicaciones como las mostradas a continuación.

Localización de personas u objetos en lugares públicos

Dentro de espacios como pueden ser museos, centros educativos o administrativos puede ser útil la localización y posicionamiento de las personas que se encuentran en el interior de los mismos.

En museos es posible establecer dos balizas por sala de exposición y en función de la ubicación del visitante, que deberá disponer de un dispositivo Bluetooth concreto, se podrán activar diversas animaciones o comentarios de interés.

En centros educativos puede ser útil el hecho de localizar y posicionar a las personas que se encuentran en cada aula diariamente para controlar la asistencia a las clases.

Localización de personas u objetos en lugares privados

En lugares privados como una vivienda es más sencillo aplicar dicho sistema de localización y aplicarlo por ejemplo a la domótica, para que dependiendo de la posición de la persona dentro de una habitación se enciendan las lámparas o cualquier dispositivo que desee controlar el usuario.

En otros espacios de tipo privado como oficinas, se puede emplear esta tecnología de localización por balizas dotando a los empleados de dispositivos Bluetooth similares y de esta forma poder conocer la posición de los mismos, en caso de que estén en su puesto de trabajo o se encuentren tomando un descanso.

Localización y contaje de objetos en almacenes

Con este tipo de tecnología de localización y el auge de la industria 4.0, que intenta hacer de cada objeto un dispositivo conectado, sería posible dotar a cada objeto de un dispositivo Bluetooth de bajo coste y consumo para que pueda ser localizado.

Es posible llevar un control de stock dentro de un pequeño almacén si cada uno de los objetos que contiene disponen de pequeños módulos Bluetooth, de esta forma será posible llevar un contaje de elementos y saber en qué lugar del almacén se encuentran.

Lo mismo sucedería en el sector del transporte de mercancías, para saber si todo lo que se envió en el origen ha llegado correctamente al destino, se puede disponer de un sistema de localización dentro del transporte que tenga control constante sobre la ubicación de los elementos que contiene y notificar si en algún momento desaparecen del área de búsqueda.

4 Introducción

4.1 Navegación y posicionamiento global

La localización y la ubicación en el medio físico ha sido, desde la creación de los primeros navíos, uno de los grandes intereses del hombre. Desde la orientación por el posicionamiento de las estrellas con el uso de astrolabios hasta el sistema actual de navegación y localización por satélite que permite la localización por coordenadas terrestres.

En la antigüedad se comenzaron a utilizar instrumentos como los astrolabios para obtener una orientación o posición sobre el plano terrestre, estos instrumentos se usaban para determinar la posición de las estrellas en el cielo y, así mismo, era posible calcular la hora y latitud a la que se encontraba la persona.

Más tarde aparecieron las brújulas y los mapas con los cuales se podría obtener una posición a menor escala y con mayor precisión con técnicas de triangulación tomando como referencia objetos físicos como grandes elevaciones de terreno.

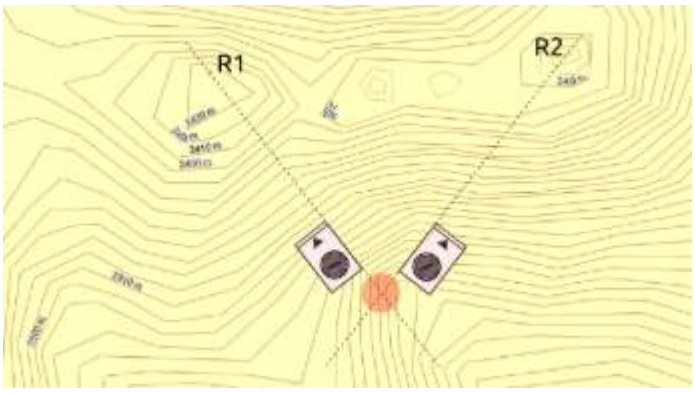

Figura 4 : Localización por triangulación con brújula

Con las nuevas tecnologías y el desarrollo de nuevos objetos y medios de transporte surgieron los sistemas de navegaciones inerciales, capaces de estimar posición, aceleración y orientación del objeto en cuestión.

Seguido de estos aparecieron los sistemas de radio localización, los cuales usan las ondas de radio emitidas por estaciones de radio y, en función de los tiempos de recepción de dichas

señales es posible obtener una posición específica, la teoría de funcionamiento de estas es similar a medida de distancias con sensores de ultrasonidos que calculan la distancia al objeto usando los tiempos de envío y recepción.

Finalmente, surgieron los sistemas de localización basados en el uso de satélites, sistemas como el GPS cuyo funcionamiento es similar al de los sistemas de radiolocalización, cada satélite emite señales a una misma frecuencia y, en función de la distorsión con la que se reciba la onda y las coordenadas en las que se encuentre el satélite en cuestión se puede obtener una posición con bastante precisión y rapidez.

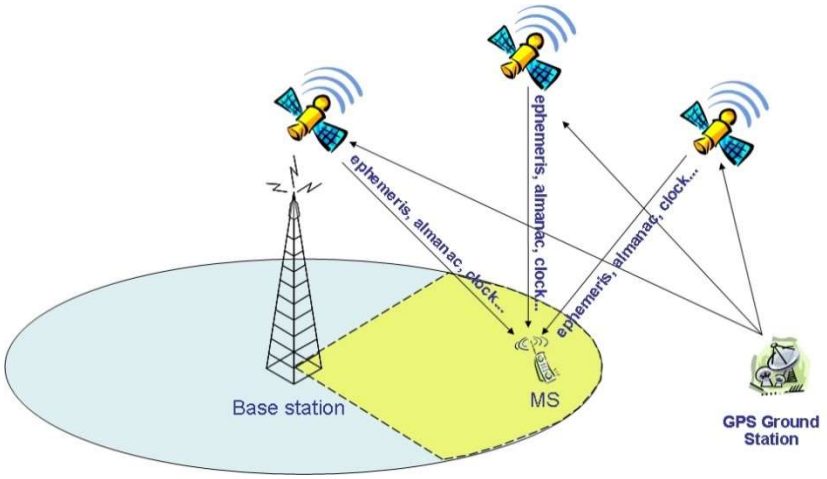

Figura 5 : Localización por satélite

4.2 Sistemas de posicionamiento local

Todos estos sistemas hacen referencia a la geolocalización, por tanto, las precisiones estimadas serán como mucho del término de metros e incluso decámetros, pero cuando la localización de los objetos se quiere llevar a cabo en espacios cerrados o interiores la precisión de la localización ha de ser mucho mayor, dependiendo también de las dimensiones del objeto a localizar.

La localización de objetos en interiores da lugar a los sistemas de posicionamiento local, los cuales están basados en la nuevas tecnologías de la comunicación por radio frecuencia, como pueden ser las tecnologías Wii o Bluetooth.

Para el posicionamiento en interiores se pueden describir una serie de tecnologías, cada cual tiene su ventaja particular y sus convenientes desventajas.

1. Ultrasonidos: Tecnología muy común en el mundo de la sonorización y la medición de distancias, a partir de señales sonoras es capaz de detectar la distancia a la que se encuentra un objeto determinado.

   Los sensores se diseñan de tal forma que contienen un emisor de señal que se encargara de propagar las ondas sonoras en una dirección, y otro elemento que

será el receptor de la señal. El fundamento de esto consiste en que, el emisor propaga la señal en una dirección, la señal emitida se topara con un objeto y, a no ser que el objeto absorba la señal, rebotará y llegara hasta el elemento receptor del sensor. De esta forma, conociendo la velocidad del sonido y los tiempos desde que la señal se propaga hasta que se recibe se puede obtener una distancia hasta el objeto con una precisión de centímetros y dentro de un rango de 2 cm hasta unidades de metros dependiendo del diseño del sensor.

Esta tecnología presenta el inconveniente de que, a menos que incluya un sistema móvil, únicamente obtendrá las distancias en una única dirección y sentido.

Figura 6 : Localización por ultrasonidos, sonar

2. Radio de banda ultra ancha: Esta tecnología consiste en la transmisión de datos de forma inalámbrica a baja potencia dentro de un ancho de banda de en torno a 500MHz, de este modo, dicho ancho de banda transmitirá a una relativa velocidad constante. Estos sistemas están basados mayormente en la comunicación y transmisión de datos a distancia más que orientados a aplicación de posicionamiento.

   En términos de localización, estos sistemas interesan en el aspecto de la transmisión, el emisor envía el mensaje y el receptor lo devuelve de forma cuasi-simultanea, de esta forma, y con los tiempos de comunicación se podrá obtener una distancia entre emisor y receptor. Presenta la ventaja de que las velocidades de transmisión son muy altas y por tanto esto mejora la precisión en la medida de la distancia.

   Por el contrario, emisor y receptor han de estar totalmente sincronizados y el hecho de no estarlo podría significar deformaciones en la transmisión, que afectaría a la medida de la distancia.

3. RFID(*Radiofrecuency Identificator*): Esta tecnología sigue el mismo comportamiento emisor-receptor que en el caso de radio de banda ultra ancha, solo que en este caso se presenta la venta de que el receptor actúa de forma pasiva y no es necesaria ninguna sincronización entre elementos, también presenta ventajas como la independencia de obstáculos.

El funcionamiento de estos sistemas consiste en que, el emisor envía una señal de radio al receptor dentro de un ángulo determinado, el receptor únicamente contiene información y en el momento en que recibe la señal, este envía la información que contiene sin realizar ningún procesamiento de la señal original.

El cálculo de la distancia emisor-receptor se realiza de la misma forma que en el caso de la tecnología anterior, con los tiempos de emisión-recepción.

Esta tecnología presenta el inconveniente de que las distancias han de ser del orden de 1m como máximo.

Figura 7 : Localización RFID, etiquetas RFID

4. Visión artificial: Esta tecnología puede ser una de las más favorables para la realización de una aplicación de localización y posicionamiento de objetos en interiores ya que, en la actualidad ya se encuentran desarrollados una serie de algoritmos de reconocimiento de objetos y cálculo de distancias.

   Con la visión artificial se pueden obtener patrones o formas geométricas determinadas para la identificación y localización de objetos. Tomando referencias del mundo físico y presentes en las imágenes tomadas por los sensores se pueden obtener cálculos de distancias con bastante precisión y rango de medida.

   Estos sistemas presentan inconvenientes como el coste o que, el objeto a detectar debe encontrarse dentro de la imagen captada por el sensor, eso sin contar con el factor de la iluminación ya que, para identificar objetos debe haber un nivel adecuado de luz.

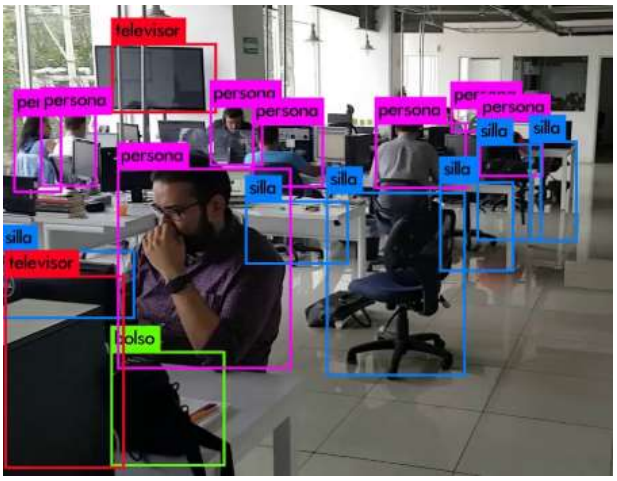

Figura 8 : Localización y rastreo por visión artificial

5. Zigbee: Tecnología basada en redes inalámbricas y de bajo consumo, consta de una red de dispositivos interconectados entre sí, de tal forma que, uno de ellos coordina la mensajería global y los demás procesan y envían información.

   Una de las ventajas de este tipo de tecnología es el bajo volumen de datos transmitido y por tanto, la velocidad de transmisión. También tiene ventajas como que, al ser una red de dispositivos interconectados, los obstáculos no son un problema.

   Esta tecnología, enfocada a la detección de dispositivos e interiores, es una buena solución dado que cualquiera de los nodos de la red puede detectar el objeto o dispositivo presente y, por medio de sus conexiones, hacer llegar la información hasta un controlador general remoto.

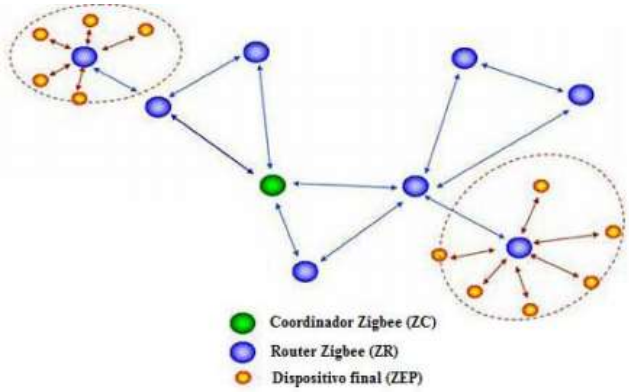

Figura 9 : Arquitectura de red Zigbee

6. WiFi: Es posible medir la posición o localización de un dispositivo con la tecnología WiFi siempre y cuando el dispositivo a localizar se encuentre usando la misma tecnología en ese momento.

   Hay dos maneras de determinar la posición con la tecnología WiFi, una de ellas es con la intensidad de la señal de radio o la radiación emitida por el dispositivo a detectar (RSSI) que varía en función de la distancia a la que se encuentre el detector. Otra de ellas es mediante la comunicación entre los dispositivos y calculando el tiempo de vuelo de la información al igual que en tecnologías como la de RFID, solo que en este caso las distancias a detectar podrán ser mucho mayores.

Figura 10 : Localización mediante balizas y WiFi

7. Bluetooth: El funcionamiento de la tecnología Bluetooth para la localización de elementos en espacios interiores es idéntico al de la tecnología WiFi, se tienen los métodos de localización por RSSI o por tiempo de vuelo de la transmisión de datos.

   La diferencia entre ambos es el ancho de banda por radiofrecuencia que utiliza cada uno, en el caso del Bluetooth es de 2.4GHz y en el caso del WiFi es de 5.0GHz, y por tanto las distancias de trabajo son menores en el caso de la tecnología Bluetooth porque la potencia de emisión de señal es mayor en el caso de la tecnología WiFi.

8. Infrarrojos: El fundamento de esta tecnología para el cálculo de distancias es el mismo que en el caso de os sensores de ultrasonidos solo que en este caso la señal emitida es de luz infrarroja en lugar de sonido.

   La distancia se obtiene a partir del tiempo de ida y vuelta de la señal solo que en este caso se presentan varios problemas con el tema de la iluminación, la luz infrarroja es la menos afectada por la luz ambiental, pero si las distancias d calculo son elevadas, la luz del emisor se verá muy debilitada, por tanto las distancias de trabajo son bastante limitadas con este tipo de tecnología.

   De igual forma ocurre con la dirección en a que se encuentre el objeto, si se desea medir una distancia concreta, se convierte en un sistema unidireccional y eso puede presentar un problema a la hora de localizar objetos.

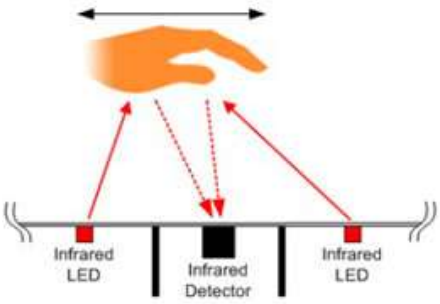

Figura 11 : Obtención de distancias y posición con sensores de infrarrojos

5 Tecnologías empleadas

5.1 Estudio de la comunicación Bluetooth

5.1.1 Comunicaciones inalámbricas

Las comunicaciones inalámbricas se basan en la transmisión de datos, como el propio nombre indica, sin cables, sin la necesidad de un medio físico que enlace a los dos interlocutores, emisor y receptor.

El hardware únicamente es empleado en los sistemas propios del dispositivo de comunicación, como las antenas o los drivers de comunicación, del resto de la comunicación se encarga de gestionarla el software y los protocolos de comunicación.

Por lo general, las tecnologías de comunicación inalámbrica se realizan con señales de radiofrecuencia de bajo consumo y las cuales reservan un ancho de banda determinado para realizar sus transmisiones.

Este tipo de tecnología presenta problemas como la velocidad, dado que las ondas de radio no se desplazan de misma forma por el ambiente que la corriente por los cables y problemas como la seguridad, dado que estas ondas son susceptibles de ser captadas por otro dispositivo que no sea el receptor objetivo.

En la Tabla 1 se pueden observar varias tecnologías de comunicación inalámbricas en función de su potencia según los estándares del IEEE (*Institute of Electrical and Electronics Engineer*).

Tipo de red	WWAN (Wide)	WMAN (Metropolitan)	**WLAN (Local)**	WPAN (Personal)
Estándar	GSM/GPRS/ UMTS	IEEE 802.16	IEEE 802.11	IEEE 802.15
Tecnología		WiMAX	WiFi	Bluetooth
Velocidad	9,6 Kbps hasta 100 Mbps (4G)	15-134 Mbps	54-300 Mbps	Menos de 1 Mbps
Frecuencia	0,9/1,8/2,1 GHz	266 GHz	2,4 y 5 GHz Infrarrojos	2,4 GHz
Rango	35 Km	1 – 50 Km	30-150 m	10 m

Tabla 1 : Características de las comunicaciones inalámbricas

Dentro de estas tecnologías de comunicación inalámbricas se pueden encontrar a gran escala como las comunicaciones móviles GSM, comunicaciones de área metropolitana con distancias de transmisión intermedias como la tecnología WiMAX aun en desarrollo.

Por último se tienen las tecnologías de comunicación más populares como son el WiFi o Bluetooth con distancias de transmisión inferiores a las anteriores ya que están diseñadas para redes de área local o de área personal, como es el caso del Bluetooth.

Comúnmente el nivel físico utilizado es el definido por el IEEE 802.11 Radio DSSS (*Direct Sequence Spread Spectrum*), espectro ensanchado por secuencia directa, que modula la señal con una especie de ruido añadido a la trama del mensaje, de tal forma que, únicamente el receptor objetivo sea el que traduzca esta modulación si se ha configurado según el mismo protocolo.

Las redes inalámbricas pueden presentar problemas de interferencias externas por solapamiento de otras señales que trabajes en un ancho de banda similar o muy próximo como es el caso de las señales microondas (2.49GHz) y las señales Bluetooth (2.4GHz).

La arquitectura de los niveles inferiores de los nodos de una red inalámbrica se rige por un modelo como el mostrado en la Figura 12.

SUBCAPA LLC (CONTROL DE ENLACE)
SUBCAPA MAC (CONTROL DE ACCESO AL MEDIO)
INTERCAPA ENLACE-FISICO
CAPA DEPENDIENTE DEL MEDIO FISICO (DSSS)

Figura 12 : Niveles inferiores de la arquitectura de un sistema inalámbrico

La arquitectura de las redes inalámbricas está basada en puntos de acceso entre la red inalámbrica, o conjunto de servicio básico, y la red cableada de comunicación o sistema de distribución, todo esto se refleja en la Figura 13.

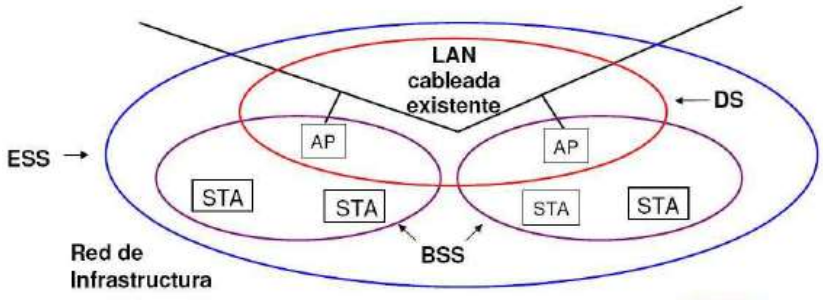

Figura 13 : Arquitectura de redes inalámbricas basadas en puntos de acceso

Esta configuración es la más usual en casos como las tecnologías WiFi, donde se tiene un punto de acceso conectado a la red por medio de cableado como puede ser la fibra óptica, y una serie de estaciones consumidoras de servicios de comunicación que utilizan como intermediario al punto de acceso.

También se puede tener una configuración de red *AdHoc* donde cada estación o nodo se comunica individualmente con otro sin necesidad de un concentrador o punto de acceso.

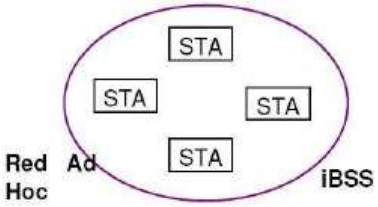

Figura 14 : Configuración de red *adHoc*

Esta configuración es típica para las comunicaciones Bluetooth o por datos móviles.

5.1.2 Comunicación Bluetooth

Bluetooth es una tecnología de comunicación inalámbrica que utiliza la radiofrecuencia en la banda de 2.4GHz como medio de transmisión de datos y con la tecnología de comunicación FHSS (*Frecuency Hop Spread Spectrum*) o espectro esparcido por salto de frecuencia, que quiere decir que el ancho de banda reservado para la comunicación, en este caso de 2.4 a 2.48 GHz, se subdividirá en varios canales para evitar confusiones o solapamientos entre distintas conversaciones y en el desarrollo de la misma se saltara de canal en canal de forma aleatoria, a una velocidad de 1600 hops/s.

La comunicación se realiza punto a punto sin necesidad de un punto de acceso intermedio con la inconveniencia de que, se necesita de una distancia máxima de 10m, pudiendo llegar a 100m dependiendo del dispositivo, para poder establecer conexión entre dispositivos.

Clase 1: 100m
Clase 2: 10m
Clase 3: 1m

La velocidad de transmisión no supera 1Mbps por tanto no es recomendable para la transferir grandes volúmenes de información.

Cada paquete como el anterior tiene un tiempo máximo de transmisión cercano a los 625us, durante el cual debe enviar el mensaje antes de que se produzca un salto de frecuencia, de este modo la comunicación temporalmente se subdivide en marcos de 625us para la transmisión y de esta forma poder controlar la comunicación.

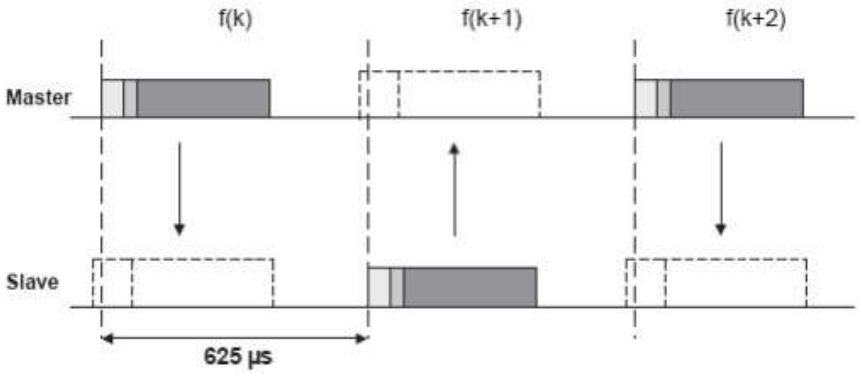

Figura 15 : Diagrama temporal de transferencia de mensajes maestro-esclavo

El consumo energético de la transmisión es de en torno a 1 milivatio, por tanto es un buen método de solución para pequeñas comunicaciones.

Es posible conectar 8 dispositivos Bluetooth simultáneamente creando una red de área personal o del inglés *Personal Area Network* (PAN) siempre asignándose una como maestro y las demás como esclavos, la configuración de la red será *adHoc*, y por tanto no se necesitara de punto de acceso, y dos o más adHoc se podrán enlazar por medio de uno de los esclavos que hará de puente entre ambas.

En cuanto a la mensajería, solo es posible transmitir paquetes de datos de hasta 2700 bits de los cuales, los 72 primeros de código de acceso que contienen información del dispositivo que ejercerá de maestro o coordinador en la conversación y deben de ser idénticos a lo largo de la comunicación, los 54 siguientes se corresponden con una cabecera contenedora de información sobre el protocolo de comunicación, puerto de acceso, dirección de acceso al medio MAC, información del paquete como la longitud o caracteres de control y protocolo de errores o confirmaciones y reconocimientos.

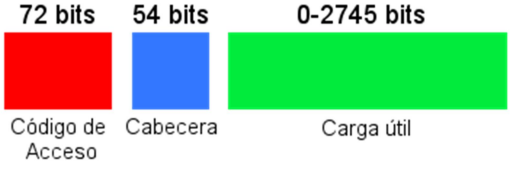

Figura 16 : Subdivisión de los paquetes de comunicación Bluetooth

La comunicación Bluetooth es de tipo síncrona, lo que quiere decir que, el receptor del mensaje identifica cada uno de los bits recibidos, conoce el principio y el final de cada uno de los paquetes que se le envían y la estructura de los mismos, esta información de sincronismo se carga en el espacio de código de acceso de cada mensaje mencionada anteriormente en formato de caracteres de control.

En la Figura 17 se puede observar con mejor detalle la estructura de la comunicación Bluetooth.

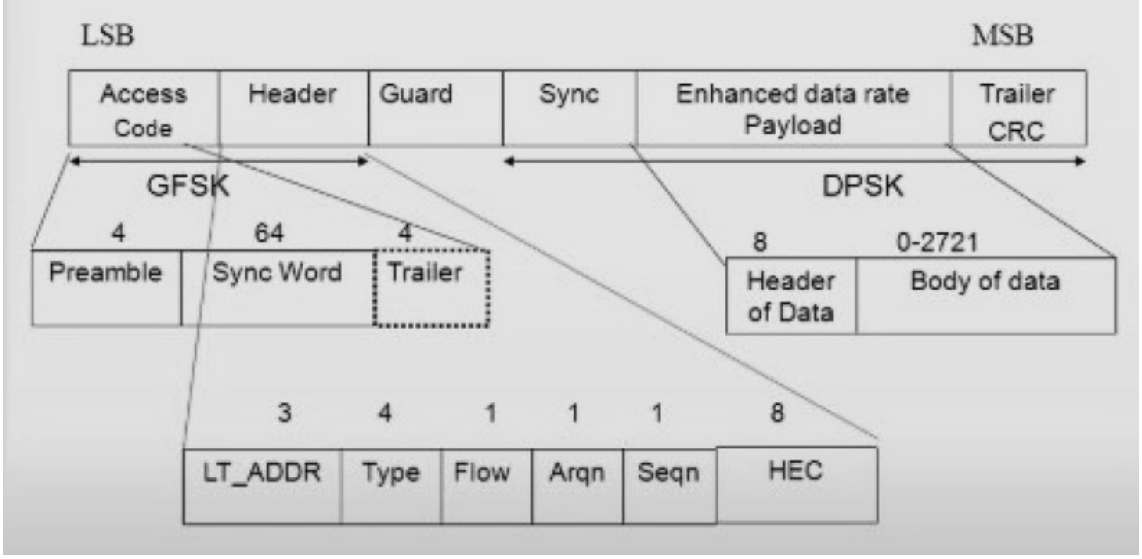

Figura 17 : Estructura de cada paquete en comunicación Bluetooth síncrona

El código de acceso se corresponde con los primeros 72 bits de la trama, de los cuales solo 64 contienen información útil generada por el maestro y que contiene caracteres de sincronización, compensación y ajuste. Este campo también es utilizado para los métodos de acceso al medio de forma aleatoria donde son probables las colisiones y de esta forma indicar al medio que el dispositivo está preparado para escuchar y escribir.

Los primeros 4 bits del código de acceso forman el *Preámbulo* de compensación de sincronía, de esta forma podrán ser 1010 o 0101 en función del siguiente mensaje a transmitir.

Los siguientes 64 bits forman la palabra de sincronía, la cual varía en función del tipo de control de acceso establecido, pudiendo ser Código de Acceso a Canal, Código de Acceso a Dispositivo o Código de Acceso a Información.

Los últimos 4 bits del campo de control de acceso forman el *Tráiler*, que define la transición o compensación entre el control de acceso y la cabecera del mensaje con las mismas configuraciones binarias que en el caso del preámbulo.

Seguido del campo de control de acceso se tiene la cabecera del mensaje con información sobre el mensaje más que sobre la comunicación como era el caso del campo de mensaje anterior.

De los 54 bits que componen la cabecera se tienen 18 principales:

- Los 3 primeros componen la *AM-ADDR*, que es la dirección del miembro para distinguirlo dentro de la red como maestro o esclavo.

- Los siguientes 4 bits forman la información sobre el tipo de mensaje que se va a enviar dependiendo del tipo de enlace *TYPE*, si este es de sincronización de conexión orientada (SCO) o es de tipo enlace asíncrono de baja conexión (ACL).

- El siguiente bit, *FLOW*, es el encargado del control de flujo en la transmisión con enlace asíncrono ACL, este bit se podrá a 0 para detener la transmisión en un posible caso de buffer de lectura saturado, este bit se podrá restaurar a 1 en caso de espera a lectura o buffer RX vacío.

- El bit que le sigue es el encargado de los reconocimientos de mensaje, *ARQN*, este se insta a 1 en caso de reconocimiento de mensaje o *ACK*, en caso de que el mensaje no se haya recibido correctamente en el destino este tomara como valor 0 que indica el no reconocimiento o *NAK*.

- El siguiente bit de cabecera se corresponde con el de sincronización de paquetes, *SEQN*, en caso de que uno de los paquetes del mensaje total haya tenido un problema en el reconocimiento y tenga que ser devuelto, este bit controla que no se almacenen ambos paquetes (el no reconocido y el nuevo) en la secuencia del mensaje.

- Por último se tiene un byte destinado a la comprobación de errores en el encabezado, *HEC*, este byte es codificado por un método de cálculo interno y ha de ser idéntico en el regreso, en caso de no ser así se desecharía el paquete entero.

Por último el campo de datos contiene otra cabecera de 8 bits de datos que detalla la información de los datos contenidos en el paquete concreto, en el caso de la cabecera de paquete la información era sobre el mensaje conjunto de paquetes.

El resto del paquete se corresponde con los datos de información para el host que no tienen por qué importar al nivel de control de enlace y mensajes.

5.1.3 Intensidad de la señal RSSI y distancia

Hablando de los dispositivos móviles y de las nuevas tecnologías de comunicación inalámbricas, una de las características más representativa es la potencia de las señales de los transmisores.

En el caso de las comunicaciones inalámbricas, se usa el término RSSI o indicador de la fuerza de la señal recibida, a la potencia de las señales. Esta magnitud es adimensional y se podría considerar como una ganancia en el sentido de decibelios (dB).

El rango de valores de RSSI oscila normalmente entre 0 y aproximadamente -120dB, siendo 0 el valor ideal para establecer una conexión e indica la potencia máxima, cuanto más negativo sea el valor de esta intensidad de señal, más débil será la misma y por tanto la comunicación no será igual de fiable.

En lo que respecta a estos valores, se pueden definir diferentes estados de la comunicación en función de los mismos, de tal forma que:

- 0dB Potencia ideal.
- [-40, -60] dB Potencia muy buena y estable.
- [-60, -80] dB Potencia débil pero todavía estable.
- < -80 Potencia muy débil, corre riesgo de no poder establecer comunicación.

Una vez conocida la intensidad de la señal del transmisor, mencionar que esta varía, principalmente, en función de la distancia a la que se encuentren los dispositivos involucrados.

De tal forma que cuanto más cera estén uno de otro, la potencia de la señal será mucho mayor, y cuanto más lejos se encuentren más débil y negativa será la intensidad de la señal.

El cálculo de la potencia de la señal, según estudios experimentales, puede definirse según la expresión (1).

$$\boldsymbol{RSSI_1 = RSSI_0 + 10nlog_{10}\left(\frac{d_1}{d_0}\right)} \qquad (1)$$

De la misma ecuación se puede obtener la distancia a raíz de las intensidades de la señal recibida, siendo el punto 0 una distancia conocida con su distancia conocida.

$$\boldsymbol{d_1 = 10\, e^{-\left(\frac{RSSI_1 - RSSI_0}{10n}\right)}} \qquad (2)$$

La única incógnita, en una situación real, sería el parámetro '*n*' que se corresponde con una constante dependiente del medio de propagación de la señal, en el vacío este parámetro se toma con el valor 2, en otro medio de propagación como el aire puede encontrarse en torno a 2,5.

5.1.4 Nuevas tecnologías Bluetooth

Desde la aparición del Bluetooth en 1994, esta tecnología ha seguido avanzando ya que era y sigue siendo una tecnología de comunicación inalámbrica de las más utilizadas en el ámbito personal, como se indica según su estándar IEEE 802.11 WPAN (*Wide Personal Area Network*), en todos sus aspectos para optimizar al máximo sus características, de tal forma que se ha conseguido disminuir el consumo hasta su última versión denominada *Bluetooth Low Energy*.

Esta evolución, Bluetooth 4.0, se desarrolló con el objetivo de llevar más lejos la comunicación Bluetooth aparte de en dispositivos de consumo relativamente alto como pueden ser los ordenadores o los teléfonos móviles actuales. El hecho de disminuir el consumo de los sistemas Bluetooth presenta ventajas de cara a los *wereables* modernos como los relojes o los dispositivos de audio portátil, ya que generalmente necesitan continuamente de los servicios de la comunicación Bluetooth.

La idea de localizar dispositivos en un espacio interior mediante el uso de Bluetooth resulta más favorable con la aparición de estos nuevos sistemas de bajo consumo, ya que se pueden disponer un mayor número de puntos de detección y tenerlos constantemente buscando dispositivos Bluetooth.

5.2 Android

5.2.1 Introducción

Sistema operativo creado en 2006 por Android Inc., la cual fue comprada posteriormente por Google en 2007.

Android es un uno de los sistemas operativos para dispositivos móviles más usados en el mundo, con más de mil millones de dispositivos que lo incorporan, entre los tres principales que son iOS de la marca Apple y Windows Mobile de Microsoft.

El sistema operativo Android desarrollado por Google está basado en Linux, esto es un punto positivo para el desarrollo en este entorno dado que es de carácter libre, personalizable y fácil de utilizar. La seguridad también es uno de los puntos fuertes, dado que las actualizaciones en este ámbito son más periódicas y aumentan con el paso del tiempo.

El sistema operativo del robot presenta ventajas de cara a los desarrolladores de sus aplicaciones ya que se trabaja con lenguajes de programación comunes y más usuales hoy día como puede ser Java o C.

En el tema de la conectividad, aunque Android pueda presentar una pequeña falta de seguridad, permite total o parcialmente acceso a su sistema y de esta forma aumentar la velocidad del proceso de transferencia de archivos o información.

5.2.2 Arquitectura

Aplicaciones: El sistema operativo Android integra una serie de aplicaciones base de relativa necesidad como pueden ser el servicio de llamadas, correo, buscador o navegador, y la aplicación home o directorio principal.

Marco de aplicaciones o APIs: gestión de herramientas internas como la localización, paquetes, notificaciones, gestión de ventanas y actividades. Herramientas que facilitan el trabajo a la programación de bajo nivel en Android. Lenguaje de programación Java.

Bibliotecas: herramientas software de alto nivel base en el sistema operativo, se utilizan para gestionar los gráficos, bases de datos, también pueden ser utilizadas para el desarrollo de aplicaciones. Lenguaje de programación en C / C++.

Sistema de tiempo real: Máquina virtual interna en Android sobre la cual se ejecutan las aplicaciones.

Núcleo Linux: Software interno en Android, destinado sobre todo a la gestión de procesos y de memoria.

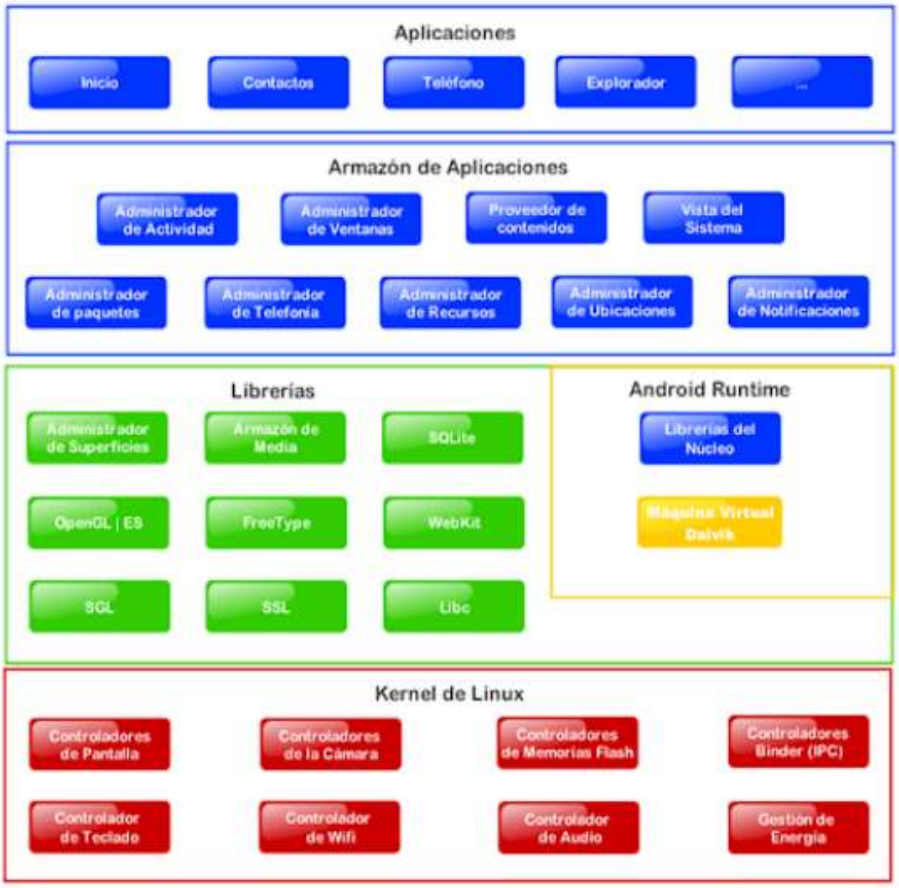

Figura 18 : Estructura software interna de Android

5.2.3 Versiones

Como todo programa, Android y sus APIs no hacen una excepción y van renovando sobre versiones a medida que la tecnología avanza.

El hecho de crear nuevas versiones, ya no solo trata de evolucionar el sistema operativo sino que presenta un impulso comercial a modo de marketing, ya que las cosas viejas como una versión, pierden atractivo con el tiempo y de ahí la necesidad de actualizarse.

En el caso de Android, al igual que otro SO o aplicación las versiones vienen con correcciones de defectos apreciados y con mejoras o inserciones de nuevas tecnologías.

Las nuevas versiones, por lo general vienen cargadas de nuevas APIs o librerías de gran utilidad para los desarrolladores de aplicaciones, de tal forma que aumenta el atractivo de cara al consumidor y al desarrollador ya que puede llamar más la atención el hecho de desarrollar una aplicación que maneje utilidades de sensores como puede ser un acelerómetro antes que una se solo permita implementar un par de botones para jugar.

A continuación se presenta una breve descripción de las versiones más vistosas de Android con sus mejoras y correcciones más determinantes.

Cupcake Android 1.5 (API 3)

Primera versión oficial de Android ya obsoleta. En su aparición ofrecía el teclado por pantalla en los primeros dispositivos móviles táctiles y predicción de texto en el mismo.
Otras utilidades como grabación de audio y video, y conexión a Bluetooth de dispositivos audiovisuales. También incorpora los primeros *widgets* o accesos rápidos de Android.

Donut Android 1.6 (API 4)

Mejoras para realización de búsquedas dentro del dispositivo Android (contactos, música, imágenes…).
Incorpora la primera aparición de síntesis de audio para transformación en texto.
Mejoras en la resolución y nuevos elementos y accesos gráficos para la programación de aplicaciones.

Eclair Android 2.0 (API 5)

Esta versión de Android pasó desapercibida de cara a los mayores consumidores de este sistema operativo dado que las innovaciones fueron insuficientes como la incorporación de una API para gestionar el nuevo Bluetooth 2.1 que permitía conectarse a cualquier soporte Bluetooth comercial y la mejora del manejo de cuentas de google.

Froyo Android 2.2 (API 8)

Como característica más destacada se puede indicar la mejoría de velocidad de ejecución de las aplicaciones, que se logró con la mejora del compilador de la máquina virtual de Android, Dalvik.
Nuevas mejoras en los navegadores con motores JavaScript de Chrome o Adobe Flash.
De cara al desarrollo de aplicaciones en Android, se mejoran los mensajes de interfaz o ventanas emergentes dentro de aplicaciones, también se incorporan las actualizaciones

automáticas de las aplicaciones cuando está disponible una nueva versión de la misma, se comienzan a ofrecer servidores de guardado de información dependiente de la aplicación para, de esta forma no perder la información en caso de problemas.
Nueva mejora de reconocimiento de voz.
En el aspecto de la conectividad, los terminales pueden compartir su acceso a internet, ya sea por medio de USB o por WiFi.
Aparición de notificaciones emergentes desde la ventana principal y de acceso rápido.
Modificaciones en la luminosidad de la pantalla para ajustar según el ambiente.

Gingerbread Android 2.3 (API 9)

Con el aumento de la popularidad de Android en la nuevas Tabletas, se incorpora mejoras gráficas para la adaptación de pantallas a mayores dimensiones.
Se añaden funcionalidades propias de las pantallas multitáctiles como las opciones de cortar, pegar o copiar en función de la presión o el número de tics sobre la pantalla.
Se incluye soporte para varias cámaras, pensado en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha propiciado la inclusión de reconocimiento facial para identificar el usuario del terminal.
Aumento de la capacidad de respuesta en juegos y aplicaciones similares.
Se dispone de mayor apoyo para el desarrollo de código nativo NDK (*Native Develop* Kit)
También se mejora la gestión de energía y control de aplicaciones.
Inserción de la tecnología NFC (*Near Field Communication*) para comunicaciones inalámbricas de forma dual.
Mejora del soporte nativo para adaptar más sensores (como giroscopios y barómetros) y un gestor de descargas para las descargas más voluminosas.

Honeycomb Android 3.0 (API 11)

Disponible barra de acciones y acceso a ajustes de aplicaciones para desarrolladores.
Teclas físicas totalmente reemplazadas por botones táctiles en la pantalla.
Mejoras en las notificaciones y en las acciones de interfaz en lo referente a modificación de iconos y archivos, como arrastrar y soltar o mantener pulsado y desplegar opciones.
Mejora de gráficos 2D y 3D, así como el motor de imágenes y video que mejora la calidad y los fps de las pantallas.
Primera versión de la plataforma que soporta procesadores multinúcleo. La máquina virtual Dalvik ha sido optimizada para permitir multiprocesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.
En esta versión se añaden nuevas alternativas de conectividad, como las nuevas APIS de Bluetooth. También, se permite conectar teclados completos por USB o Bluetooth.
Mejora de los dispositivos en el ámbito empresarial, mejoras en la protección y encriptación, control de usuarios y caducidad de contraseñas.

Ice cream Sandwich Android 4.0 (API 14)

Nueva API de reconocimiento facial que permite desbloquear el teléfono.
Mejora en el reconocimiento de voz en el aspecto de que no es necesaria la conexión previa al servidor de búsqueda para comenzar la grabación de voz.
Mejora en la gestión del tráfico de datos para poder establecer límites de operación y evitar de esta forma gastos en la factura, también dispone de gráficos que representan el consumo de datos móviles.
Incorporación de editor de imágenes en el momento de la toma de la foto sin necesidad de esperar a que esta ya este tomada.
Se mejora la API para comunicaciones por NFC y la integración con redes sociales.

Jelly Bean Android 4.1 (API 16)

Mejoras en el rendimiento del terminal, mejoras sobre todo en el refresco de pantalla y la velocidad de actuación del procesador cada vez que se activa un punto de la pantalla capacitiva. Aumento del buffer de imágenes para evitar saturaciones en pantalla y ejecución de refresco de forma vertical descendente.
Mejora en las notificaciones de la ventana deslizable y de los iconos de escritorio, de esta forma se puede ajustar el tamaño de los mismos y personalizar la ubicación del mismo.
Salto notable en el reconocimiento de voz que incluye una nueva librería que permite la no conexión necesaria a internet.
Mejora en la instalación de aplicaciones, mejora la descarga e instalación en segundo plano.

Jelly Bean Android 4.2 (API 17)

En esta ocasión se decidió no cambiar de nombre a la versión más que a la API.
Una de las novedades más importantes es que podemos crear varias cuentas de usuario en el mismo dispositivo. Aunque, esta característica solo está disponible en tabletas. Cada cuenta tendrá sus propias aplicaciones y configuración.
Mejora en la gestión de cuentas y usuarios, en esta versión es posible crear más de un usuario en el mismo terminal, en principio solo en tablets, y navegar entre ellos de forma independiente.
Mejora en la conectividad para dispositivos de reproducción como televisores, ahora es posible conexión mediante WiFi con la tecnología denominada Miracast.
Mejoras en la cámara de tal forma que se pueden realizar capturas a 360° para crear panorámicas.

Jelly Bean Android 4.3 (API 18)

Se añade el control de aplicaciones por usuarios, de esta forma solo determinados usuarios con permiso de acceso pueden abrir determinadas aplicaciones. Útil para el control parental y la privacidad como el ejemplo de las redes sociales. Los programadores también pueden definir restricciones en las apps, que los propietarios puedan activar si quieren.

Se da soporte para *Bluetooth Low Energy* (BLE) que permite a los dispositivos Android comunicarse con los periféricos con bajo consumo de energía.
Se agregan nuevas características para la codificación, transmisión y multiplexación de datos multimedia.

KitKat Android 4.4 (API 19)

Creación de nueva api que permite el ajuste automático de aplicaciones para dispositivos con poca capacidad de memoria RAM.
Mejora notable con la implementación del modo pantalla completa que elimina temporalmente las herramientas de interfaz como barras de inicio y menú de usuario, hasta el momento en que se pasa cerca de los bordes de la pantalla cuando se vuelven a habilitar los comandos.
Se mejora la conectividad con soporte de NFC, varios protocolos sobre Bluetooth y soporte para mandos infrarrojos.
Mejora en la eficiencia energética de los sensores e incorporación de nuevo sensor medidor de pasos.
Servicio de acceso de aplicaciones a la nube para almacenar y gestionar la información de los usuarios por cuentas y perfiles en cada aplicación.
Acceso a la nube para gestión general de archivos del usuario del terminal con plataformas como Google Drive.
Implementación de nueva máquina virtual ART en formato beta para desarrolladores dado que su rendimiento apunta a una notable mejora frente a Dalvik pero de momento no se implementa de forma activa.

Lolly pop Android 5.0 (API 21)

La novedad más importante de Lollipop es la extensión de Android a nuevas plataformas, incluyendo Android Wear, Android TV y Android Auto.
Implementación parcial en aplicaciones de la máquina virtual ART en sustitución de Dalvik ya que, como se probó en la versión anterior poseía notables mejoras.
En lo referente a la eficiencia energética, se incorpora el modo automático en la aplicación de ahorro de batería que inhabilita una serie de complementos del terminar incluso llegando a quedarse únicamente con modos de llamada y mensajería.
Se implementa nueva api para restringir el uso de determinados servicios con condiciones concretas como cuando esté conectado a la corriente o cuando el sensor de presencia este activo.
Nueva configuración en el análisis de consumo de batería de las aplicaciones individualmente.
Nuevos sensores como la incorporación al giroscopio para umbrales de inclinación en diversas aplicaciones, sensor de pulso cardíaco, o detectores de gestos desarrollados por Google.
Funcionalidad de bloqueo de aplicaciones en función de la situación según preferencias del usuario.

Mashmallow Android 6.0 (API 23)

Mejora en la administración y seguridad de datos, los archivos y datos de usuario se sincronizan con la nube y se pueden actualizar de forma automática quitándole responsabilidad al usuario.
También incorporar la función de recuperar los datos almacenados en la nube en lugar de formatear el dispositivo de fábrica o en la adquisición de un nuevo dispositivo por parte del usuario.
Asistente de voz No won Tap, variante de Google Now, mas integrado en algunas aplicaciones que permitan su uso. Las aplicaciones activas pueden hacer consciente al asistente de la actividad y de esta forma hacer más intuitiva la predicción.
En esta misma línea, se añade un API que permite interacciones basadas en voz. Es decir, si nuestra aplicación ha sido lanzada por voz, podremos solicitar una confirmación de voz del usuario, seleccionar de una lista de opciones o cualquier información que necesite.
Mejora en los modos de autentificación en las aplicaciones de tal forma que se incorpora la huella digital para el permiso de acceso, como aplicaciones bancarias, de esta forma el usuario no tiene por qué recordar todas las contraseñas en caso de que tenga varias en varias aplicaciones.
Se puede utilizar un dispositivo de almacenamiento externo, para que sea usado como almacenamiento interno. Podemos fragmentar, formatear y encriptar una tarjeta SD para ser usada como memoria interna. También podemos montar y extraer lápices de memoria USB de forma nativa.
Se da soporte de forma nativa a pantallas 4K, lápices Bluetooth, múltiples tarjetas SIM y linterna.

Nougat Android 7.0 (API 24)

De pantallas múltiples de tal forma que, el usuario, puede abrir un número concreto de aplicaciones simultáneamente.
Mezcla de métodos de compilación entre compilador en tiempo que compila en el momento que se lanza la solicitud de ejecución y el compilador de predicción que compila con anterioridad a la petición de ejecución, el principal impacto de esta técnica se nota en la instalación de las aplicaciones y actualizaciones del sistema.
El usuario va a poder activar el modo de ahorro de datos cuando se encuentre en itinerancia o cuando esté a punto de agotar un paquete de datos. En este caso, tanto el sistema como las aplicaciones han de tratar de minimizar al máximo las transferencias de datos.

Oreo Android 8.0 (API 26)

Gestión de aplicaciones seguras, de esta manera se puede dar opción a escaneo o no de malware presente en apps o apk.
Se fija el límite de procesos en segundo plano para evitar un alto consumo de batería. Introducción de capa interfaz entre librerías framework y núcleo Linux para control de acceso a la misma dentro de la estructura, esto evitara acceso directo de actualizaciones al núcleo.

Modificaciones en las notificaciones de sistema, se ordenan por importancia y por prioridad fijada por el usuario, fondos personalizables y accesos directos a notificaciones de aplicaciones.
Uso de aplicación de video ahora es posible en segundo plano mientras se ejecutan otras aplicaciones de forma paralela.

5.2.4 Conceptos y herramientas

API

Interfaz de programación de aplicaciones. Se podría decir que una API es una herramienta o una facilidad que se le puede ofrecer al programador de aplicaciones, se encarga de todos los protocolos internos y las acciones de bajo nivel que pueden presentar una tarea engorrosa para el desarrollador de aplicaciones.

Las APIs se encuentran a bajo nivel dentro del nivel estructural marco de aplicaciones o framework y es el más importante para los desarrolladores. Existen diferentes tipos de APIs dependiendo de su finalidad, para entenderlo mejor se puede hablar de la similitud entre APIs y Bibliotecas de programación. De esta forma las APIs proveen de recursos que facilitan la programación más engorrosa como podría ser el caso de escritura de un archivo, estas APIs dispondrán de una herramienta directa que permita al desarrollador escribir una cadena sin necesidad de programar el buffer de salida.

Play Store

Se trata de la aplicación destinada a la búsqueda, descarga e instalación de aplicaciones desarrolladas externas al terminal, existen aplicaciones ya integradas de fábrica en el dispositivo como pueden ser la aplicación de contactos o las aplicaciones de Google, y por otro lugar están las aplicaciones desarrolladas por otras entidades ajenas a Google la cuales se deben descargar en la aplicación de Play Store.

El contenido del portal de descarga de Play Store esta depurado y libre malware, por tanto todas la apps disponibles en esta plataforma son de uso comercial.

APK

Toda app tiene su apk o kit de aplicación, los archivos APK contienen todo el programa desarrollado por los programadores, tanto su programación funcional como su programación gráfica.

Ya conociendo lo que es Play Store, en esta plataforma se encuentran las aplicaciones a las que cualquier usuario con un mínimo conocimiento del sistema operativo de Android puede acceder, por otro lado, existen aplicaciones que, por diversos motivos no han podido acceder

al mercado, ya sea de forma libre o con coste, y están disponibles en internet y pueden ser instaladas en cualquier dispositivo siempre y cuando tengan el permiso del usuario.

Los archivos APK se generan cuando de crea y se compila un nuevo proyecto de aplicación Android.

Eclipse

Entorno de trabajo. Eclipse es el programa que se usa como interfaz de desarrollo de aplicaciones en cualquier lenguaje de programación, según desee el usuario.

En el caso de la programación de aplicaciones Android y Java, es el entorno más usado debido a su estabilidad en el aspecto de ejecución, compilación y depuración.

Es un entorno de carácter intuitivo y con todo tipo de herramientas que facilitan la tarea del desarrollador, de este mismo modo, el editor permite instalar una serie de plugins que se pueden añadir como en el caso de los paquetes de desarrollo Android o incluso podría llegar a ser los de programación en iOS.

Android Studio

Entorno de desarrollo, principalmente desarrollado ara aplicaciones Android, en este caso el editor no presenta buenas estabilidades en lo referente a depuración y compilación.

Este entorno de desarrollo se ha configurado para programación en Android, lo que quiere decir que, no es posible la programación en otro tipo de lenguaje que no sea Java-Android.

Ya dispone integralmente de herramientas de emulación de aplicaciones y de gestor de paquetes de Android para cargar e instalar las APIs.

***Java Develop Kit* (JDK)**

Kit de desarrollo Java. Paquete instalador que provee de las herramientas necesarias para poder trabajar con el lenguaje Java, dentro de entornos de programación y desarrollo como puede ser el caso de interfaces como Eclipse o similares.

***Android Software Develop Kit* (SDK)**

Kit de desarrollo software de Android. Paquete instalador de herramientas necesarias para poder trabajar con utilidades de Android y de las Apis necesarias.

En el caso de IDEs como Android Studio no es necesario instalar este kit de desarrollo ya que viene integrado, pero en el caso de Eclipse sí que será necesario y más que útil dado que

añadirá al compilador las bibliotecas de Android y complementos, también añadirá como herramientas el gestor de paquetes de Android y su emulador.

Android SDK Manager

Plataforma que se incluye en el Android SDK, la cual permite al desarrollador gestionar los paquetes de desarrollo Android, de esta forma puede instalar las APIs que desee para adecuar su proyecto a diferentes versiones de Android, también se pueden actualizar y eliminar dichos paquetes instaladores.

***Android Virtual Device Manager* (AVD)**

El dispositivo virtual de Android sirve para emular el comportamiento de un terminal para poder ejecutar sobre el los proyectos que se estén desarrollando en Android.

Esta herramienta permite emular cualquier tipo de dispositivo y versión Android según le plazca al desarrollador.

6 Desarrollo del proyecto

6.1 Selección de IDE de programación y lenguaje

La idea inicial del proyecto era llevar a cabo la programación en lenguaje C/C++ debido a la comodidad y la anterior experiencia en la programación de dicho lenguaje.

Tras informarse acerca de los diversos lenguajes de programación para la realización de programas y aplicaciones Android, se llegó a la conclusión de que la programación con lenguaje java es más sencilla y hay más información disponible que en el caso del lenguaje C++.

El lenguaje C++ es más avanzado para estas utilidades ya que se puede acceder a funciones directas del procesador que se encuentran dentro de la arquitectura en el segundo nivel de Bibliotecas y el hardware del dispositivo, así como el lenguaje java se ejecuta sobre una interfaz o una máquina virtual desde el nivel de marco de aplicaciones.

Con la programación en el nivel de marco de aplicación la tarea de programar se hace menos pesada dado que no es necesario indagar ni empezar de cero con las utilidades del dispositivo, permite que se pueda programar con la ayuda de las APIs de Android y sobre ellas trabajar con objetos de forma más cómoda para el desarrollador.

Por tanto, la programación en java es más recomendada para principiantes en la materia, y el entorno de desarrollo se deberá acondicionar para la programación en lenguaje java.

Pasos para la instalación del entorno de desarrollo:

1. Instalación del kit de desarrollo Java o máquina virtual sobre la que se ejecutaran los programas a desarrollar. Archivos de instalación disponibles en la página oficial de Oracle.

2. Instalación del IDE de Eclipse para programación java. Archivos de instalación disponibles en la página oficial de Eclipse.

En caso de optar por la programación en C/C++:

1. Instalación del kit de desarrollo java o máquina virtual sobre la que se ejecutaran los programas a desarrollar. Archivos de instalación disponibles en la página oficial de Oracle.

2. Instalación en memoria interna del compilador MinGW.

3. Instalación del IDE de Eclipse para programación C/C++. Archivos de instalación disponibles en la página oficial de Eclipse.

6.2 Instalación de herramientas de desarrollo

Lo primero que hay que hacer es instalar la máquina virtual de Java para poder compilar y ejecutar cualquier tipo de programa, ya sea en Java nativo o Android-Java, que se encuentra disponible en la página oficial de Oracle.

Figura 19 : Punto de acceso a la descarga del Kit de Desarrollo Java

Una vez esté disponible el archivo descargado simplemente bastara con ejecutarlo y seguir los pasos que marca el instalador para disponer del kit de desarrollo Java y de su máquina virtual.

Posteriormente se necesitara el IDE de Eclipse como entorno para programar y editar el código de la aplicación. Para ello se accede a la página oficial de Eclipse y se descarga la versión que desee el desarrollador, en principio cualquiera de ellas bastara si se trata de desarrollar en Java, puesto que la mayoría viene con paquetes configurados por defecto aunque sea posible instalar posteriormente para cualquier otro tipo de lenguaje como pueda ser HTML5.

Figura 20 : Punto de acceso a la descarga del entorno de desarrollo Eclipse

También es posible descargar el IDE de Eclipse Galileo apto para trabajar con desarrollo en C/C++, la configuración es idéntica.

En este caso al igual que con el paquete de desarrollo de Java bastara con seguir los pasos que indica el instalador.

6.3 Preparación del entorno para trabajar en Android

Una vez se dispone de las herramientas básicas para poder programar aplicaciones en lenguaje Java se necesita de los paquetes de herramientas que permitan poder trabajar con las bibliotecas y APIs de Android en este caso para poder empezar a desarrollar aplicaciones en lenguaje Android-Java.

Para poder habilitar la programación con Android y disponer de las bibliotecas propias en el entorno de trabajo de Eclipse se necesita de un plugin, denominado Android SDK, instalable desde el mismo IDE Eclipse, será necesario el acceso a internet.

Dentro del banco de trabajo de Eclipse hay que acceder a la pestaña de ayuda e instalar un nuevo software, dentro de la ventana seleccionar añadir repositorio.

1. En los campos que aparecen se deben introducir escribir los siguientes textos:
   - Name: “ADT Plugin”
   - Location: “https://dl-ssl.google.com/android/eclipse/ “

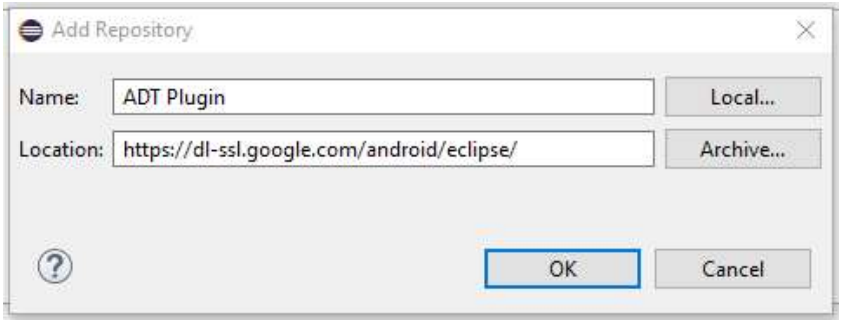

Figura 21 : Instalación online del Kit de Desarrollo Software de Android para Eclipse

2. Una vez introducidas las credenciales de acceso al servidor de descarga del plugin simplemente hay que seguir los pasos que se indican.

3. A continuación, en la barra de herramientas, podremos disponer de dos nuevas utilidades, SDK manager para instalar los paquetes de APIs que se deseen y AVD manager para gestionar el simulador virtual de un Smartphone.

Figura 22 : Gestor de paquetes de Android a la izquierda.
Dispositivo Virtual de Android a la derecha

Se instalaran por defecto solo unas pocas APIs de carácter básico, por tanto, si se desea aprovechar al máximo este kit de desarrollo Android lo aconsejable seria acceder al Android SDK Manager e instalar las APIs disponibles.

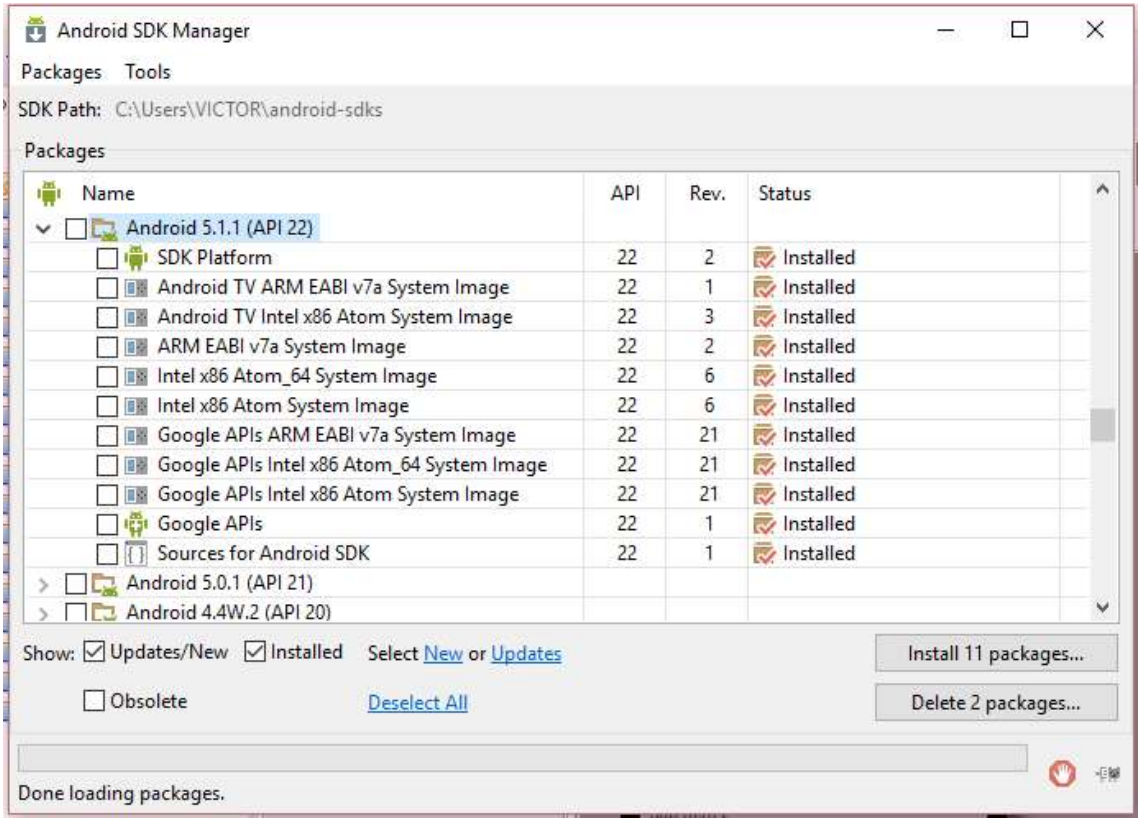

Figura 23 : Listado de paquetes disponibles para instalar en el entorno de desarrollo

6.4 Estructura de un proyecto en Android

Una vez se dispone de todas las herramientas necesarias para empezar a trabajar, se puede comenzar con la iniciación en programación Android.

Dentro del IDE de Eclipse, abrir un nuevo proyecto de la clase proyecto de aplicación Android.

A continuación se mostrara una ventana donde se solicita la información básica del proyecto y las APIs sobre las cuales va a estar disponibles. Tener en cuenta que, cuanto más bajo sea el nivel de la API, un mayor número de dispositivos podrán ejecutar la aplicación a desarrollar, por el contrario, las utilidades que ofrecerá una api de nivel más bajo serán inferiores a las de sus sucesoras.

En la actualidad, más del 50% de usuarios se encuentran utilizando servicios de Android en la versión 6.0 Android Mashmellow y por debajo de la versión de Android 4.0 Android Ice cream sandwich no se supera el 3% de dispositivos del mercado.

Por tanto, estos datos son una buena referencia para determinar la mínima versión de Android compatible con el proyecto a desarrollar con el máximo de APIs o utilidades posibles de las que se pueda beneficiar.

Figura 24 : Primera ventana de creación de proyecto, nombre y niveles de API

Una vez hecho esto, el creador de proyectos dará la opción de poder establecer un icono para la aplicación desarrollada.

Por último, se creara la actividad principal, la cual será una clase o class en programación orientada a objetos y la misma en formato gráfico XML será la primera ventana de la aplicación.

Una vez creado el proyecto, en forma de árbol en la parte izquierda del editor, se tienen todos los archivos necesarios y de utilidad para la aplicación.

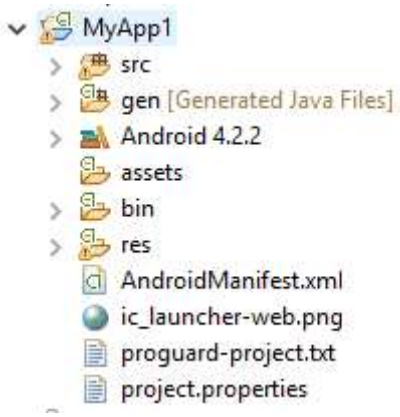

Figura 25 : Estructura y contenido de un proyecto en Android

SRC

La carpeta SRC contiene los paquetes de programación base de la aplicación en formato Java, en caso de las aplicaciones Android, cada paquete tiene las actividades que forman la aplicación o el número de ventanas posibles dentro de la aplicación.

Las actividades, dentro de la carpeta SRC, se encuentran en formato Java o JAR, lo que indica que es código en lenguaje Java y cada uno será una class.

GEN y BIN

Las carpetas GEN y BIN incluyen los archivos generados tras una compilación del programa, entre ellos se incluyen, por ejemplo, el archivo apk o la clase primaria de java R que contiene las identidades de cada ítem incluido en las actividades.

Android X.X.X

En la carpeta de la versión Android seleccionada en la creación del proyecto se encuentran los paquetes con todas las herramientas de las que dispone dicha API incluida.

Herramientas hardware y software que tienen todas las APIs concretas y utilidades como se comentó en el apartado de versiones Android.

RES

La carpeta RES contiene todos los archivos de tipo gráfico, la configuración de las ventanas en formato XML, los layout de cada actividad o los valores iniciales de los recursos básicos de las actividades como pueden ser botones o cuadros de dialogo.

Android Manifest

En este archivo de formato XML se escribirán todos los permisos a los que podrá acceder el usuario y la misma aplicación, en este mismo se podrán habilitar o bloquear actividades si se desea o incluso dar permiso a ciertas APIs como puede ser por ejemplo el uso de cámara.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="com.myapp1"
4       android:versionCode="1"
5       android:versionName="1.0" >
6
7       <uses-sdk
8           android:minSdkVersion="17"
9           android:targetSdkVersion="17" />
10      <application
11          android:allowBackup="true"
12          android:icon="@drawable/ic_launcher"
13          android:label="@string/app_name"
14          android:theme="@style/AppTheme" >
15          <activity
16              android:name="com.myapp1.MainActivity"
17              android:label="@string/app_name" >
18              <intent-filter>
19                  <action android:name="android.intent.action.MAIN" />
20
21                  <category android:name="android.intent.category.LAUNCHER" />
22              </intent-filter>
23          </activity>
24          <activity
```

```
25              android:name="com.myapp1.SecondaryActivity"
26              android:label="@string/title_activity_secondary" >
27          </activity>
28          <activity
29              android:name="com.myapp1.ThirdActivity"
30              android:label="@string/title_activity_third" >
31          </activity>
32      </application>
33  </manifest>
```

Fragmento 1 : Fichero de permisos y definiciones globales

6.4.1 Estructura de una actividad Java

El programa java de cada actividad es inaccesible al usuario y se accede al él desde la interfaz que sería el layout de la misma actividad.

El usuario se comunica con el programa a través de la interfaz de aplicación y esta será quien comunique al programa Java las acciones que debe ejecutar.

Las clases de tipo activity tienen una estructura concreta como se muestra a continuación.

En primer lugar se tiene la definición del paquete al que pertenece

```
1  package com.aaa;
```

Fragmento 2 : Título del proyecto en cada actividad

Las siguientes líneas del programa de aplicación están dedicadas a la implementación de bibliotecas y clases que se utilizaran dentro de la misma actividad.

```
1   import android.app.Activity;
2   import android.content.DialogInterface.OnClickListener;
3   import android.content.Intent;
4   import android.os.Bundle;
5   import android.view.Menu;
6   import android.view.MenuItem;
7   import android.view.View;
8   import android.widget.Button;
9   import android.widget.EditText;
10  import android.widget.TextView;
```

Fragmento 3 : Ejemplos de bibliotecas importadas a una actividad

Una vez identificado en contexto la actividad se comienza a declarar como clase, pudiendo en ella misma implementar acciones como por ejemplo métodos de escucha de botones o clases heredadas de otras como los hilos por ejemplo.

```
1   public class MainActivity extends Activity implements View.OnClickListener{}
```

Fragmento 4 : Definición de una actividad dentro de un proyecto Android

Dentro de la definición de la clase, como en toda función class es posible la creación de un constructor que se pueda llamar desde otra clase, pero en este caso de programas Android, como las actividades solo pertenecerán a un objeto, en principio, no se genera un constructor por defecto.

En lugar de esto, por defecto se tiene una función *onCreate(),* que será la que se ejecute en primera instancia seguidamente de abrir la actividad y se mantendrá dentro de la misma a la escucha de interacciones con la interfaz de usuario.

En esta función será donde se inicializaran las variables locales a la clase como por ejemplo listas, botones o adaptadores.

```
1  protected void onCreate(Bundle savedInstanceState) {
2              super.onCreate(savedInstanceState);
3              setContentView(R.layout.activity_main);
4  }
```

Fragmento 5 : Función de carga continua necesaria dentro de cada actividad

El Fragmento 5 corresponde a la función es la principal en cada activity, sin la cual no se ejecutara ninguna acción ni se podrá acceder a las utilidades.

También se definen por defecto otras funciones adicionales de menor importancia como son la gestión de menú de opciones de la aplicación o la función de escucha del menú de opciones que se activa cada vez que se selecciona un ítem dentro del mismo menú desplegable de opciones.

```
1   public boolean onCreateOptionsMenu(Menu menu) {
2
3               getMenuInflater().inflate(R.menu.main, menu);
4               return true;
5   }
6
7
8   public boolean onOptionsItemSelected(MenuItem item) {
9
10              int id = item.getItemId();
11              if (id == R.id.action_settings) {
12                    return true;
13              }
14              return super.onOptionsItemSelected(item);
15  }
```

Fragmento 6 : Funciones miembro a cada actividad alternativas, creación de menu e interacción con el mismo

Como ya se ha dicho, estas dos funciones son opcionales, dado que es posible que no se vaya a definir un menú dentro de la actividad o la aplicación.

6.4.2 Layout de una actividad

Una vez explicada la estructura interna de cada actividad, se puede definir la interfaz gráfica o el layout al que accederá el usuario ajeno a todo el código interno de la aplicación.

Este archivo será el que utilizara el usuario para comunicarse y ordenar acciones al programa java, oculto al usuario.

La estructura del archivo XML del layout es mucho más sencilla que en el caso de su hermana actividad en java. En esta solo se incluyen descripciones graficas como disposición de los elementos en la pantalla del dispositivo, dimensiones o colores.

En primera instancia, al igual que en el archivo Java, se introduce el contexto del layout como la información de con que actividad esta enlazada o características del fondo de aplicación.

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:background="@color/NEGRO"
10     tools:context="com.myapp1.SecondaryActivity" >
11
12 </RelativeLayout>
```

Fragmento 7 : Definicion de layout de una actividad en formato XML

Como se ha dicho, se indica el contexto del layout de la actividad, en el ejemplo *SecondaryActivity*, las dimensiones graficas de la actividad y el color de fondo de la misma.

Todos los elementos gráficos, como los que se definirán a continuación, deberán de tener una serie de características o atributos que serán definidos en el archivo gráfico o layout XML.

A continuación se mostraran una serie de características básicas de los recursos o atributos, en la biblioteca de Android se pueden encontrar muchas de estas características.

- *Android:id* : Con este atributo se define el nombre identificador de cada uno de los elementos gráficos y, de esta manera, poder acceder a ellos cómodamente. Por ejemplo, si se tienen varios botones y se pulsa uno de ellos, el id nos permite identificar cuál de ellos ha activado la función *onCLickListener()*.

- *Android:layout_width* y *Android:layout_height* : Estos atributos sirven para establecer las dimensiones graficas del elemento, las dimensiones se definirán con un número real acompañado de los caracteres dp (densidad de pixeles, dimensión fija), sp (scale pixeles, dimensión variable), in(pulgadas), mm(milímetros).

- *Android:text* : Atributo que permite fijar un texto dentro del elemento gráfico.

- *Android:background* : Atributo que permite definir el color de fondo del objeto.

- *Android:textColor*: Atributo que permite definir el color del texto contenido en el elemento gráfico.

Dentro de esta definición se especificaran las características graficas de cada recurso básico o vista como los botones, listas o campos de texto.

Por otro lado, dentro de la misma edición de layout, se puede editar y modificar este mismo interfaz de usuario de forma gráfica sin tener que modificar líneas de código en XML. Esta utilidad es bastante cómoda para los desarrolladores novel no acostumbrados a los nombres de los recursos empleados.

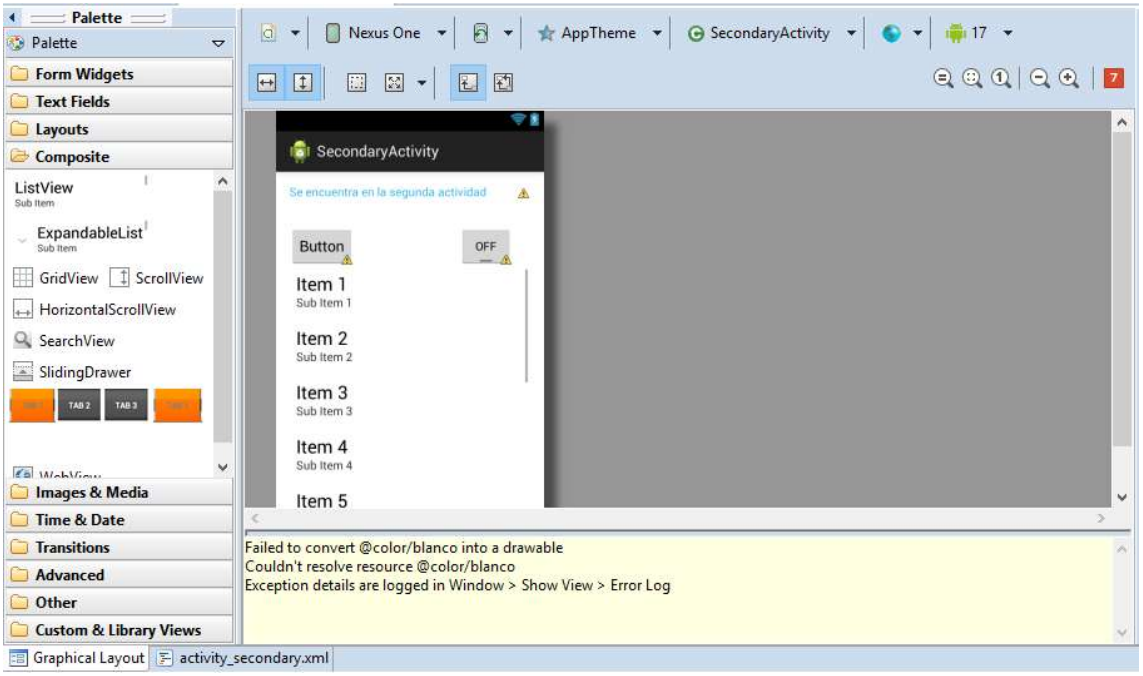

Figura 26 : Entorno de edición de layout de una actividad sin tener que acceder al fichero XML

En este otro tipo de ventana, la edición de la interfaz se hace mucho más rápida y visual, de tal forma que, a la izquierda se definen todo tipo de vistas incluidas en la versión de Android mínima utilizada en el proyecto, o mínimo nivel de API necesario para poder ejecutarlo.

Para ubicar un recurso dentro de la actividad, basta únicamente con pinchar, arrastrar y soltar sobre la posición final en la que se desee colocar el ítem seleccionado.

Se pueden editar también las dimensiones desde esta ventana como si se tratara de imágenes dentro de un editor de texto.

También destacar, que todas las operaciones que se lleven a cabo dentro de este editor gráfico de interfaz, quedarán reflejadas dentro del fichero XML que se ha comentado anteriormente.

En esta misma ventana se dispone de un desplegable en la parte superior derecha que indica la API que se está simulando en este entorno gráfico, de este modo se puede comprobar la compatibilidad del layout creado.

6.4.3 Recursos básicos o vistas

Dentro de la edición de la aplicación en Android se pueden encontrar diferentes elementos gráficos que darán el toque personal y necesario al programa que se esté desarrollando.

Dentro de estos recursos o vistas de carácter básico se pueden mencionar los siguientes, más comunes en casi todas las aplicaciones del mercado:

- *TextView*: Elemento gráfico que servirá para mostrar fragmentos de texto estático definido por el programador o texto dinámico que, según se defina en el programa java, tome diferentes valores o cadenas.

- *Button*: Elemento gráfico que normalmente servirá para dar comienzo a una acción concreta, como por ejemplo lanzar otra ventana dentro de la misma aplicación.
  La consecuencia de pulsar dicho botón de definirá en el programa java con una función denominada *onCLickListener(),* que como indica, se lanzara cuando se realice un click.

- *EditText*: En este caso, se trata de un elemento gráfico en el cual el usuario puede insertar caracteres o números y estos quedarán almacenados en una variable interna del programa.

- *ListView*: Elemento gráfico que, al igual que el *TestView*, puede ser estático o variable en función que como se defina (si está relacionado con un arreglo estático o dinámico). Se trata de una lista de elementos consecutivos.

6.4.4 Manejo de actividades

En lo referente a las actividades o ventanas de programa, se pueden tener tantas como se deseen y siempre deberán tener un archivo de programación java y su correspondiente archivo gráfico.

Cada una de las actividades tendrá sus propios elementos gráficos (TextView, button…) con id distintas.

Para la navegación entre actividades se usaran los lanzadores o *intents* que, cuando se realice una acción concreta se llamara a dichos intents, y estos podrán llamar a funciones como *startActivity*, por ejemplo:

```
1  Intent intent1 = new Intent(MainActivity.this,SecondActivity.class);
2  startActivity(intent1);
```

Fragmento 8 : Salto de una activiad a otra secundaria

6.4.5 Manejo de listas

Como ya se ha mencionado antes, las listas mostraran una serie de elementos definidos dentro de vectores ya sean estáticos o sean dinámicos, la *ListView* tomara dicho carácter.

Para poder lanzar una lista de elementos se necesita de un adaptador que ya se encuentra definido en la clase Android con el título de variable *ArrayAdapter<Type>*, este adaptador permitirá enlazar el elemento gráfico con el objeto contenedor de los datos, definiéndolo de la siguiente manera:

```
1  adaptador = new
2  ArrayAdapter<String>(this,android.R.layout.simple_expandable_list_item_1,miArray);
3
4  miLista.setAdapter(adaptador);
```

Fragmento 9 : Definición de adaptador de lista y vinculación del adaptador al elemento gráfico

Donde adaptador se corresponde con el *ArrayAdapter* creado, *miArray* es el objeto contenedor de los datos y *miLista* se corresponde con el *ListView* de la aplicación.

De esta forma aparecerá en el programa una lista con los datos del objeto contenedor.

Si lo que se desea es hacer una lista dinámica que siga al vector o *array* al que se le ha asignado, cada vez que se efectúa un cambio como una adición o eliminación hay que notificarlo.

6.5 Android y la API Bluetooth

Con todas las nociones básicas sobre la programación en Android, el entorno de trabajo y las herramientas de trabajo necesarias para el correcto desarrollo de un primer proyecto basándose en la tecnología y las APIs de Bluetooth se puede empezar a trabajar y a crear.

6.5.1 Declaración de permisos

Antes de empezar a trabajar con las herramientas Bluetooth en Android será necesario dar una serie de autorizaciones o permisos de acceso a la API correspondiente de Bluetooth.

Estos permisos deberán solicitarse dentro del archivo de texto XML de *Android Manifest* que se comentó con anterioridad en la estructura de los proyectos Android.

En el caso del uso del Bluetooth deberán aplicarse tres permisos para que la aplicación pueda ser utilizada de forma general por cualquier versión de Android a partir de la que se haya fijado en el proyecto.

- Permisos básicos de Bluetooth.

```
1  <uses-permission android:name="android.permission.BLUETOOTH" />
2  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Fragmento 10 : Permisos para poder utilizar Bluetooth en cualquier dispositivo Android

- Permisos avanzados de Bluetooth para habilitar herramientas en versiones de Android 6.0 y superiores.

```
1  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
2  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Fragmento 11 : Permisos para poder utilizar Bluetooth en versiones Android iguales o superiores a Android Nougat

6.5.2 Comprobación y conexión Bluetooth

Antes de comenzar con cualquier aplicación que vaya a necesitar del uso de la comunicación Bluetooth, primero se debe verificar que el dispositivo que ejecutara la aplicación tiene disponibilidad para acceder a las utilidades de dicha comunicación.

Para ello se necesitara de la clase o librería Bluetooth.

Primero se creara un objeto adaptador de Bluetooth para poder acceder a las utilidades de esta tecnología.

```
1  mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

Fragmento 12 : Creación de objeto adaptador Bluetooth por defecto

Se puede decir, que el adaptador Bluetooth que se crea una vez se llama a su constructor, es el carnet de identidad Bluetooth del dispositivo, este mismo contiene información como la *MAC Address* del propietario o el nombre que mostrara a los demás dispositivos que lo vean.

Una vez sabido esto, se puede decir, que lo primero que hay que buscar para localizar un dispositivo Bluetooth es su adaptador.

Ya estando creado el adaptador, se debe verificar que el dispositivo dispone de comunicación Bluetooth con la siguiente condición, ya que es posible que el dispositivo que se esté utilizando no soporte la tecnología Bluetooth, como ocurre como el ADV del emulador, ya que al no ser un dispositivo real, no dispone de características hardware como la tarjeta de comunicación Bluetooth.

```
1  mBluetoothAdapter != null
```

Fragmento 13 : No es posible crear un adaptador o no se ha creado aún

Si la condición anterior se cumple entonces se puede pasar a la comprobación de, si las comunicaciones Bluetooth están habilitadas o no con la siguiente función miembro de la clase *BluetoothAdapter*:

```
1  mBluetoothAdapter.isEnabled()
```

Fragmento 14 : Comprobación de sí el adaptador se encuentra habilitado, Bluetooth activo

Con esto se puede comprobar la posibilidad de conexión con Bluetooth y su estado actual de activación.

La función anterior permitiría condicionar el acceso a ciertas acciones del programa, como por ejemplo enlazar dispositivos o buscar, dado que para llevar a cabo estas acciones es estrictamente necesario haber creado un adaptador y haberlo habilitado para las comunicaciones Bluetooth.

6.5.3 Detección de dispositivos Bluetooth

Una vez activadas las comunicaciones y las herramientas Bluetooth lo siguiente que se puede hacer es detectar e identificar los dispositivos Bluetooth conectados y visibles dentro del radio de acción del dispositivo que se esté utilizando.

Para comenzar a detectar dispositivos se debe habilitar el modo *discovery* del dispositivo con el Fragmento 15.

```
1  mBluetoothAdapter.startDiscovery();
```

Fragmento 15 : Orden para que el adaptador Bluetooth comience a buscar

Para comenzar la búsqueda de dispositivos en el área se llama a un *registerReceiver* que necesita en su construcción de un *broadcast* receiver que se mantenga activo mientras el adaptador este en modo descubrimiento y la condición de activación que en caso de querer buscar dispositivos, la condición sería un ACTION_FOUND.

```
1  registerReceiver(bReceiver, new IntentFilter(BluetoothDevice.ACTION_FOUND));
```

Fragmento 16 : Llamada al receptor de dispositivos entrantes

A continuación, al igual que los adaptadores, se creara el elemento emisor-receptor que se mantendrá constantemente a la escucha mientras el adaptador se mantenga en su estado de *Discovery*, cuando este *BroadcastReceiver* detecte un dispositivo entrante, podría almacenar la información del dispositivo detectado en una lista de objetos de tipo *BluetoothDevice*, cuya estructura será similar a la del *BluetoothAdapter*.

Si luego se desea, por ejemplo, mostrar esa información en una *ListView*, se almacenara la información en formato *String*.

```
1  BroadcastReceiver mReceiver = new BroadcastReceiver() {
2      public void onReceive(Context context, Intent intent) {
3          String action = intent.getAction();
4          if (BluetoothDevice.ACTION_FOUND.equals(action))
5          {
6            BTArrayAdapter.add(device.getName() + "\n" + device.getAddress()");
7            BTArrayAdapter.notifyDataSetChanged();
8            deviceList.add(new DevLocalized(device, rssi));
9           }
```

```
10      }
11  };
```

Fragmento 17 : Receptor de dispositivos, si se detecta uno almacena información e imprime en pantalla

6.5.4 Medición de intensidad de la señal

Para obtener la intensidad de la señal de un dispositivo que se encuentra usando los servicios de la comunicación Bluetooth se procede de la misma forma que para detectar un dispositivo, solo que en este caso, una vez invocado el *broadastReceiver* se accederá al dato de la señal RSSI del dispositivo en lugar de a sus características generales.

```
1  int rssi =intent.getShortExtra(BluetoothDevice.EXTRA_RSSI,Short.MIN_VALUE);
2  rssiArray.add(String.valueOf(rssi));
```

Fragmento 18 : Obtención de la intensidad de la señal recibida del dispositivo entrante

Una vez hecho esto ya es posible disponer de unos vectores formados por los dispositivos detectados dentro del área de búsqueda del dispositivo localizador, y la información necesaria de los dispositivos para poder realizar un buen posicionamiento en el espacio.

6.5.5 Dispositivos enlazados

Del mismo modo que para localizar dispositivos desconocidos, aunque visibles, es posible obtener una lista o conjunto de los dispositivos actualmente enlazados con el dispositivo que ejecute la aplicación, aunque en este caso la información viene de la misma memoria del dispositivo y, por tanto el proceso de obtención de la lista será más rápido, y por el contrario se tiene que de estos dispositivos no es posible obtener una intensidad de la señal ya que no siempre tienen por qué estar conectados a la tecnología Bluetooth aunque se encuentren en la lista de dispositivos enlazados o emparejados.

Para obtener la mencionada lista bastara con crear un set y llenarlo con *getBondedDevices()*, función que estará disponible dentro de la clase del adaptador Bluetooth.

```
1   Set<BluetoothDevice> pairedDevices;
2
3   pairedDevices = myBluetoothAdapter.getBondedDevices();
4
5   BTArrayAdapter.clear();
6   devList.clear();
7
8   for(BluetoothDevice device : pairedDevices) {
9       BTArrayAdapter.add(device.getName()+ "\n" + device.getAddress());
10      devList.add(device);
11  }
```

Fragmento 19 : Generación de lista de dispositivos enlazados

También tienen el formato de *BluetoothDevice* y por tanto, se puede añadir a una *ListView* de tal forma que se muestre la información precisa de los dispositivos que se tienen vinculados.

6.5.6 Sistema de identificación de dispositivos Bluetooth

Con todo esto se podría comenzar a desarrollar un primer programa que active y desactive las herramientas Bluetooth, detecte y obtenga información de dispositivos presentes dentro del área de búsqueda del dispositivo que se encuentre ejecutando la aplicación.

Se definen cuatro botones y una lista como elementos visuales o recursos de interfaz, los botones pueden corresponderse con los de activar Bluetooth, desactivar Bluetooth, ver dispositivos enlazados y buscar dispositivos.

Cuando se pulse cualquier botón se lanzará un on*ClickListener(),* función que está constantemente a la escucha, declarada dentro de la función miembro *onCrate* de la actividad correspondiente.

```
1   Button = (Button)findViewById(R.id.interfaceButton);
2
3   Button.setOnClickListener(new OnClickListener() {
4
5               @Override
6               public void onClick(View v) {
7                   //Actuar cuando se pulse interfaceButton
8               }
9   });
```

Fragmento 20 : Función de escucha de botones de interfaz

Una vez entendido el concepto de actuación de los botones, sobre la primera aplicación de identificación de dispositivos, el botón de activar (*on*), cuando se pulse, llamara a su *onClick* correspondiente y tendrá dos opciones: En caso de que, la comunicación Bluetooth se encuentre ya activada mostrará por pantalla una pequeña ventana emergente que notificara al usuario que el Bluetooth se encuentra en ese momento activado, en caso de que no se encuentre activado, con un intent, lanzara otra pequeña ventana emergente como la que se muestra en la Figura 27 ya definida en la API de Bluetooth que preguntara al usuario si desea activar, dando a este la opción.

```
1   onBtn = (Button)findViewById(R.id.turnOn);
2   onBtn.setOnClickListener(new OnClickListener() {
3
4         @Override
5         public void onClick(View v) {
6             if (!myBluetoothAdapter.isEnabled()) {
7                 Intent turnOnIntent = new Intent
8                         (BluetoothAdapter.ACTION_REQUEST_ENABLE);
9                 startActivityForResult(turnOnIntent, REQUEST_ENABLE_BT);
10
11                Toast.makeText(getApplicationContext(),"Bluetooth turned on" ,
12                        Toast.LENGTH_LONG).show();
13            }else{
14                Toast.makeText(getApplicationContext(),"Bluetooth is already on",
15                        Toast.LENGTH_LONG).show();
16            }
17        }
18  });
```

Fragmento 21 : Cuando se pulsa el botón de activar Bluetooth se lanza ventana emergente, lineas 7, 8 y 9

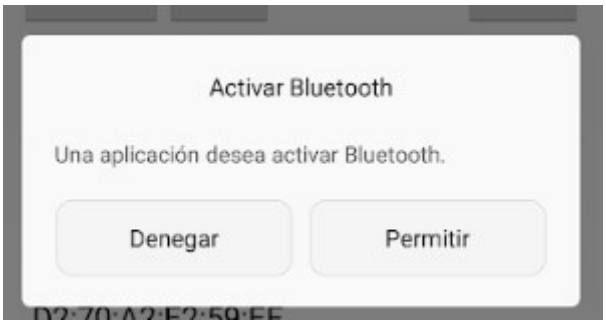

Figura 27 : Ventana emergente de solicitud de activación Bluetooth

Para el botón de desactivación será algo similar solo que, en esta ocasión no solicitara petición al usuario, sino que cancelara el adaptador directamente.

```
1   offBtn = (Button)findViewById(R.id.turnOff);
2   offBtn.setOnClickListener(new OnClickListener() {
3
4           @Override
5           public void onClick(View v) {
6              myBluetoothAdapter.disable();
7
8              Toast.makeText(getApplicationContext(),"Bluetooth turned off",
9                  Toast.LENGTH_LONG).show();
10          }
11  });
```

Fragmento 22 : Función de desactivación Bluetooth, BluetoothAdapter.disable()

Pasando a las acciones de búsqueda y muestreo de dispositivos, el procedimiento es algo más distinto.

En el caso de la vista de dispositivos enlazados, simplemente bastara con incluir el Fragmento 19 dentro del método *onClick()* de su botón correspondiente, después de reiniciar el adaptador del *ListView* de la actividad.

En el caso de la búsqueda de dispositivos se hará referencia al Fragmento 17 desde el Fragmento 23, en el cual primero se comprueba si el adaptador Bluetooth del dispositivo ya se encuentra buscando dispositivos, en caso afirmativo cancelará la búsqueda y reiniciara todas las listas correspondientes (lista de dispositivos y adaptador de *ListView*). A continuación comenzara la búsqueda y llamara al Fragmento 17 que actuara en consecuencia de encontrar un nuevo dispositivo.

```
1   findBtn = (Button)findViewById(R.id.search);
2   findBtn.setOnClickListener(new OnClickListener() {
3
4           @Override
5           public void onClick(View v) {
6               if (myBluetoothAdapter.isDiscovering()) {
7                   myBluetoothAdapter.cancelDiscovery();
8               }
9               devLocalized.clear();
10              BTArrayAdapter.clear();
11              myBluetoothAdapter.startDiscovery();
12
13              registerReceiver(bReceiver, new
```

```
14                      IntentFilter(BluetoothDevice.ACTION_FOUND));
15              }
16      });
```

Fragmento 23 : Pulsando el botón de buscar dispositivos, se sincronizan listas y se llama al receptor

Finalmente ya se dispone de un programa que es capaz de controlar los servicios de activación y desactivación del Bluetooth y que puede descubrir y almacenar la información de los dispositivos que encuentre en sus alrededores.

El resultado de la aplicación será como el mostrado en la figura siguiente.

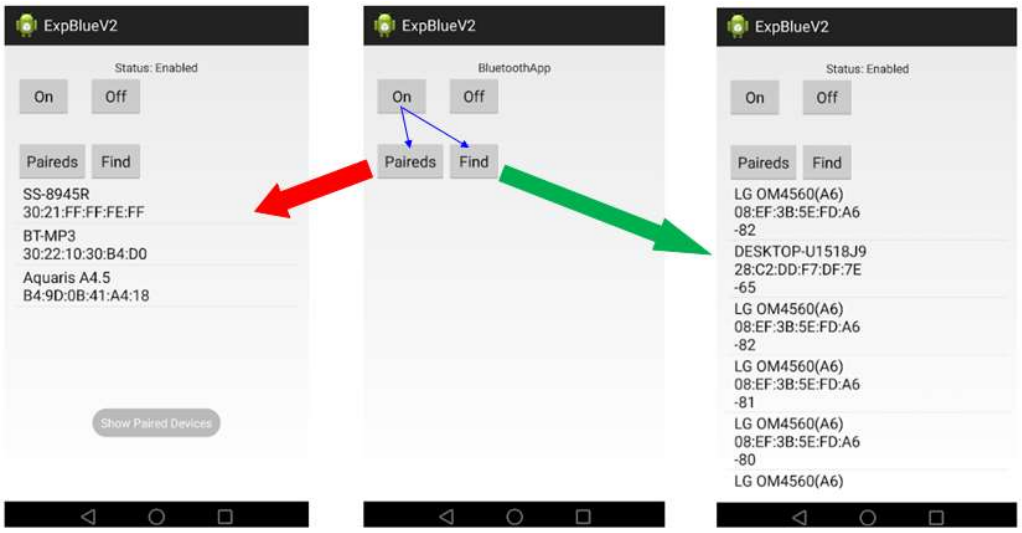

Figura 28 : Funcionamiento de primera aplicación de rastreo e identificación de dispositivos Bluetooth

6.5.7 Sistema de balizas, paradigma cliente-servidor

Con el programa anterior de un ejemplo de sistema de identificación de dispositivos Bluetooth se puede encontrar la posición relativa al dispositivo buscador en función de la potencia de la señal recibida RSSI.

Pero solo con eso no basta para identificar un dispositivo en el plano, con un dispositivo buscador se obtiene una intensidad de la señal, que se puede traducir en distancia ya que es inversamente proporcional, pero lo que se obtiene es un radio en cuya circunferencia descrita se encuentra el dispositivo a posicionar, el problema es que puede estar en cualquier punto de ella.

Como se ha comentado en el planteamiento inicial, puede haber varias soluciones para abordar este problema. Una de ellas es incluir otros dos dispositivos a mayores con la misma función que el primero, buscar y detectar dispositivos que se encuentren utilizando servicios Bluetooth.

Con dos dispositivos Bluetooth, la posición del dispositivo a detectar se reduce a dos únicos puntos que se obtendrán de forma matemática con la intersección de las dos circunferencias

formadas por los valores de RSSI medidos en cada uno de los detectores como se mostraba en la Figura 1.

El hecho de añadir un tercer dispositivo reducirá la solución a un único punto posible como posición del dispositivo a localizar. Sin necesidad de calcular ningún punto de intersección entre las tres circunferencias (ya que será complicado por la desviación de la distancia en función de la RSSI), es posible discriminar uno de los dos puntos obtenidos anteriormente.

La distancia que se obtenga del tercer buscador definirá cuan de los dos puntos posibles es el mejor candidato en función de la cercanía del punto obtenido por el tercer buscador y los otros dos. Solamente uno de ellos estará más cerca que el otro y por tanto este será el punto que defina la posición en el plano del dispositivo a localizar.

La segunda opción, consistía en obtener el punto final por medio de ecuaciones trigonométricas en referencia al triángulo formado por las distancias obtenidas por dos dispositivos buscadores hasta el objetivo y la distancia entre ellos mismos. En esta ocasión, como se mostraba en la Figura 3, únicamente se necesitarían dos buscadores.

La segunda opción consistía en obtener el punto final discriminando uno de los dos puntos obtenidos por la intersección de dos circunferencias, partiendo de dos balizas, en función del semiplano en el que se encuentre ya que, si se establece la posición de las balizas en el plano, quedan definidos los ejes X e Y del plano de búsqueda.

En ambas soluciones se presenta el problema de que, para llevar a cabo los cálculos pertinentes, es necesario que un dispositivo conozca todos los datos (distancias de buscadores a objetivo, distancias entre buscadores, posiciones en el plano de buscadores…) y para ello se plantea la opción de establecer un maestro que coordine todas las operaciones del sistema de localización.

Partiendo de la idea de introducir un maestro que coordine los movimientos y acciones de los buscadores, se puede empezar a nombrar a los buscadores con el termino de balizas, ya que en el momento en que se introduzca a un dispositivo gestor, estos dispositivos buscadores tendrán únicamente la función de buscar objetivos o dispositivos Bluetooth en el área de búsqueda y transmitir la información recabada a un órgano central que procese la información recibida de todas las balizas. El planteamiento de acciones a realizar por cada tipo de dispositivo dl sistema seria el siguiente:

- Balizas: Dispositivos buscadores, se encargaran de rastrear en su área de acción los posibles dispositivos ajenos al sistema que estén utilizando servicios o comunicaciones de la tecnología Bluetooth, obtendrán el identificador del dispositivo y la intensidad de la señal recibida, enviaran al organismo central la información que hayan conseguido de los dispositivos presentes.

- Maestro: Dispositivo procesador y muestreador de resultados, este dispositivo se encargara de recibir o solicitar la información pertinente a cada baliza para poder realizar los cálculos correspondientes y obtener una posición concreta de los dispositivos que estén dentro del área de búsqueda. Finalmente mostrará los

resultados obtenidos para que el usuario pueda ver la ubicación de los dispositivos encontrados.

Este procedimiento necesita de una cierta sincronización entre los dispositivos participantes dentro del sistema de localización. Para ello se presentan varios paradigmas dentro de las comunicaciones entre procesos y, en este caso, entre dispositivos que utilizan la comunicación Bluetooth.

- Paradigma cliente – servidor : El paradigma del cliente y el servidor es el más conocido en arquitecturas de red de internet, solo que también es posible aplicarlo a cualquier tipo de sincronización de dos procesos independientes. Consta de diversos servidores, que como el propio nombre indica proveen de servicios de cualquier tipo al cliente que acceda a la comunicación, por el otro lado se encuentran los clientes que se encargan únicamente de solicitar servicios de cualquier tipo a un servidor determinado.

- Paradigma productor –consumidor : El problema del productor va enfocado al suministro de recursos, en este caso, el productor será el que almacene los recursos en un *buffer* a medida que los vaya produciendo, por otro lado está el consumidor que será quien recoja los recursos almacenados por el productor de ese *buffer*.

- Paradigma lector – escritor: Este paradigma está enfocado al acceso de recursos por parte de distintos nodos dentro de una misma red, dentro de la cual, cada uno tiene una prioridad u otra. Los escritores podrán acceder a la escritura del recurso en cuestión siempre y cuando no haya ningún lector o, en casos particulares, haya un número limitado de lectores, en cambio los lectores podrán tener acceso continuo a no ser que el número de lectores alcance el límite máximo.

- Paradigma gestor – agente: Dentro de una misma red de dispositivos se encuentra uno de ellos que recibe el nombre de gestor que suele ser el que hace de interfaz con el usuario y que, a petición de este, realiza las gestiones pertinentes y le encarga a los agentes que el crea convenientes que realicen las acciones por él y le notifiquen los resultados.

Para la sincronización de un sistema formado por dos o más balizas que se van a encargar de buscar información y enviarla a un maestro que la gestionara conforme crea oportuno, se puede descartar el planteamiento del problema lector – escritor ya que no hay ningún recurso por el cual deban competir ambos dispositivos, en cuanto a los otros tres planteamientos de solución al problema de la sincronización para compartir datos, cualquiera de ellos es válido.

En el caso del productor – consumidor, el maestro haría las veces de consumidor y las balizas de productores que llenan un buffer independiente cada una de ellas, el problema de esto es que, la velocidad de producción pueda ser mucho más rápida que la de consumo y en este caso se perdería la información en tiempo real captada por las balizas.

En el caso del paradigma del gestor – agente, el maestro haría el papel de gestor dado que es el interfaz de usuario y es quien se encargaría de gestionar las operaciones de todo el sistema de localización, y los agentes en este caso serían cada una de las balizas que, a petición del maestro realizarían las acciones oportunas de búsqueda de dispositivos y se lo mandarían al gestor. En este caso el problema que se presenta es que, para poder buscar dispositivos por parte de las balizas es necesaria una petición por parte del gestor o el maestro y esto puede llevar bastante tiempo si una de las balizas queda bloqueada ya que el proceso de comunicación es secuencial.

Por último, en el paradigma del cliente y el servidor, los dos tipos de dispositivo podrían hacer el papel de servidor o de cliente, pero para tener un buen rendimiento del sistema, lo aconsejable seria disponer del maestro como servidor a la espera de recopilar la información necesaria por parte de alguna de las balizas, mientras tanto, las balizas harían de clientes que realizan sus operaciones de búsqueda y transmiten información al servidor, la acción de búsqueda y transmisión las realizaría periódicamente de tal forma que, el servidor o maestro tenga en todo momento la información actualizada de cada baliza. Este paradigma presentaría el problema de colisión entre clientes para el acceso al servidor.

De todos estos paradigmas de resolución de sincronización entre nodos el más recomendado, para esta aplicación, sería el de cliente – servidor debido a que permite al maestro disponer de la información actualizada de las balizas y realizar los procesos de forma paralela, en cuanto a los problemas de colisiones entre clientes, se pueden configurar para que repitan el ciclo de búsqueda y retransmisión de forma indefinida hasta que consigan conectar con el servidor y poder mandar a este la información.

Esta arquitectura podría quedar reflejada de la siguiente forma como se muestra en la Figura 29.

Figura 29 : Arquitectura funcional del sistema de comunicación entre maestro y balizas

6.5.8 Bluetooth Sockets

Una vez tomada una decisión, y decidir abordar el problema como uno de tipo cliente – servidor, lo necesario ahora es establecer una comunicación entre los dispositivos para que estos puedan compartir información entre sí.

Para poder llevar a cabo una comunicación por métodos inalámbricos, la posibilidad es realizarlo con el protocolo TCP actualmente utilizado en internet, estableciendo la conexión entre ambos host intervinientes por medio de una dirección de acceso al medio o MAC, una dirección de dispositivo como puede ser la del protocolo de internet o IP, y un puerto de acceso dentro del *host* destino.

Este tipo de comunicaciones se realizan mediante *sockets* o canales temporales de comunicación entre nodos de la red, este tipo de canales necesita únicamente de datos como direcciones de acceso para poder enlazar los dos extremos de la comunicación. Cada interlocutor creara su *socket* respectivo de tal forma que, de forma global se tenga un canal bidireccional.

El proceso de comunicación de un *socket*, como se muestra en la Figura 30, primeramente se crea el *socket* en el cliente con la información que necesita del nodo destinatario o servidor, del mismo modo se crea el *socket* en paralelo del servidor solo que en este caso no necesita de información acerca del destinatario ya que será arbitrario.

Una vez creado el socket del servidor, este se mantiene a la escucha de nodos entrantes que soliciten conexión con el servidor, el mismo socket del servidor será quien se encargue de aceptar o no la conexión solicitada por parte del cliente, esto se suele administrar con unas claves de acceso al servidor.

Cuando el *socket* del servidor acepta la entrada del cliente se crea el hilo o canal de comunicación entre ambos y se mantendrán en comunicación (envío y recepción de datos) mientras no se cierren los respectivos *sockets* de comunicación.

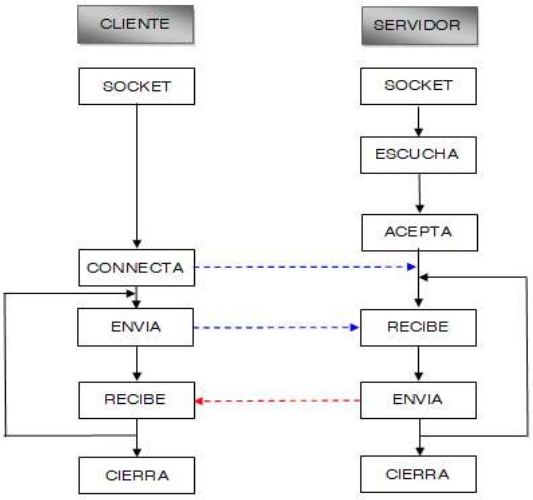

Figura 30 : Diagrama de comunicación entre cliente y servidor mediante *sockets*

En el caso práctico de las comunicaciones Bluetooth la configuración de la comunicación entre *sockets* es similar, el servidor crea su *socket* y se mantiene a la escucha de dispositivos que deseen establecer conexión con este, mientras tanto, el cliente crea su socket y lanza su petición de acceso al servidor con la clave de acceso predefinida.

Una vez aceptada la petición de comunicación se abren los canales de comunicación y sus correspondientes *buffers* que les permitirán a ambos host escribir y recibir mensajes del otro interlocutor.

Finalmente, cuando no se desee continuar con la comunicación, se cerraran lo *sockets*.

Como ya se ha mencionado y se indica en el Fragmento 24, el servidor crea su *socket* y se mantiene a espera de petición de conexión hasta aceptar dicha solicitud.

El elemento *mServerSocket* será el *socket* del servidor, para cuya creación se necesita del nombre del servidor y de la clave de acceso al mismo que será el *bluetoothUUID*.

```
1  mServerSocket = myBluetoothAdapter.listenUsingRfcommWithServiceRecord(name,
2  bluetoothUUID);
3  mBluetoothSocket = mServerSocket.accept();
```

Fragmento 24 : Creación del socket servidor y espera de aceptación de clientes por parte del mismo

Por otro lado en el Fragmento 25 se observa cómo, el cliente, crea su socket y solicita la conexión con los parámetros necesarios al servidor, siendo el elemento *mBluetoothSocket* el *socket* del cliente, el elemento *device* el adaptador Bluetooth del servidor y *bluetoothUUID* la clave de acceso al servidor para que la solicitud sea aceptada.

```
1  mBluetoothSocket = device.createRfcommSocketToServiceRecord(bluetoothUUID);
2  mBluetoothSocket.connect();
```

Fragmento 25 : Creación del socket cliente y petición de conexión con el servidor

Por otro lado será necesario crear una clase que gestione los *buffers* de comunicación de los sockets servidor y cliente, un hilo de comunicación. En esta clase se crearán las corrientes de

entrada y de salida a modo de constructor y tendrá dos funciones miembro encargadas de leer y escribir respectivamente.

La función de escritura introducirá una cadena de bytes en la corriente de salida, y la función de lectura cargara una cadena de caracteres desde la corriente de entrada.

De esta forma quedaría organizada la class de gestión de mensajes entre *sockets*, como se indica en el Fragmento 26.

```
1   //STREAM CLASS
2   //CONSTRUCTOR       :     CREATE A COMMUNICATIONS STREAMS (BUFFERS) INPUT
3                               AND OUTPUT
4   //READ              :     EXTRACT BYTES FROM IN BUFFER AND SAVE IN A
5                               STRING VARIABLE
6   //WRITE             :     WRITE BYTES IN THE OUT BUFFER
7
8   private class StreamThread extends Thread {
9
10      private InputStream in;
11      private OutputStream out;
12
13      public StreamThread() {
14             try {
15                 in = mBluetoothSocket.getInputStream();
16                 out = mBluetoothSocket.getOutputStream();
17             } catch (IOException e) {}
18      }
19
20      public String read() {
21             byte[] data = new byte[30000];
22             int length;
23             String text = null;
24             while (true) {
25                  try {
26                      length = in.read(data);
27                      text = new String(data, 0, length);
28
29                  } catch (IOException e) {
30                      text = "Empty buffer";
31                  }
32                  break;
33             }
34             return text;
35     }
36
37     public void write(byte[] data) throws IOException {
38            try {
39                out.write(data);
31             }catch (Exception e) {}
40     }
41  }
```

Fragmento 26 : Gestor de tráfico entrante y saliente de mensajes entre sockets de maestro y cliente

Una vez creado el gestor de trafico de mensajes y de *buffers* entre sockets es posible crear una class para cada tipo de *host* del sistema que se encargue de realizar las operaciones correspondientes como se indicó en la Figura 30.

De este modo se tendría el Fragmento 27 que correspondería a la estructura de conexión del servidor.

```
//SERVER CLASS
//CONSTRUCTOR       :      CREATE A BLUETOOTH SERVER SOCKET
//RUN               :      ACCEPT A CLIENT COMM / SEND A MESSAGE IF SOMEONE
                           IS LISTENING / RECEIVE A MESSAGE FROM CLIENT /
                           CLOSE SOCKETS
//CLOSE             :      CANCEL THE SERVER SOCKET

private class ServerThread extends Thread {

    private BluetoothServerSocket mServerSocket;

    public ServerThread(String name) {
             mServerSocket = null;
             try {
                 mServerSocket = myBluetoothAdapter
                 .listenUsingRfcommWithServiceRecord(name, bluetoothUUID);
             } catch (IOException e) {
                 Toast.makeText(getApplicationContext(),"Server NOT
                           AVAILABLE" , Toast.LENGTH_LONG).show();
              }
    }

    public void run() {

             while (mServerSocket!=null) {
             try {

                 mBluetoothSocket = mServerSocket.accept();
                 if (mBluetoothSocket != null) {
                     StreamThread mStreamThread = new StreamThread();
                     mStreamThread.start();

                    String a = message;
                    byte[] data = a.getBytes();
                    mStreamThread.write(data);

                     message = mStreamThread.read();
                     try {
                         mBluetoothSocket.close();
                     } catch (IOException e) {}

                     try {
                         mServerSocket.close();
                     } catch (IOException e) {}
                                 mServerSocket = null;
                     }

              } catch (IOException e) {
                     Toast.makeText(getApplicationContext(),"Server NOT
                          accept a client" , Toast.LENGTH_LONG).show();
              }
              break;
              }
       }

```

```
56  }
57
```

Fragmento 27 : Clase de gestión del socket servidor, control de creación, conexión y cierre

Y en el Fragmento 28 queda representado el código de la clase correspondiente a la ejecución del *socket* del cliente.

```
1   //CLIENT CLASS
2   //CONSTRUCTOR       :       CREATE A BLUETOOTH SOCKET
3   //RUN               :       CONNECT WITH SERVER AND SEND MESSAGE
4
5   private class ConnectThread extends Thread {
6
7           public ConnectThread(BluetoothDevice device) {
8                   try {
9                       mBluetoothSocket =
10  device.createRfcommSocketToServiceRecord(bluetoothUUID);
11                  } catch (IOException e) {
12                       Toast.makeText(getApplicationContext(),"client NOT
13                                  AVAILABLE" , Toast.LENGTH_LONG).show();
14                  }
15          }
16
17          public void run() {
18                  try {
19                      mBluetoothSocket.connect();
20                      if (mBluetoothSocket != null) {
21                          StreamThread mStreamThread = new StreamThread();
22                          mStreamThread.start();
23
24                         byte[] data = message.getBytes();
25                         mStreamThread.write(data);
26
27                          Toast.makeText(getApplicationContext(), message,
28                                    Toast.LENGTH_LONG).show();
29
30                              }
31                  } catch (IOException e) {
32                      Toast.makeText(getApplicationContext(),"Server NOT
33                            accept a client" , Toast.LENGTH_LONG).show();
34                  }
35          }
36
37          public void cancel() {
38                  try {
39                      mBluetoothSocket.close();
31                  } catch (IOException e) {}
40          }
41
42  }
```

Fragmento 28 : Clase de gestión del socket cliente, control de creación, conexión y cierre

Una vez configurada la interconexión de los *sockets* y de sus correspondientes dispositivos, sobre la aplicación Android en desarrollo se pueden incluir dos botones como recurso de vista para poder definir cada dispositivo como baliza (cliente) o como maestro (servidor) como en la Figura 31.

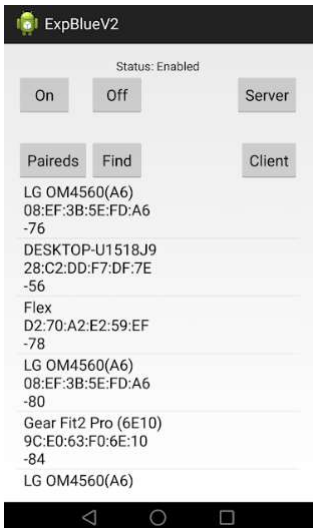

Figura 31 : Layout de aplicación para comunicaciones socket y búsqueda de dispositivos

6.5.9 Calibración de balizas

Una vez conseguida la sincronización de la comunicación con varias balizas y un maestro, y la localización de dispositivos en el área individualmente por parte de cada una de las balizas, es posible comenzar a transferir la información que obtienen las balizas del entorno hacia el dispositivo maestro o servidor.

Para ello, hay que definir primero que tipos de datos se almacenaran durante la búsqueda y finalmente serán transmitidos.

De todos los datos el más irrelevante será el de la distancia del dispositivo hasta la baliza localizadora y hasta ahora solo se ha logrado obtener el valor de la intensidad de la señal recibida en destino por parte del dispositivo localizado.

Para obtener la distancia a partir de la intensidad de la señal recibida se disponen de diversos métodos de cálculo, uno de ellos se ha comentado anteriormente en la expresión (1) donde, a partir de ensayos, se hace posible la obtención de la distancia con respecto a la intensidad de la señal recibida de una forma genérica. Por otro lado se plantea la realización de una calibración que hace posible una discriminación entre dispositivos y obtiene los valores de las distancias de forma más dinámica ya que la intensidad de la señal recibida por el dispositivo destino no será igual para todos los aparatos electrónicos que soporten la comunicación Bluetooth, es decir, si se tienen dos dispositivos distintos a una misma distancia del dispositivo localizador, la intensidad de la señal recibida de cada uno de ellos no siempre tiene por qué ser la misma.

De este modo, y en función de la utilidad que se le vaya a dedicar a esta aplicación de localización de dispositivos Bluetooth, se podrá calcular la distancia de un método más rápido pero generalizado, o por un método más dinámico y ajustado al dispositivo a localizar, pero que por el contrario llevara más tiempo de preparación.

Para esta aplicación se ha decidido optar por el cálculo dinámico de distancias usando la calibración del dispositivo localizador o baliza.

La idea es, con la baliza que se va a calibrar y con un dispositivo de los que se van a localizar se va a proceder a realizar la calibración, se irán tomando medidas de la intensidad de la señal recibida por parte de la baliza a unas distancias conocidas como se representa gráficamente en la Figura 32.

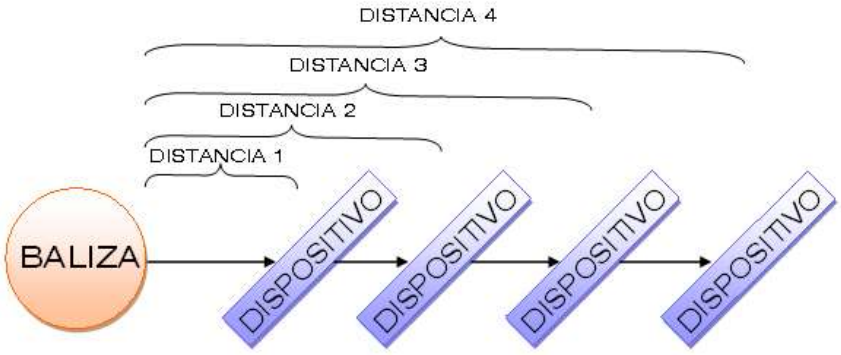

Figura 32 : Modo de operación para llevar a cabo la calibración

Todos los puntos tomados, cuyos parámetros serán la intensidad de la señal recibida y la distancia para esa misma RSSI, se almacenaran en un *array* o vector de forma ordenada según su distancia anotada.

Posteriormente, una vez tomados todos los puntos deseados y habiendo cumplido un mínimo para validar una buena calibración, los próximos valores de intensidad de la señal recibida de los dispositivos que se encuentren en el espacio de búsqueda se obtendrán mediante interpolación a partir de los puntos de calibración tomados con anterioridad. Por tanto se tendría una gráfica de conversión RSSI – Distancia como la mostrada en la Figura 33 para un dispositivo cualquiera.

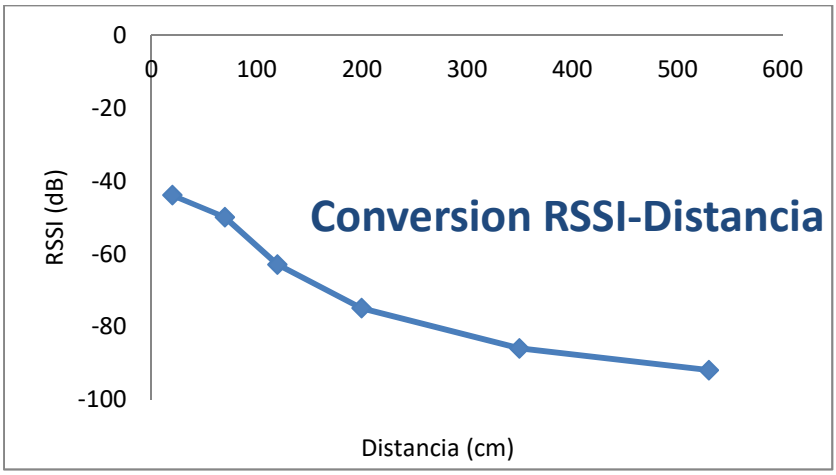

Figura 33 : Curva empírica de conversón RSSI a distancias

De esta forma, entrando con el valor de RSSI del objeto identificado será posible obtener el valor aproximado de la distancia a la que se encuentra el dispositivo de la baliza que lo localizó.

Teniendo ya el concepto y el procedimiento para realizar una calibración y a raíz de esto empezar a toar valores reales de distancia dentro del entorno se puede comenzar a diseñar el entorno grafico que corresponderá a la calibración de las balizas.

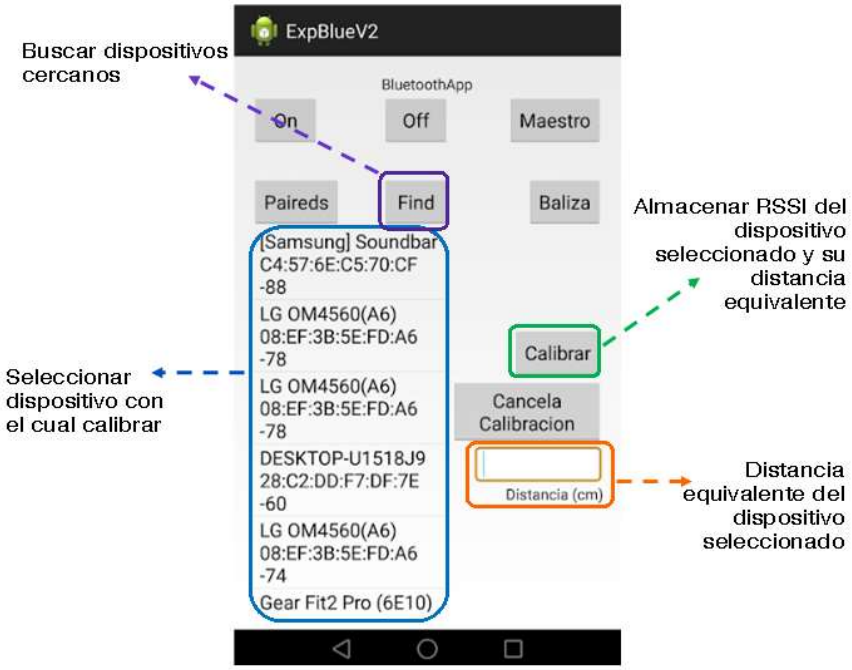

Figura 34 : Configuración de layout de la actividad encargada de realizar la calibración

Como se muestra en Figura 34, para realizar la calibración se necesita de un protocolo concreto, para ello se han implementado nuevos botones y accesorios que facilitaran la tarea del usuario a la hora de calibrar la baliza. El procedimiento seria el siguiente.

1. Asegurarse de que el dispositivo objeto de calibración tendrá las opciones Bluetooth activadas y su estado será visible.

2. Buscar dispositivos cercanos y generar lista de dichos dispositivos, dentro de esta lista generada aparecerá un dispositivo concreto que será con el cual se desea calibrar la baliza.

3. Seleccionar el dispositivo deseado de la lista, con un *click* se almacenara temporalmente el valor de la intensidad de la señal recibida por la baliza.

4. Insertar el valor de la distancia equivalente en la ventana de escritura.

5. Calibrar el dispositivo con ese punto, se almacenaran el valor de la distancia y su RSSI en un mismo elemento del vector de calibración.

6. Volver al paso 2 hasta que la baliza llegue a un mínimo de puntos para aceptar la calibración.

7. En caso de querer eliminar la lista generada y crear una nueva, cancelar calibración.

Con el planteamiento anterior se puede configurar la estructura interna a modo de función dentro de la actividad desarrollada en lenguaje Java para que cumpla con dichas especificaciones y de esta forma obtener buenos resultados. De esta forma se tendría el Fragmento 29.

Finalizada la calibración se ha generado una lista ordenada indicada en este caso como *calibList* la cual contiene elementos de tipo *calibPoint* que contienen la información de la RSSI y su distancia calibrada.

Primero averigua en que intervalo, entre que dos puntos de calibración, se encuentra y toma estos como los valores máximo y mínimo para realizar la interpolación.

Finalmente la función representada en el Fragmento 29 devuelve un entero que representa la distancia a la que se encuentra el dispositivo cuya RSSI se tomó como valor de partida.

```
1   public int convert2Distance(int rssi) {
2
3       int distInt = 0;
4
5       for(int i = 0; i < calibList.getCount() ; ++i) {
6                 if(rssi <= calibList.getItem(0).getRSSI()) {
7                     calibPoint limitMax = new
8   calibPoint(calibList.getItem(0).getDistance(),calibList.getItem(0).getRSSI());
9                     calibPoint limitMin = new
10  calibPoint(calibList.getItem(1).getDistance(),calibList.getItem(1).getRSSI());
11
12  //Interpolation between limitMax and limitMin from calibList
13                    distInt = limitMin.getDistance() + (limitMax.getDistance()-
14                              limitMin.getDistance())/(limitMax.getRSSI()-
15                              limitMin.getRSSI())*(rssi- limitMin.getRSSI());
16                    break;
17                }
18                else if(rssi > calibList.getItem(calibList.getCount()-
19                1).getRSSI()){
20                    calibPoint limitMax = new
21                            calibPoint(calibList.getItem(calibList.getCount()-
22                            1).getDistance(),calibList.getItem(calibList
23                            .getCount()-1).getRSSI());
24
25                    calibPoint limitMin = new
26                            calibPoint(calibList.getItem(calibList.getCount()-
27                            2).getDistance(),calibList.getItem(calibList
28                            .getCount()-2).getRSSI());
29
30  //Interpolation between limitMax and limitMin from calibList
31                    distInt = limitMin.getDistance() + (limitMax.getDistance()-
32                           limitMin.getDistance())/(limitMax.getRSSI()-
33                           limitMin.getRSSI())*(rssi - limitMin.getRSSI());
34                    break;
35                }
36                else if(rssi <= calibList.getItem(i).getRSSI()) {
37                    calibPoint limitMax = new calibPoint(calibList.getItem(i-
38                           1).getDistance(),calibList.getItem(i-1).getRSSI());
```

```
39                  calibPoint limitMin = new
31
40  calibPoint(calibList.getItem(i).getDistance(),calibList
41                          .getItem(i).getRSSI());
42
43  //Interpolation between limitMax and limitMin from calibList
44                  distInt = limitMin.getDistance() + (limitMax.getDistance()-
45                          limitMin.getDistance())/(limitMax.getRSSI()-
46                          limitMin.getRSSI())*(rssi - limitMin.getRSSI());
47                  break;
48              }
49          }
50          return distInt;
51  }
```

Fragmento 29 : Función miembro de la actividad de calibración que se encarará de convertir la RSSI en distancia

La información de la distancia obtenida de cada uno de los dispositivos identificados será la más representativa de los mensajes intercambiados.

6.5.10 Generación y tratamiento de mensajes

Una vez asegurada la conversión de los valores de intensidad de la señal recibida por las balizas en valores de distancia y asegurada la comunicación entre las balizas y el maestro mediante *sockets* de comunicación Bluetooth, se puede dar un paso más y configurar el formato de la transferencia de información para que el receptor comprenda todos y cada uno de los datos que le llegan desde el emisor.

Sea cual sea el método de obtención de posición de los dispositivos, ya sea el indicado en la Figura 2 o en la Figura 3 del planteamiento inicial del problema, los datos esenciales que deberá transmitir cada una de las balizas serán:

- Nombre de dispositivo: Información sobre el nombre del dispositivo a localizar ya sea el nombre definido por el usuario o la dirección de acceso al medio del mismo de carácter más estable y unívoco.

- Distancia de dispositivo a baliza: Valor de la distancia calculada por la baliza a partir de la intensidad de la señal recibida con ayuda del método de calibración.

- Posición de la baliza emisora: Información sobre la posición en el plano XY de la baliza que mando el mensaje al maestro para que este pueda realizar correctamente el algoritmo de triangulación.

Toda esta información debería ser suficiente para que el maestro realice correctamente sus operaciones y determine la posición del dispositivo que se encuentra dentro del área de búsqueda de las balizas.

No es posible enviar toda esta información en bloque de una forma arbitraria dado que los datos anteriores forman un conjunto de caracteres y números como es el caso de las direcciones MAC.

Para diferenciar unos datos de otros se insertara un carácter especial antes de cada uno de ellos para que el destinatario comprenda el mensaje, de esta forma el maestro leerá uno de los comandos especiales y ya sabrá qué tipo de dato forman los caracteres siguientes.

Como se puede observar en el Fragmento 30, se escribirán cuatro tipos de caracteres especiales:

- '/': Nuevo dispositivo, este comando indicara al maestro la lectura de un nuevo dispositivo, de esta forma, si la baliza envía información acerca de más de un dispositivo detectado, el maestro almacenara la información siguiente en un nuevo elemento de la lista o no.

- '%': Posición de la baliza emisora, este dato indica que el dato que viene a continuación será la posición en el eje X del plano de búsqueda en el que se encuentra ubicada la baliza.

- '$': Dirección MAC, este caracter indica al maestro que los siguientes caracteres que extraiga del mensaje formaran la dirección de acceso al medio del dispositivo que se ha detectado.

- '&': Distancia del objetivo, este caracter informará al maestro la distancia a la que se identificó el dispositivo con la dirección MAC ya definida por parte de la baliza emisora del mensaje.

Por tanto, la estructura de la información de cada dispositivo dentro del mensaje será como la indicada a continuación:

/%POSICION_BALIZA$MAC_DISPOSITIVO&DISTANCIA_DISPOSITIVO_A_BALIZA/…

Creación del mensaje

Lo primero de todo será crear el mensaje, y para ello la baliza actualizara la información contenida en el mensaje cada vez que detecte un nuevo dispositivo.

Por tanto, en el BroadcastReceiver que se creó en el Fragmento 17 se añadirá una nueva cadena de caracteres al mensaje en el momento que se detecte un dispositivo nuevo ya la baliza este totalmente calibrada, ya que de no estarlo habría problemas a la hora de intentar traducir la intensidad de la señal recibida, quedando de la siguiente forma, como se indica en el Fragmento 30.

```
1  if(isCalibrated) {
2          dist = convert2Distance(rssi);
3          message = message + "%" + posX + "$" + device.getAddress() + "&" +
4                  dist + "/";
5  }
```

Fragmento 30 : Escritura del mensaje por parte del emisor y localizador de dispositivos

El mensaje completamente generado, en formato *String*, se enviara al maestro en forma de cadena de bytes y este lo traducirá de nuevo un String como se indicó anteriormente.

Lectura del mensaje

Una vez que el gestor de tráfico en los *sockets* del maestro ha recibido el mensaje y lo ha convertido en una cadena con el formato que se indicó al inicio del apartado es tarea del maestro traducirlo y almacenar la información de forma organizada dentro de una matriz o de un vector para posteriormente hacer el tratamiento de forma correcta y sin problemas.

Con la llegada de un caracter '/' el maestro detectara que se trata de un nuevo dispositivo, posteriormente leerá el caracter '%' que le dirá en qué posición del eje X se encontrará la baliza que le ha enviado el mensaje, el siguiente carácter especial que leerá será '$' y por tanto el próximo dato que vaya a extraer será la dirección MAC del dispositivo, y por ultimo recibirá la distancia calculada por la baliza de forma posterior al carácter '&'.

Finalmente recibirá de nuevo el carácter '/' que le indicara el final de la información de un dispositivo y la llegada de la información de uno nuevo, en este momento comprobara si el dispositivo ya se había localizado y actualizará el valor de la distancia obtenida, en caso contrario introducirá en la lista de dispositivos localizados un nuevo objeto con los datos de posición de la baliza, dirección de acceso al medio del dispositivo y la correspondiente distancia.

Toda la explicación del procedimiento anteriormente descrita de detalla de forma programada en Java en el Fragmento 31.

```
1   //MESSAGE MANAGE AND STORAGE FUNCTION
2   //IF THE IMPUT MESSAGE CONTAINS OR BEGINS WITH '/' READS THE COMPLETE MESSAGE
3   //IF THE CHARACTER IS '%' MEANS THAT THE NEXT INFORMATION IS THE NAME OF THE
4   BEACON TRANSMITTER
5   //IF THE CHARACTER IS '$' MEANS THAT THE NEXT INFORMATION IS THE ADDRESS OF
6   THE FOUND DEVICE
7   //IF THE CHARACTER IS '&' MEANS THAT THE NEXT INFORMATION IS THE DISTANCE OF
8   THE FOUND DEVICE
9   //IF THE CHARACTER IS '/' MEANS THAT THE NODE INFORMATION (BEACON, DEVICE
10  ADDRESS, DISTANCE) IS READY TO STORE IN FINALDEVICELIST
11
12  public void extractInfo(String msg) {
13
14        String tempBeacon = "";
15        String tempAddress = "";
16        String tempDist = "";
17        int typeData = 0;
18        int devicePointer;
19        boolean deviceHere;
20
21        if(msg.contains("/")) {
22              for (int i = 0; i < msg.length(); ++i ) {
23                    if( msg.charAt (i) == '%')
24                          typeData = 1;
25                    else if( msg.charAt (i) == '$')
26                          typeData = 2;
27                    else if( msg.charAt (i) == '&')
28                          typeData = 3;
```

```
29                      else if( msg.charAt (i) == '/')
30                            typeData = 4;
31
32                      if(typeData == 1 & msg.charAt (i) != '%')
33                            tempBeacon = tempBeacon + msg.charAt (i);
34                      else if(typeData == 2 & msg.charAt (i) != '$')
35                            tempAddress = tempAddress + msg.charAt (i);
36                      else if(typeData == 3 & msg.charAt (i) != '&')
37                            tempDist = tempDist + msg.charAt (i);
38
39                      if(msg.charAt (i) == '/' & i != 0) {
31                            devicePointer = 0;
40                            deviceHere = false;
41                            for(int m = 0; m < finalDevArray.getCount() ; ++m){
42                                  if(Integer.parseInt(tempBeacon) ==
43                                     finalDevArray.getItem(m).getOrigin() &
44                                     tempAddress.equals(finalDevArray
45                                     .getItem(m).getMac())) {
46
47                                        deviceHere = true;
48                                        devicePointer = m;
49                                  }
50                             }
51
52                             tempBeacon.trim();
53                             tempAddress.trim();
54                             tempDist.trim();
55
56                             if(!deviceHere )
57                                finalDevArray.add(new
58                                BeaconData(Integer.parseInt(tempBeacon),
59                                tempAddress,Integer.parseInt(tempDist)));
60                             if(deviceHere)
61                                finalDevArray.getItem(devicePointer).
62                                setSig(Integer.parseInt(tempDist));
63
64                             tempBeacon = "";
65                             tempAddress = "";
66                             tempDist = "";
67                             typeData = 0;
68                      }
69                }
70          }
71    }
```

Fragmento 31 : Función que se emplea en el dispositivo maestro para traducir los mensajes recibidos desde balizas

6.5.11 Sincronización servidor-multicliente, temporizadores

En lo respectivo a la sincronización de la comunicación entre el maestro y todas las posibles balizas será necesario fijar un protocolo de acceso al medio y buscar una alternativa a la operación del maestro y de las balizas para evitar que cualquiera de ellos se quede bloqueado en alguna operación, como por ejemplo, es posible que el maestro de quede esperando a

recibir una conexión por parte de un cliente o baliza cuando podría estar realizando alguna operación de cálculo de posición.

Para evitar este tipo de problemas se planteara un protocolo de acceso al medio de tipo aleatorio sin escucha del medio, que provocara una contienda pura, donde el primer cliente será el que tenga el privilegio de ser aceptado.

Las balizas realizaran un ciclo de operación de forma indefinida hasta que el usuario solicite la parada de las mismas, primeramente buscara dispositivos en el área de búsqueda durante un determinado tiempo y, cumplido este tiempo intentaran de mandar el mensaje al maestro, en caso de no ser aceptada la petición de acceso porque el maestro este realizando otras operaciones, no se quedan bloqueadas las balizas y vuelven a repetir el ciclo para mantener la información siempre actualizada.

El maestro tendrá otro ciclo similar que evitara sus bloqueos, de tal forma que, cuando acepte una petición de conexión Bluetooth, recibirá el mensaje por parte de la baliza y realizara las tareas encomendadas eh un determinado marco temporal, el problema podría venir en el caso de que no haya ninguna baliza que quiera comunicarse con él en una franja temporal grande, por tanto, el mismo marco temporal que se le fija como límite de realización de operaciones se fijara para su ciclo completo, de esta forma, si no establece comunicación en un ciclo por cualquier motivo, este se vuelve a reiniciar de forma indefinida hasta que el usuario solicite una cancelación del maestro.

Para conseguir lo mencionado en los dos párrafos anteriores se presenta la opción de insertar temporizadores asíncronos, de tal forma que, cada tiempo de marco se realizan operaciones y cuando termina el ciclo completo se puede reiniciar el ciclo o finalizar la tarea como baliza o maestro.

Los temporizadores tienen la siguiente estructura mostrada en el Fragmento 32.

```
1   final CountDownTimer Timer = new CountDownTimer(cycle, tick) {
2
3           public void onTick() {
4               //Activate when tick time finishes
5           }
6
7           public void onFinish() {
8               //Activate when cycle time is completed
9           }
10  };
```

Fragmento 32 : Estructura de temporizador importado de biblioteca Android

Por tanto, y como se indicó en el planteamiento se control de los dispositivos baliza y maestro se podrían tener unos fragmentos como los mostrados a continuación.

Por parte del cliente o baliza, cada fin de marco se reiniciara la búsqueda y se intentará enviar la información al cliente

```
1   final CountDownTimer clientTimer = new CountDownTimer(11000, 2000) {
2
3           public void onTick(long millis_to_end) {
```

```
4                 find();
5
6                 if(message.length()>5) {
7
8                     ConnectThread conecta = new
9                             ConnectThread(pairedList.getItem(indexMaster));
10                    if(mBluetoothSocket!=null) {
11                            conecta.run();
12                            conecta.cancel();
13                    }
14
15                 }
16
17        }
18
19        public void onFinish() {
20                      this.start();
21        }
22  };
```

Fragmento 33 : Temporizador correspondiente al ciclo del cliente o baliza

Cuando se cumpla el tiempo de marco, el servidor inicia la búsqueda de nuevos dispositivos y tiene 2000 milisegundos (tiempo de marco) para obtener resultados y generar un mensaje de dimensión superior a 5 caracteres, si es menos se entenderá que no ha detectado ningún dispositivo. Si el tamaño del mensaje es superior a 5 caracteres entonces intentara conectar con el maestro que estará almacenado en la lista de elementos Bluetooth emparejados con un índice definido.

En caso de no conectar con el maestro, se acabará cumpliendo el tiempo de marco y reiniciará la acción.

Finalmente, cuando se finaliza el tiempo e ciclo, e este caso de 11000 milisegundos, se reiniciará el ciclo de nuevo.

En el caso del maestro el funcionamiento del temporizador es el mismo solo que las acciones por marco serán diferentes.

```
1   final CountDownTimer serverTimer = new CountDownTimer(11000, 500) {
2
3         public void onTick(long millis_to_end) {
4
5                 AcceptThread accept = new AcceptThread("miServidor");
6                 accept.run();
7
8                 extractMatrix(message);
9
10                BTArrayAdapter1.clear();
11                for(int i = 0 ; i < finalDevArray.getCount() ; ++i) {
12
13                      String cad1 = finalDevArray.getItem(i).getOrigin() + " " +
14  finalDevArray.getItem(i).getMac() + " " + finalDevArray.getItem(i).getSig();
15                      BTArrayAdapter1.add(cad1);
16                      BTArrayAdapter1.notifyDataSetChanged();
17
18                      BeaconData tempDev = new
19  BeaconData(finalDevArray.getItem(i).getOrigin(),
```

```
20  finalDevArray.getItem(i).getMac(), finalDevArray.getItem(i).getSig());
21                      if(finalDevArray.getCount() > 1)
22
23                          //Calculate the device position
24
25                      }
26          }
27          public void onFinish() {
28               this.start();
29          }
30  };
```

Fragmento 34 : Temporizador correspondiente al ciclo del servidor o maestro

Cando se cumpla el tiempo de marco de 500 milisegundos, más que suficiente para realizar las operaciones y realizar una conexión con cualquier cliente que lo solicite, el maestro se mantendrá a la escucha de peticiones de conexiones entrantes, cuando acepte una de estas solicitudes leerá el mensaje recibido y lo almacenará en la estructura definida anteriormente y lo mostrara por pantalla en una lista grafica en la interfaz.

Si el tamaño de la lista generada con la información recibida de las balizas es mínimo de dos elementos entonces se puede intentar calcular la posición del dispositivo.

Cuando finaliza el tiempo total de ciclo, también de 11 segundos como en el caso del cliente, se reinicia el temporizador del maestro.

6.5.12 Localización de dispositivos por triangulación

Una vez definidas todas y cada una de las herramientas y soluciones adoptadas para la resolución del problema en la localización solo resta obtener la ubicación concreta dentro del plano de búsqueda definido.

Hasta el momento se tiene:

- Identificación de dispositivos en rango de búsqueda de las balizas.
- Obtención de la distancia de los dispositivos a partir de la intensidad de la señal recibida por parte de las balizas.
- Comunicación entre balizas (clientes) y maestro (servidor) por acceso al medio aleatorio.
- Sincronización de operaciones con temporizadores.

A partir de la traducción de mensaje enviado por la baliza por parte del maestro se ha obtenido una lista o vector de elementos con datos específicos y de utilidad para realizar las operaciones y cálculos correspondientes a la obtención de la posición en el plano XY.

Tras la traducción del mensaje se tendría un conjunto similar al mostrado en la Figura 35.

OBJETO1	OBJETO2	OBJETO3
POSICION BALIZA	POSICION BALIZA	POSICION BALIZA
0	150	0
DISPOSITIVO	DISPOSITIVO	DISPOSITIVO
2A:33:32:9F	2A:DD:11:01	2A:DD:11:01
DISTANCIA	DISTANCIA	DISTANCIA
228	184	72

Figura 35 : Vector de objetos de la clase *BeaconData* creada en la actividad maestro

A partir de esta información será posible obtener una posición concreta del dispositivo a localizar. Para poder comprobar si es posible comenzar a hacer cálculos será necesario por lo menos tener dos objetos dentro del vector mencionado.

Para poder ubicar el dispositivo detectado será necesario tener por lo menos dos elementos de la lista con el mismo nombre de dispositivo y con distinta posición de baliza, lo que indicara que el dispositivo ha sido detectado por dos balizas diferentes y la información le ha sido enviada al maestro.

Una vez detectados los objetos correspondientes al dispositivo detectado por ambas balizas es posible proceder al cálculo de la posición.

Anteriormente, el cálculo de la posición por ecuaciones trigonométricas se propuso como una solución.

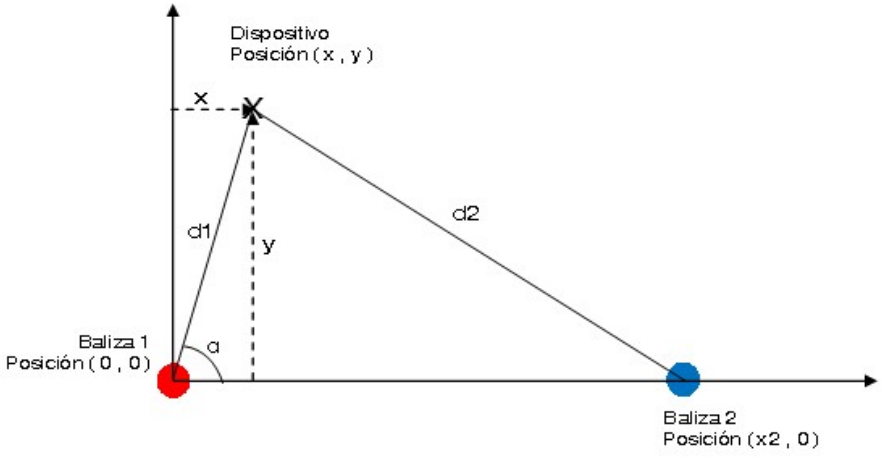

Figura 36 : Obtención de punto de posición con dos balizas dentro de un plano definido

De esta forma se obtiene un triángulo formado por ‘d1’, ‘d2’ y ‘x2’, donde las incógnitas a calcular serán ‘x’ e ’y’.

Para la obtención de dichos parámetros se parte del cálculo del semiperímetro.

$$s = \frac{d_1 + d_2 + x_2}{2} \qquad (3)$$

También se conoce que, el área de un triángulo viene definida por la siguiente expresión.

$$\boldsymbol{A = \frac{x_2 \cdot y}{2}} \qquad (4)$$

Y por otro lado, el área también es posible de obtener con el semiperímetro conocido siguiendo la expresión (5).

$$\boldsymbol{A = \sqrt{s \cdot (s - d_1) \cdot (s - d_2) \cdot (s - x_2)}} \qquad (5)$$

Por tanto, sustituyendo la expresión (5) en la expresión (4), y despejando el parámetro de la altura, o en este caso el parámetro *'y'*, se puede obtener dicho valor.

$$\boldsymbol{y = \frac{2}{x_2} \cdot \sqrt{s \cdot (s - d_1) \cdot (s - d_2) \cdot (s - x_2)}} \qquad (6)$$

Ahora solo quedaría extraer el valor de la posición 'x' donde se encuentra el dispositivo a localizar, para ello, simplemente se tendría un triángulo rectángulo, como el mostrado en la Figura 37 una vez obtenida la coordenada 'y' del punto.

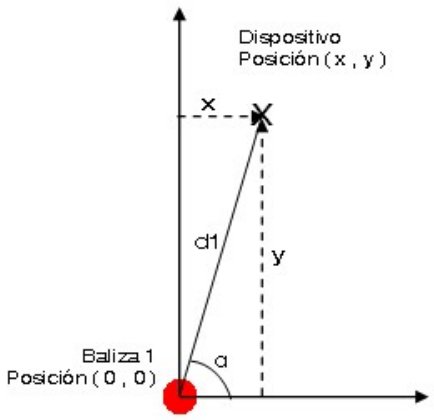

Figura 37 : Cálculo del parámetro x conociendo el parámetro y

Con el uso de las ecuaciones trigonométricas es posible obtener el valor del ángulo 'α' que vendrá determinado por la expresión (7).

$$\boldsymbol{\alpha = arcsen(\frac{y}{d_1})} \qquad (7)$$

Y a partir del valor conocido del ángulo formado por los lados 'd1' y 'x' es posible obtener el valor de la coordenada 'x' con la ecuación del coseno.

$$\boldsymbol{x = d_1 \cdot \cos(\alpha)} \qquad (8)$$

Los parámetros obtenidos representaran las coordenadas 'x' e 'y' del punto de ubicación del dispositivo localizado por las dos balizas. Está el inconveniente de que, como el cálculo de la distancia por la baliza no es del todo preciso, este cálculo de las coordenadas de localización del dispositivo no siempre vayan a dar los valores reales, para ello se plantean variantes a los casos particulares.

En caso de que la suma de las distancias obtenidas por las balizas sea inferior a la distancia entre balizas, la coordenada 'y' se puede interpretar como valor cero, y la coordenada 'x' será la distancia de la baliza situada en el origen.

Con todo esto, se creara una función dentro de la actividad correspondiente al maestro que se encargará de obtener las coordenadas a partir de los datos de distancia entre balizas y distancias de balizas a dispositivo localizado, como se muestra en el siguiente fragmento.

```
1  public void obtainCordinates(String devMAC, int beaconsDist, int rad1, int
2  rad2){
3
4          long x = 0;
5          long y = 0;
6
7          if(rad1 + rad2 <= beaconsDist)
8              locationList.add(new Coordinates(devMAC, rad1, 0));
9          else {
10             double tempSemiperiod = 0;
11             double tempArea = 0;
12
13             tempSemiperiod = (rad1 + rad2 + beaconsDist)/2;
14             tempArea = Math.sqrt(Math.abs(tempSemiperiod*(tempSemiperiod –
15                  rad1)*(tempSemiperiod - rad2)*(tempSemiperiod - beaconsDist)));
16
17             y = Math.round(tempArea*2/beaconsDist);
18
19             x = Math.round(rad1 * Math.cos(Math.asin(y/rad1)));
20
21             if(x>0 & !locationList.isEmpty()) {
22                 for(int i = 0; i < locationList.getCount() ; ++i) {
23                     if(locationList.getItem(i).getMac().equals(devMAC)) {
24
25                         BTArrayAdapter2.remove(BTArrayAdapter2.getItem(i));
26                         BTArrayAdapter2.add(devMAC + " (" + x + "," + y + ")");
27                         BTArrayAdapter2.notifyDataSetChanged();
28                         locationList.remove(locationList.getItem(i));
29                         locationList.add(new Coordinates(devMAC, x, y));
30
31                         break;
32                     }
34                     if(i == locationList.getCount()-1) {
35                         BTArrayAdapter2.add(devMAC + " (" + x + "," + y + ")");
36                         BTArrayAdapter2.notifyDataSetChanged();
37
38                         locationList.add(new Coordinates(devMAC, x, y));
39                     }
40                 }
41             }else if(x>0 & locationList.isEmpty()) {
42                 BTArrayAdapter2.add(devMAC + " (" + x + "," + y + ")");
43                 BTArrayAdapter2.notifyDataSetChanged();
44
45                 locationList.add(new Coordinates(devMAC, x, y));
46             }
47         }
48 }
```

Fragmento 35 : Obtención de coordenadas partiendo del vector generado con los mensajes

Los puntos obtenidos, con las coordenadas ya calculadas con el Fragmento 35, se irán almacenando en una lista que a su vez saldrá por pantalla en interfaz de usuario y servirá como patrón para solo actualizar las coordenadas y no el nombre del dispositivo y de esta forma se hace dinámica y no lo añade de nuevo aumentando las dimensiones de la lista.

6.5.13 Funcionamiento de la App y resultados obtenidos

Para finalizar el desarrollo de la aplicación Android de localización de dispositivos Bluetooth, mencionar todos y cada uno de los puntos y pasos importantes que se han dado para lograr el objetivo:

1. Gestión del adaptador Bluetooth propio de los dispositivos activos del sistema de localización Bluetooth.

2. Identificación de dispositivos Bluetooth en zona de búsqueda.

3. Obtención de intensidad de la señal recibida y transformación a magnitud de longitud.

4. Transferencia de mensajes, contenedores de información sobre los dispositivos encontrados, entre balizas y maestro.

5. Calculo de la posición dentro del plano de los dispositivos descubiertos por las balizas.

De esta forma quedan definidas las pantallas de la aplicación de tal forma que queda una pantalla principal donde poder activar y desactivar las comunicaciones Bluetooth, y pasar a cualquier modo de funcionamiento.

Figura 38 : Layout de actividad principal, *Initial*

Se tienen un modo de funcionamiento sin conexión para buscar dispositivos y obtener su correspondiente RSSI sin transmitir ni recibir información de otro usuario.

Figura 39 : Layout de actividad modo sin conexión, *Single*

También está el modo baliza desde el cual se puede calibrar la baliza como se comentó anteriormente, seleccionar la posición de la baliza en el plano para informar al maestro y seleccionar a quien será el maestro del sistema.

Figura 40 : Layout de actividad modo baliza, *Beacon*

Por ultimo esta la actividad maestro, que únicamente muestra la cola de mensajes recibidos desde las balizas que se hayan definido y la lista de dispositivos localizados donde se mostrará la dirección de acceso al medio del dispositivo y su posición en coordenadas x e y del plano.

Figura 41 : Layout de actividad modo maestro, *Master*

Para definir el funcionamiento a nivel de usuario de la aplicación realizada se expone en la Figura 42.

En el diagrama se puede observar con algo de detalle el desplazamiento entre ventanas o actividades dentro de la aplicación, como es posible navegar dentro de la aplicación y con qué condiciones.

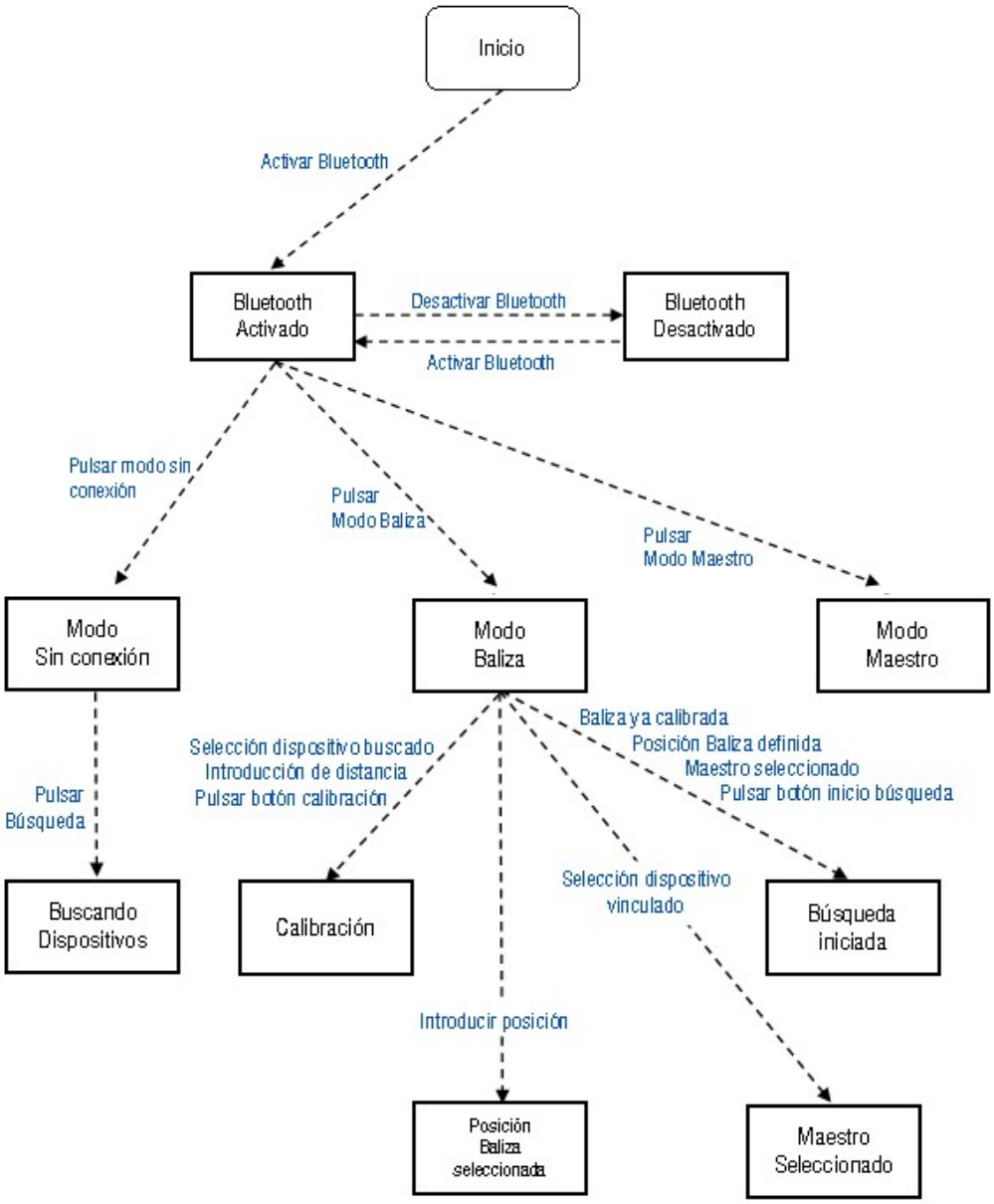

Figura 42 : Diagrama de funcionamiento del programa completo

7 Conclusiones

7.1 Conclusiones

Se ha conseguido desarrollar un sistema de localización de dispositivos que se encuentren utilizando los servicios de comunicación Bluetooth y además se encuentren en estado visible para el resto de usuarios conectados con esta tecnología.

Se ha creado una aplicación en Android capaz de actuar tanto como baliza, buscando dispositivos, como maestro, recopilando la información de las balizas y procesando la misma para calcular y mostrar un resultado, en forma de coordenadas X e Y, en la interfaz de usuario y así la persona que se encuentre usando esta aplicación, y con el dispositivo maestro en la mano, sepa donde se encuentran los dispositivos Bluetooth conociendo las ubicaciones mostradas.

Cada dispositivo Bluetooth emitirá la señal al exterior con una determinada fuerza dado que no todos los dispositivos hardware Bluetooth están diseñados de la misma forma y por tanto, al calibrar las balizas con un único dispositivo de referencia se producirá un error en la conversión de distancias para otros dispositivos con una configuración de hardware distinta. Esto se puede traducir en que, este sistema de localización tendría una buena aplicación para la localización de dispositivos con características de comunicación Bluetooth homogéneas.

En cuanto al rango de trabajo, este sistema está diseñado para espacios cerrados que no superen los 8 metros de longitud máxima, dado que la mayoría de dispositivos Android disponen de un sistema Bluetooth de categoría 2, lo que indica que su distancia máxima de detección será de 10 metros en el mejor de los casos. En caso de querer trabajar en espacios mayores sería necesaria la inclusión de nuevas balizas o trabajar con dispositivos Bluetooth de categoría 3 que podrían alcanzar hasta 100 metros de detección, poco comunes en dispositivos Android.

8 Bibliografía

8.1 Enlaces web consultados

General y fundamentos del proyecto por orden de importancia

https://developer.android.com/ndk/guides/concepts.html

https://labs.beeva.com/posici%C3%B3n-y-contexto-con-beacons-15e662cfacf2

https://www.youtube.com/watch?v=iCVs6BYn_DM

https://movilfacil.wordpress.com/2011/03/25/cap-8-posicionamiento-localizacion/

https://techlandia.com/programacion-c-android-info_232366/

https://geekytheory.com/que-es-el-android-ndk-parte-1

http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo3.pdf

Detección de dispositivos y su información por orden de importancia

https://solidgeargroup.com/detectar-dispositivos-bluetooth-android?lang=es

https://es.stackoverflow.com/questions/4447/leer-datos-desde-dispositivo-bluetooth

https://www.hell-desk.com/comunicar-una-app-de-android-con-un-arduino-utilizando-la-api-de-bluetooth-24/

http://maneldeantonio.com/dbs-dbm-rssi

Comunicación con sockets por orden de importancia

https://danielggarcia.wordpress.com/2013/10/28/bluetooth-vi-creando-el-hilo-cliente/

https://www.programcreek.com/java-api-examples/index.php?api=

android.bluetooth.BluetoothSocket

https://code.tutsplus.com/es/tutorials/create-a-bluetooth-scanner-with-androids-bluetooth-api--cms-24084

https://es.stackoverflow.com/questions/149536/como-obtener-el-uuid-de-bluetooth-en-un-dispositivo-android

https://www.matec-conferences.org/articles/matecconf/pdf/2017/46/matecconf_dts2017_04025.pdf

Páginas web para instalar herramientas necesarias

https://dl-ssl.google.com/android/eclipse/

https://eclipse.org/downloads/

http://oracle.com/technetwork/es/java/javase/downloads/index.html

9 Anexos

9.1 Código desarrollado

9.1.1 Manifest

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="com.ebtlocator"
4       android:versionCode="1"
5       android:versionName="1.0" >
6
7       <uses-sdk
8           android:minSdkVersion="14"
9           android:targetSdkVersion="14" />
10
11      <uses-permission android:name="android.permission.BLUETOOTH" />
12      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
13      <uses-permission
14  android:name="android.permission.ACCESS_COARSE_LOCATION" />
15      <uses-permission
16  android:name="android.permission.ACCESS_FINE_LOCATION" />
17
18      <application
19          android:allowBackup="true"
20          android:icon="@drawable/ic_launcher"
21          android:label="@string/app_name"
22          android:theme="@style/AppTheme" >
23          <activity
24              android:name=".Initial"
25              android:label="@string/app_name" >
26              <intent-filter>
27                  <action android:name="android.intent.action.MAIN" />
28                  <categoryandroid:name="android.intent.category.LAUNCHER"/>
29              </intent-filter>
30          </activity>
31          <activity
32              android:name=".Calibration"
33              android:label="@string/title_activity_calibration" >
34          </activity>
35          <activity
36              android:name=".Master"
37              android:label="@string/title_activity_master" >
```

```
38          </activity>
39          <activity
31              android:name=".Single"
40              android:label="@string/title_activity_single" >
41          </activity>
42          <activity
43              android:name=".Beacon"
44              android:label="@string/title_activity_beacon" >
45          </activity>
46      </application>
47
48  </manifest>
49
50
```

Fragmento 36 : Manifest de aplicación

9.1.2 Actividad Initial layout

```
1   <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent"
6       android:paddingBottom="@dimen/activity_vertical_margin"
7       android:paddingLeft="@dimen/activity_horizontal_margin"
8       android:paddingRight="@dimen/activity_horizontal_margin"
9       android:paddingTop="@dimen/activity_vertical_margin"
10      android:background="@android:color/black"
11      tools:context="com.ebtlocator.Initial" >
12
13      <Button
14          android:id="@+id/turnOff"
15          android:layout_width="220dp"
16          android:layout_height="wrap_content"
17          android:layout_alignParentBottom="true"
18          android:layout_centerHorizontal="true"
19          android:layout_marginBottom="15dp"
20          android:background="@android:color/holo_blue_light"
21          android:text="DESACTIVAR BLUETOOTH" />
22
23      <Button
24          android:id="@+id/beacon"
25          android:layout_width="220dp"
26          android:layout_height="wrap_content"
27          android:layout_above="@+id/master"
28          android:layout_centerHorizontal="true"
29          android:layout_marginBottom="16dp"
30          android:background="@android:color/holo_blue_light"
31          android:text="MODO BALIZA" />
32
33      <Button
34          android:id="@+id/single"
35          android:layout_width="220dp"
36          android:layout_height="wrap_content"
37          android:layout_above="@+id/turnOff"
```

```
38          android:layout_alignRight="@+id/turnOn"
39          android:layout_marginBottom="142dp"
31          android:background="@android:color/holo_blue_light"
40          android:text="MODO SIN CONEXION" />
41
42      <Button
43          android:id="@+id/master"
44          android:layout_width="220dp"
45          android:layout_height="wrap_content"
46          android:layout_above="@+id/single"
47          android:layout_centerHorizontal="true"
48          android:layout_marginBottom="15dp"
49          android:background="@android:color/holo_blue_light"
50          android:text="MODO MAESTRO" />
51
52      <Button
53          android:id="@+id/turnOn"
54          android:layout_width="220dp"
55          android:layout_height="wrap_content"
56          android:layout_above="@+id/turnOff"
57          android:layout_centerHorizontal="true"
58          android:layout_marginBottom="18dp"
59          android:background="@android:color/holo_blue_light"
60          android:text="ACTIVAR BLUETOOTH" />
61
62  </RelativeLayout>
```

Fragmento 37 : Programa Layout de actividad *Initial*

9.1.3 Actividad Initial programa Java

```
1   package com.ebtlocator;
2
3
4   import com.ebtlocator.Beacon;
5   import com.ebtlocator.Master;
6   import com.ebtlocator.Single;
7
8   import android.app.Activity;
9   import android.bluetooth.BluetoothAdapter;
10  import android.content.Intent;
11  import android.os.Bundle;
12  import android.view.View;
13  import android.view.View.OnClickListener;
14  import android.widget.Button;
15  import android.widget.Toast;
16
17  public class Initial extends Activity {
18
19      private static final int REQUEST_ENABLE_BT = 1;
20      private Button onBtn;          //Turn on bluetooth button
21      private Button offBtn;         //Show available bluetooth device in
22                                       area bluetooth button
23      private Button serverBtn;      //Turn off master mode button
24      private Button beaconBtn;      //Turn on beacon mode button
25      private Button singleBtn;      //List accessory of interface
```

```
26        private BluetoothAdapter myBluetoothAdapter;
27
28              @Override
29              protected void onCreate(Bundle savedInstanceState) {
30                 super.onCreate(savedInstanceState);
31                 setContentView(R.layout.activity_initial);
32
33                 myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
34                 if(myBluetoothAdapter == null) {
35                     onBtn.setEnabled(false);
36                     offBtn.setEnabled(false);
37                     serverBtn.setEnabled(false);
38                     beaconBtn.setEnabled(false);
39                     singleBtn.setEnabled(false);
31                     Toast.makeText(getApplicationContext(),"Your device
40                      does not support Bluetooth",Toast.LENGTH_LONG).show();
41                 } else {
42
43                     onBtn = (Button)findViewById(R.id.turnOn);
44                     onBtn.setOnClickListener(new OnClickListener() {
45
46                       @Override
47                       public void onClick(View v) {
48                           on(v);
49                       }
50                     });
51
52                     offBtn = (Button)findViewById(R.id.turnOff);
53                     offBtn.setOnClickListener(new OnClickListener() {
54
55                       @Override
56                       public void onClick(View v) {
57                           off(v);
58                       }
59                     });
60
61                      beaconBtn = (Button)findViewById(R.id.beacon);
62                      beaconBtn.setOnClickListener(new OnClickListener() {
63
64                             @Override
65                             public void onClick(View v) {
66                              if (myBluetoothAdapter.isEnabled()) {
67                                     Intent mi_intento = new
68                                         Intent(Initial.this,Beacon.class);
69                                     startActivity(mi_intento);
70                              }else
71                                     Toast.makeText(getApplicationContext
72                                           (),"ACTIVAR BLUETOOTH" ,
73                                           Toast.LENGTH_LONG).show();
74                             }
75                           });
76
77                      serverBtn = (Button)findViewById(R.id.master);
78                      serverBtn.setOnClickListener(new OnClickListener() {
79
80                             @Override
81                             public void onClick(View v) {
82                              if (myBluetoothAdapter.isEnabled()) {
83                                     Intent mi_intento = new
```

```
84                                      Intent(Initial.this,Master.class);
85                                  startActivity(mi_intento);
86                               }else
87                                      Toast.makeText(getApplicationContext
88                                            (),"ACTIVAR BLUETOOTH" ,
89                                            Toast.LENGTH_LONG).show();
90                               }
91                           });
92
93                      singleBtn = (Button)findViewById(R.id.single);
94                      singleBtn.setOnClickListener(new OnClickListener() {
95
96                              @Override
97                              public void onClick(View v) {
98                               if (myBluetoothAdapter.isEnabled()) {
99                                      Intent mi_intento = new
100                                         Intent(Initial.this,Single.class);
101                                     startActivity(mi_intento);
102                              }else
103                                  Toast.makeText(getApplicationContext(),"
104                                       ACTIVAR BLUETOOTH" ,
105                                       Toast.LENGTH_LONG).show();
106                              }
107
108                          });
109             }
110         }
111
112 ////////////////////////////////////////////////////////////////////////
113 ////////////////////////////////////
114 //              CLASS FUNCTIONS
115 ////////////////////////////////////////////////////////////////////////
116 ////////////////////////////////////
117
118
119         //TURN ON BLUETOOTH FUNCTION
120         //ENABLE THE BLUETOOTH ADAPTER OF DEVICE, REQUIRES MANUAL
121 ACTION
122
123     public void on(View view){
124         if (!myBluetoothAdapter.isEnabled()) {
125            Intent turnOnIntent = new
126                Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
127            startActivityForResult(turnOnIntent, REQUEST_ENABLE_BT);
128
129            Toast.makeText(getApplicationContext(),"BLUETOOTH ACTIVADO"
130                                          , Toast.LENGTH_LONG).show();
131          }
132          else{
133            Toast.makeText(getApplicationContext(),"BLUETOOTH YA SE
134                        ENCUENTRA ACTIVADO",Toast.LENGTH_LONG).show();
135          }
136     }
137
138 //TURN OFF BLUETOOTH FUNCTION
139 //DISABLE THE BLUETOOTH ADAPTER OF DEVICE
140
141     public void off(View view){
142         myBluetoothAdapter.disable();
```

```
143
144             Toast.makeText(getApplicationContext(),"BLUETOOTH DESACTIVADO",
145                             Toast.LENGTH_LONG).show();
146         }
147
148     }
```

Fragmento 38 : Programa Java de actividad *Initial*

9.1.4 Actividad Single layout

```
1   <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="wrap_content"
5       android:layout_height="wrap_content"
6       android:background="@android:color/black"
7       android:paddingBottom="@dimen/activity_vertical_margin"
8       android:paddingLeft="@dimen/activity_horizontal_margin"
9       android:paddingRight="@dimen/activity_horizontal_margin"
10      android:paddingTop="@dimen/activity_vertical_margin"
11      tools:context="com.ebtlocator.Single" >
12
13      <ListView
14          android:id="@+id/listView1"
15          android:layout_width="match_parent"
16          android:layout_height="wrap_content"
17          android:layout_above="@+id/search"
18          android:background="@android:color/darker_gray"
19          android:layout_centerHorizontal="true" >
20
21      </ListView>
22
23      <Button
24          android:id="@+id/search"
25          android:layout_width="match_parent"
26          android:layout_height="wrap_content"
27          android:layout_alignParentBottom="true"
28          android:layout_centerHorizontal="true"
29          android:background="@android:color/holo_blue_light"
30          android:text="Find" />
31
32  </RelativeLayout>
```

Fragmento 39 : Programa Layout de actividad *Single*

9.1.5 Actividad Single programa Java

```
1   package com.ebtlocator;
2
3   import android.app.Activity;
4   import android.bluetooth.BluetoothAdapter;
5   import android.bluetooth.BluetoothDevice;
6   import android.content.BroadcastReceiver;
```

```
7   import android.content.Context;
8   import android.content.Intent;
9   import android.content.IntentFilter;
10  import android.os.Bundle;
11  import android.view.View;
12  import android.view.View.OnClickListener;
13  import android.widget.ArrayAdapter;
14  import android.widget.Button;
15  import android.widget.ListView;
16  import android.widget.Toast;
17
18  public class Single extends Activity {
19      private Button findBtn;          //Show available bluetooth device
20                                         in area bluetooth button
21      private ListView myListView;     //List accessory of interface
22      private BluetoothAdapter myBluetoothAdapter;    //Bluetooth adapter
23                                         to obtains self characteristic
24      private ArrayAdapter<String> BTArrayAdapter;
25
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28              super.onCreate(savedInstanceState);
29              setContentView(R.layout.activity_single);
30
31              // take an instance of BluetoothAdapter - Bluetooth radio
32              myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
33              if(myBluetoothAdapter == null) {
34                  findBtn.setEnabled(false);
35
36                  Toast.makeText(getApplicationContext(),"Your device
37                                          does not support Bluetooth",
38                         Toast.LENGTH_LONG).show();
39              } else {
31
40                  findBtn = (Button)findViewById(R.id.search);
41                  findBtn.setOnClickListener(new OnClickListener() {
42
43                    @Override
44                    public void onClick(View v) {
45                        find();
46                    }
47                  });
48
49                  myListView = (ListView)findViewById(R.id.listView1);
50
51                  // create the arrayAdapter that contains the
52                        BTDevices, and set it to the ListView
53                  BTArrayAdapter = new ArrayAdapter<String>(this,
54                            android.R.layout.simple_list_item_1);
55                  myListView.setAdapter(BTArrayAdapter);
56
57
58              }
59          }
60
61  ////////////////////////////////////////////////////////////////////
62  ////////////////////////////////////
63  //               CLASS FUNCTIONS
64  ////////////////////////////////////////////////////////////////////
```

```
65    ////////////////////////////////////////
66
67
68    //FOUND DEVICES LIST FUNCTION
69    //START DISCOVERY MODE OF THE BLUETOOTH ADAPTER AND INITIALIZE THE
70    ARRAYS AND THE MESSAGE TO SEND
71    //ACTIVATE REGISTER-RECEIVER TO FOUND DEVICES IN AREA
72
73        public void find() {
74            if (myBluetoothAdapter.isDiscovering()) {
75                    // the button is pressed when it discovers, so cancel
76                       the discovery
77                    myBluetoothAdapter.cancelDiscovery();
78            }
79            else {
80                     BTArrayAdapter.clear();
81
82                     myBluetoothAdapter.startDiscovery();
83
84                     registerReceiver(bReceiver, new
85                          IntentFilter(BluetoothDevice.ACTION_FOUND));
86            }
87        }
88
89    //BROADCAST RECEIVER (LISTENER) FUNCTION
90    //WAIT FOR INCOMMING DEVICE IN AREA (ACTION_FOUND)
91    //REGISTER THE DEVICE CHARACTERISTICS (EXTRA_DEVICE AND EXTRA_RSSI)
92    //STORAGE IN DEVICELIST IF IT NOT EXISTS IN THE LIST, IF IT EXISTS,
93    ONLY CHANGE THE RSSI VALUE
94    //ADD INFORMATION AT THE MASSAGE TO SEND
95
96        final BroadcastReceiver bReceiver = new BroadcastReceiver() {
97            public void onReceive(Context context, Intent intent) {
98                 String action = intent.getAction();
99
100                if (BluetoothDevice.ACTION_FOUND.equals(action)) {
101
102                     BluetoothDevice device =
103         intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
104                      int rssi =
105         intent.getShortExtra(BluetoothDevice.EXTRA_RSSI,Short.MIN_VALUE);
106
107                     BTArrayAdapter.add(device.getName() + "\n" +
108         device.getAddress() + "\n" + String.valueOf(rssi));
109
110                     BTArrayAdapter.notifyDataSetChanged();
111                }
112            }
113         };
114
115
116
117
118
119   //DESTROY FUNCTION
120   //DESTROY RECEIVER
121
122     @Override
123     protected void onDestroy() {
```

```
124             super.onDestroy();
125             unregisterReceiver(bReceiver);
126         }
127
128     }
```

Fragmento 40 : Programa Java de actividad *Single*

9.1.6 Actividad Beacon layout

```
1   <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent"
6       android:paddingBottom="@dimen/activity_vertical_margin"
7       android:paddingLeft="@dimen/activity_horizontal_margin"
8       android:paddingRight="@dimen/activity_horizontal_margin"
9       android:background="@android:color/black"
10      android:paddingTop="@dimen/activity_vertical_margin"
11
12      tools:context="com.ebtlocator.Beacon" >
13
14      <ListView
15          android:id="@+id/listView1"
16          android:layout_width="match_parent"
17          android:layout_height="wrap_content"
18          android:layout_marginTop="200dp"
19          android:layout_marginBottom="50dp"
20          android:background="@android:color/darker_gray" >
21
22      </ListView>
23
24      <Button
25          android:id="@+id/cancelBeacon"
26          android:layout_width="150dp"
27          android:layout_height="wrap_content"
28          android:layout_alignParentBottom="true"
29          android:layout_alignParentRight="true"
30          android:layout_marginLeft="155dp"
31          android:background="@android:color/holo_blue_light"
32          android:text="CIERRA BALIZA" />
33
34      <Button
35          android:id="@+id/startBeacon"
36          android:layout_width="150dp"
37          android:layout_height="wrap_content"
38          android:layout_alignParentBottom="true"
39          android:layout_alignParentLeft="true"
31          android:layout_marginRight="155dp"
40          android:background="@android:color/holo_blue_light"
41          android:text="COMIENZA BUSQUEDA" />
42
43      <Button
44          android:id="@+id/search"
45          android:layout_width="80dp"
```

```
46          android:layout_height="wrap_content"
47          android:layout_alignParentTop="true"
48          android:layout_alignParentLeft="true"
49          android:textSize="11dp"
50          android:background="@android:color/holo_blue_light"
51          android:text="BUSCA DISPOSITIVOS" />
52
53      <EditText
54          android:id="@+id/distance"
55          android:layout_width="100dp"
56          android:layout_height="wrap_content"
57          android:layout_alignBottom="@+id/search"
58          android:layout_alignLeft="@+id/distanceText"
59          android:background="@android:drawable/editbox_background"
60          android:ems="10" >
61
62          <requestFocus />
63      </EditText>
64
65      <TextView
66          android:id="@+id/distanceText"
67          android:layout_width="wrap_content"
68          android:layout_height="wrap_content"
69          android:layout_alignTop="@+id/search"
70          android:layout_centerHorizontal="true"
71          android:text="DISTANCIA (cm)"
72          android:textColor="@android:color/holo_blue_light" />
73
74      <Button
75          android:id="@+id/calib"
76          android:layout_width="80dp"
77          android:layout_height="20dp"
78          android:layout_alignParentRight="true"
79          android:layout_alignTop="@+id/distanceText"
80          android:background="@android:color/holo_blue_light"
81          android:text="NUEVO PUNTO"
82          android:textSize="11dp" />
83
84      <TextView
85          android:id="@+id/dist2OriginText"
86          android:layout_width="wrap_content"
87          android:layout_height="wrap_content"
88          android:layout_alignParentTop="true"
89          android:layout_alignParentLeft="true"
90          android:layout_marginTop="70dp"
91          android:text="DISTANCIA A ORIGEN EJE X (cm)"
92          android:textColor="@android:color/holo_blue_light" />
93
94      <EditText
95          android:id="@+id/dist2Origin"
96          android:layout_width="210dp"
97          android:layout_height="wrap_content"
98          android:layout_alignLeft="@+id/dist2OriginText"
99          android:layout_below="@+id/dist2OriginText"
100         android:background="@android:drawable/editbox_background"
101         android:ems="10" />
102
103     <Button
104         android:id="@+id/selMaster"
```

```
105         android:layout_width="match_parent"
106         android:layout_height="wrap_content"
107         android:layout_alignLeft="@+id/listView1"
108         android:layout_below="@+id/dist2Origin"
109         android:layout_marginTop="20dp"
110         android:background="@android:color/holo_blue_light"
111         android:text="SELECCIONA MAESTRO" />
112
113     <Button
114         android:id="@+id/resetCalib"
115         android:layout_width="100dp"
116         android:layout_height="25dp"
117         android:layout_alignParentRight="true"
118         android:layout_marginTop="30dp"
119         android:background="@android:color/holo_blue_light"
120         android:text="CANCELA CALIBRACION"
121         android:textSize="11dp" />
122
123     <Button
124         android:id="@+id/selecPos"
125         android:layout_width="70dp"
126         android:layout_height="wrap_content"
127         android:layout_alignBottom="@+id/dist2Origin"
128         android:layout_alignRight="@+id/resetCalib"
129         android:background="@android:color/holo_blue_light"
130         android:text="CONFIRMA POSICION"
131         android:textSize="11dp" />
132
133 </RelativeLayout>
```

Fragmento 41 : Programa Layout de actividad *Beacon*

9.1.7 Actividad Beacon programa Java

```
1   package com.ebtlocator;
2
3   import java.io.IOException;
4   import java.io.InputStream;
5   import java.io.OutputStream;
6   import java.util.Comparator;
7   import java.util.Set;
8   import java.util.UUID;
9
10  import android.app.Activity;
11  import android.bluetooth.BluetoothAdapter;
12  import android.bluetooth.BluetoothDevice;
13  import android.bluetooth.BluetoothSocket;
14  import android.content.BroadcastReceiver;
15  import android.content.Context;
16  import android.content.Intent;
17  import android.content.IntentFilter;
18  import android.os.Bundle;
19  import android.os.CountDownTimer;
20  import android.view.View;
21  import android.view.View.OnClickListener;
22  import android.widget.AdapterView;
```

```
23  import android.widget.ArrayAdapter;
24  import android.widget.Button;
25  import android.widget.EditText;
26  import android.widget.ListView;
27  import android.widget.Toast;
28  import android.widget.AdapterView.OnItemClickListener;
29
30  public class Beacon extends Activity {
31
32          private Button listBtn;
33      //Show paired bluetooth devices list button
34          private Button findBtn;
35      //Show available bluetooth device in area bluetooth button
36          private Button beaconBtn;
37      //Turn on beacon mode button
38          private Button outBeaconBtn;
39      //Turn off beacon mode button
31          private Button calibrateBtn;
40      //Calibrate beacon button
41          private Button outCalibBtn;
42      //Reset calibration button
43          private Button selPosBtn;
44      //Confirm beacon position button
45          private EditText calibDist;
46      //Text editable field to insert distance of device and calibrate
47          private EditText beaconPos;
48      //Text editable field to insert the beacon position
49          private ListView myListView;
50      //List accessory of interface
51          private BluetoothAdapter myBluetoothAdapter;
52      //Bluetooth adapter to obtains self characteristic
53          private Set<BluetoothDevice> pairedDevices;
54      //Set that contains information of paired device adapters
55          private ArrayAdapter<String> BTArrayAdapter;
56      //Adapter between list view and other generated lists
57          private ArrayAdapter<BluetoothDevice> pairedList;
58      //Generated list of paired devices
59          private ArrayAdapter<DevLocalized> deviceList;
60      //Generated list of found devices
61          private ArrayAdapter<calibPoint> calibList;
62      //Generated list of calibration points
63          private BluetoothSocket mBluetoothSocket = null;
64      //Communication bluetooth socket
65          private int typeList = 0;
66      // 1: paired list, 2: Find list
67          private int indexMaster = 99;
68      //Master address into the paired devices list
69          private int rssiSelect = 0;
70      //Device rssi signal into unknown devices list
71          private int posX = 0;
72      //Position of any beacon in the X axis, initial origin x = 0
73          private boolean isCalibrated = false;
74          private UUID bluetoothUUID = UUID.fromString("00001111-0000-
75  1000-18000-001011111111");      //Required key for socket connection
76                                  //
77          private String message = "blank";
78      //Initialization of interchanged message
79
80          @Override
```

```
81              protected void onCreate(Bundle savedInstanceState) {
82                 super.onCreate(savedInstanceState);
83                 setContentView(R.layout.activity_beacon);
84
85                 // take an instance of BluetoothAdapter - Bluetooth radio
86                 myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
87
88                     listBtn = (Button)findViewById(R.id.selMaster);
89                     listBtn.setOnClickListener(new OnClickListener() {
90
91                       @Override
92                       public void onClick(View v) {
93                           // TODO Auto-generated method stub
94                        typeList = 1;
95                           list(v);
96                       }
97                     });
98
99                     findBtn = (Button)findViewById(R.id.search);
100                    findBtn.setOnClickListener(new OnClickListener() {
101
102                      @Override
103                      public void onClick(View v) {
104                          // TODO Auto-generated method stub
105                       typeList = 2;
106                          find();
107                      }
108                    });
109
110                    beaconPos = (EditText) findViewById(R.id.dist2Origin);
111                    selPosBtn = (Button)findViewById(R.id.selecPos);
112                    selPosBtn.setOnClickListener(new OnClickListener() {
113
114                      @Override
115                      public void onClick(View v) {
116                        if(beaconPos.getText().toString().length() > 0) {
117                             posX =
118                          Integer.parseInt(beaconPos.getText().toString());
119
120                          Toast.makeText(getApplicationContext(),"BALIZA EN
121                                  X = " + posX, Toast.LENGTH_SHORT).show();
122                        }
123                      }
124                    });
125
126                    final CountDownTimer clientTimer = new
127                            CountDownTimer(11000, 2000) {
128
129                     public void onTick(long millis_to_end) {
130                        find();
131
132                        if(message.length()>5) {
133
134                            ConnectThread conecta = new
135                          ConnectThread(pairedList.getItem(indexMaster));
136                            if(mBluetoothSocket!=null) {
137                                  conecta.run();
138                                  conecta.cancel();
139                             }
```

```
140
141                         }
142
143                     }
144
145                     public void onFinish() {
146                       this.start();
147                     }
148                   };
149
150                   beaconBtn = (Button)findViewById(R.id.startBeacon);
151                   beaconBtn.setOnClickListener(new OnClickListener() {
152
153                     @Override
154                     public void onClick(View v) {
155                      typeList = 0;
156                      if(isCalibrated & indexMaster < 99)
157                            clientTimer.start();
158                      else {
159                            if(!isCalibrated)
160
161                              Toast.makeText(getApplicationContext(),
162                    "CALIBRACION SIN COMPLETAR",Toast.LENGTH_SHORT).show();
163                            if(indexMaster >= 99)
164
165                              Toast.makeText(getApplicationContext(),
166                          "SELECCIONAR MAESTRO",Toast.LENGTH_SHORT).show();
167                       }
168                     }
169                    });
170
171                   outBeaconBtn = (Button)findViewById(R.id.cancelBeacon);
172                   outBeaconBtn.setOnClickListener(new OnClickListener() {
173
174                     @Override
175                     public void onClick(View v) {
176                      clientTimer.cancel();
177                     }
178                    });
179
180                   calibDist = (EditText) findViewById(R.id.distance);
181                   calibrateBtn = (Button)findViewById(R.id.calib);
182                   calibrateBtn.setOnClickListener(new OnClickListener() {
183
184                     @Override
185                     public void onClick(View v) {
186                         // TODO Auto-generated method stub
187                      if(calibDist.getText().toString().length() > 0 &&
188                                                      rssiSelect != 0) {
189                        calibList.add(new
190               calibPoint(Integer.parseInt(calibDist.getText().toString()),
191                                                           rssiSelect));
192                        if(calibList.getCount()>1)
193                           calibList.sort(new Comparator<calibPoint>() {
194                               @Override
195                               public int compare(calibPoint o1,
196                                                  calibPoint o2) {
197                                   return o1.compareTo(o2);
198                                }
```

```
199                             });
200                           for(int i = 0; i < calibList.getCount() ; ++i)
201
202                     Toast.makeText(getApplicationContext(),"Calibrate : " +
203                                 calibList.getItem(i).getDistance() + " " +
204                 calibList.getItem(i).getRSSI(),Toast.LENGTH_SHORT).show();
205
206                                 rssiSelect = 0;
207
208                             }
209
210                         if(calibList.getCount()>2) {
211                               isCalibrated = true;
212                                  Toast.makeText(getApplicationContext(),"Ya
213                                 esta calibrado",Toast.LENGTH_SHORT).show();
214                         }
215                         else
216                               isCalibrated = false;
217                       }
218                     });
219
220                     outCalibBtn = (Button)findViewById(R.id.resetCalib);
221                     outCalibBtn.setOnClickListener(new OnClickListener() {
222
223                       @Override
224                       public void onClick(View v) {
225                           // TODO Auto-generated method stub
226                        if(!calibList.isEmpty()) {
227                               calibList.clear();
228                           }
229                       }
230                     });
231
232                     myListView = (ListView)findViewById(R.id.listView1);
233
234   //create the arrayAdapter that contains the BTDevices, and set it to
235   the ListView
236                     deviceList = new ArrayAdapter<DevLocalized>(this,
237                                   android.R.layout.simple_list_item_1);
238                     pairedList = new ArrayAdapter<BluetoothDevice>(this,
239                                   android.R.layout.simple_list_item_1);
240                     calibList = new ArrayAdapter<calibPoint>(this,
241                                   android.R.layout.simple_list_item_1);
242                     BTArrayAdapter = new ArrayAdapter<String>(this,
243                                   android.R.layout.simple_list_item_1);
244                     myListView.setAdapter(BTArrayAdapter);
245
246                     myListView.setOnItemClickListener(new
247                                                 OnItemClickListener() {
248
249                               @Override
250                         public void onItemClick(AdapterView<?> parent, View
251                                               view, int position, long id) {
252
253                                   if(typeList == 1) {
254                                         indexMaster = position;
255
256           Toast.makeText(getApplicationContext(),"Dispositivo:"+"\n" +
257   pairedList.getItem(position).getName() , Toast.LENGTH_LONG).show();
```

```
258                                     }
259
260                                     if(typeList == 2) {
261                                           rssiSelect =
262                                     deviceList.getItem(position).getRSSI();
263
264         Toast.makeText(getApplicationContext(),"RSSI = "+
265         deviceList.getItem(position).getRSSI(),Toast.LENGTH_LONG).show();
266                                     }
267
268                                 }
269                         });
270
271                 }
272
273   ////////////////////////////////////////////////////////////////////////
274   ////////////////////////////////////
275   //                                          CLASS FUNCTIONS
276   ////////////////////////////////////////////////////////////////////////
277   ////////////////////////////////////
278
279   //PAIRED DEVICES LIST FUNCTION
280   //OBTAINS BONDED DEVICES AND ADD EACH ONE (NAME AND ADDRESS) TO THE
281   PAIREDLIST IN STRING FORMAT
282
283             public void list(View view){
284                pairedDevices = myBluetoothAdapter.getBondedDevices();
285                BTArrayAdapter.clear();
286                pairedList.clear();
287                for(BluetoothDevice device : pairedDevices) {
288                    BTArrayAdapter.add(device.getName()+ "\n" +
289                                                 device.getAddress());
290                    pairedList.add(device);
291                }
292
293                Toast.makeText(getApplicationContext(),"Show Paired
294                                 Devices",Toast.LENGTH_SHORT).show();
295
296             }
297
298   //FOUND DEVICES LIST FUNCTION
299   //START DISCOVERY MODE OF THE BLUETOOTH ADAPTER AND INITIALIZE THE
300   ARRAYS AND THE MESSAGE TO SEND
301   //ACTIVATE REGISTER-RECEIVER TO FOUND DEVICES IN AREA
302
303       public void find() {
304           if (myBluetoothAdapter.isDiscovering()) {
305   // the button is pressed when it discovers, so cancel the discovery
306               myBluetoothAdapter.cancelDiscovery();
307           }else {
308               BTArrayAdapter.clear();
309               deviceList.clear();
310               if(isCalibrated)
311                    message = "/";
312
313               myBluetoothAdapter.startDiscovery();
314
315               registerReceiver(bReceiver, new
316                         IntentFilter(BluetoothDevice.ACTION_FOUND));
```

```
317                 }
318             }
319
320     //BROADCAST RECEIVER (LISTENER) FUNCTION
321     //WAIT FOR INCOMMING DEVICE IN AREA (ACTION_FOUND)
322     //REGISTER THE DEVICE CHARACTERISTICS (EXTRA_DEVICE AND EXTRA_RSSI)
323     //STORAGE IN DEVICELIST IF IT NOT EXISTS IN THE LIST, IF IT EXISTS,
324     ONLY CHANGE THE RSSI VALUE
325     //ADD INFORMATION AT THE MASSAGE TO SEND
326
327         final BroadcastReceiver bReceiver = new BroadcastReceiver() {
328             public void onReceive(Context context, Intent intent) {
329                 String action = intent.getAction();
330
331                 if (BluetoothDevice.ACTION_FOUND.equals(action)) {
332
333                     BluetoothDevice device =
334                  intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
335                     int rssi =
336          intent.getShortExtra(BluetoothDevice.EXTRA_RSSI,Short.MIN_VALUE);
337                     boolean exist = false;
338                     int pointer = 0;
339                     int dist = 0;
331
340                    for(int i = 0; i <= deviceList.getCount()-1; ++i) {
341
342     if(deviceList.getItem(i).getBTDev().getAddress().equals(device.getAddre
343     ss()) & rssi!=deviceList.getItem(i).getRSSI() &
344     deviceList.getItem(i).getBTDev().getName().equals(device.getName())) {
345                                             exist = true;
346                                             pointer = i;
347                                         }
348                     }
349
350                     if(exist) {
351
352     BTArrayAdapter.remove(BTArrayAdapter.getItem(pointer));
353                                     BTArrayAdapter.add(device.getName() +
354     "\n" + device.getAddress() + "\n" + String.valueOf(rssi));
355                                     BTArrayAdapter.notifyDataSetChanged();
356
357
358     deviceList.remove(deviceList.getItem(pointer));
359                                     deviceList.add(new DevLocalized(device,
360     rssi));
361                                 }else if(!exist){
362                                     BTArrayAdapter.add(device.getName() +
363     "\n" + device.getAddress() + "\n" + String.valueOf(rssi));
364                                     BTArrayAdapter.notifyDataSetChanged();
365                                     deviceList.add(new
366     DevLocalized(device, rssi));
367                                 }
368
369                                 if(isCalibrated) {
370                                       dist = convert2Distance(rssi);
371                                       message = message + "%" + posX + "$"
372     + device.getAddress() + "&" + dist + "/";
373                                 }
374                     }
```

```
375                 }
376             };
377 
378             public int convert2Distance(int rssi) {
379 
380                 int distInt = 0;
381 
382                     for(int i = 0; i < calibList.getCount() ; ++i) {
383                           if(rssi <= calibList.getItem(0).getRSSI()) {
384                                 calibPoint limitMax = new
385     calibPoint(calibList.getItem(0).getDistance(),calibList.getItem(0).getR
386     SSI());
387                                 calibPoint limitMin = new
388     calibPoint(calibList.getItem(1).getDistance(),calibList.getItem(1).getR
389     SSI());
390 
391     //Interpolation between limitMax and limitMin from calibList
392                                 distInt = limitMin.getDistance() +
393     (limitMax.getDistance()-limitMin.getDistance())/(limitMax.getRSSI()-
394     limitMin.getRSSI())*(rssi - limitMin.getRSSI());
395                                 break;
396                           }
397                           else if(rssi >
398     calibList.getItem(calibList.getCount()-1).getRSSI()) {
399                                 calibPoint limitMax = new
400     calibPoint(calibList.getItem(calibList.getCount()-
401     1).getDistance(),calibList.getItem(calibList.getCount()-1).getRSSI());
402                                 calibPoint limitMin = new
403     calibPoint(calibList.getItem(calibList.getCount()-
404     2).getDistance(),calibList.getItem(calibList.getCount()-2).getRSSI());
405 
406     //Interpolation between limitMax and limitMin from calibList
407                                 distInt = limitMin.getDistance() +
408     (limitMax.getDistance()-limitMin.getDistance())/(limitMax.getRSSI()-
409     limitMin.getRSSI())*(rssi - limitMin.getRSSI());
410                                 break;
411                           }
412                           else if(rssi <= calibList.getItem(i).getRSSI())
413     {
414                                 calibPoint limitMax = new
415     calibPoint(calibList.getItem(i-1).getDistance(),calibList.getItem(i-
416     1).getRSSI());
417                                 calibPoint limitMin = new
418     calibPoint(calibList.getItem(i).getDistance(),calibList.getItem(i).getR
419     SSI());
420 
421     //Interpolation between limitMax and limitMin from calibList
422                                 distInt = limitMin.getDistance() +
423     (limitMax.getDistance()-limitMin.getDistance())/(limitMax.getRSSI()-
424     limitMin.getRSSI())*(rssi - limitMin.getRSSI());
425                                 break;
426                           }
427                     }
428                     return distInt;
429             }
430 
431 
432           //DESTROY FUNCTION
433           //DESTROY RECEIVER
```

```
434
435             @Override
436             protected void onDestroy() {
437                 // TODO Auto-generated method stub
438                 super.onDestroy();
439                 unregisterReceiver(bReceiver);
431             }
440
441     ////////////////////////////////////////////////////////////////////
442     ////////////////////////////////////
443     //                                        COMMUNICATION CLASSES
444     ////////////////////////////////////////////////////////////////////
445     ////////////////////////////////////
446
447     //CLIENT CLASS
448     //CONSTRUCTOR       :       CREATE A BLUETOOTH SOCKET
449     //RUN               :       CONNECT WITH SERVER AND SEND MESSAGE
450
451         private class ConnectThread extends Thread {
452
453               public ConnectThread(BluetoothDevice device) {
454                   try {
455                       mBluetoothSocket =
456     device.createRfcommSocketToServiceRecord(bluetoothUUID);
457                   } catch (IOException e) {
458                        Toast.makeText(getApplicationContext(),"client NOT
459                                  AVAILABLE" , Toast.LENGTH_LONG).show();
460                   }
461               }
462
463               public void run() {
464                   try {
465                       mBluetoothSocket.connect();
466                         if (mBluetoothSocket != null) {
467                           StreamThread mStreamThread = new StreamThread();
468                           mStreamThread.start();
469
470                            byte[] data = message.getBytes();
471                            mStreamThread.write(data);
472
473                           Toast.makeText(getApplicationContext(), message,
474                                                   Toast.LENGTH_LONG).show();
475
476                         }
477                     } catch (IOException e) {
478                        Toast.makeText(getApplicationContext(),"Server NOT
479                              accept a client" , Toast.LENGTH_LONG).show();
480                         }
481                     }
482
483                   public void cancel() {
484                       try {
485                           mBluetoothSocket.close();
486                       } catch (IOException e) {}
487                   }
488
489         }
490
491
```

```
492         //STREAM CLASS
493         //CONSTRUCTOR       :      CREATE A COMMUNICATIONS STREAMS
494   (BUFFERS) INPUT AND OUTPUT
495         //READ              :      EXTRACT BYTES FROM IN BUFFER AND SAVE
496   IN A STRING VARIABLE
497         //WRITE                    :      WRITE BYTES IN THE OUT BUFFER
498
499         private class StreamThread extends Thread {
500
501             private InputStream in;
502             private OutputStream out;
503
504             public StreamThread() {
505                 try {
506                     in = mBluetoothSocket.getInputStream();
507                     out = mBluetoothSocket.getOutputStream();
508                 } catch (IOException e) {}
509             }
510
511              public void write(byte[] data) throws IOException {
512               try {
513                     out.write(data);
514                   }catch (Exception e) {}
515              }
516         }
517
518   ////////////////////////////////////////////////////////////////////////
519   ////////////////////////////////////////
520   //           DATA MANAGE CLASSES
521   ////////////////////////////////////////////////////////////////////////
522   ////////////////////////////////////////
523
524   //FOUND DEVICES CLASS
525   //CONSTRUCTOR       :      INITIALIZE A DEVICE FROM BLUETOOTHDEVICE
526                              CLASS AND ITS RSSI SIGNAL
527   //GETS              :      OBTAINS THE EXPLICIT OBJECT
528
529         private class DevLocalized {
530
531            private BluetoothDevice btDev;
532            private int rssi;
533
534            public DevLocalized(BluetoothDevice btd, int signal) {
535               btDev = btd;
536               rssi = signal;
537            }
538
539            public BluetoothDevice getBTDev() {
531                   return btDev;
540            }
541
542            public int getRSSI() {
543                   return rssi;
5445           }
545
546         }
547
548         private class calibPoint{
549
```

```
550                     private int distance;
551                     private int rssi;
552
553                     public calibPoint(int dist, int signal) {
554                        distance = dist;
555                        rssi = signal;
556                     }
557
558                  public int compareTo(calibPoint o2) {
559                        // TODO Auto-generated method stub
560                        if (rssi < o2.rssi) {
561                        return -1;
562                    }
563                    if (rssi > o2.rssi) {
564                        return 1;
565                    }
566                    return 0;
567                  }
568
569                  public int getDistance() {
570                           return distance;
571                     }
572
573                     public int getRSSI() {
574                           return rssi;
575                     }
576                  }
577
578   }
```

Fragmento 42 : Programa Java de actividad *Beacon*

9.1.8 Actividad Master layout

```
1   <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent"
6       android:paddingBottom="@dimen/activity_vertical_margin"
7       android:paddingLeft="@dimen/activity_horizontal_margin"
8       android:paddingRight="@dimen/activity_horizontal_margin"
9       android:paddingTop="@dimen/activity_vertical_margin"
10      android:background="@android:color/black"
11      tools:context="com.ebtlocator.Master" >
12
13      <TextView
14          android:id="@+id/textMSGList"
15          android:layout_width="wrap_content"
16          android:layout_height="wrap_content"
17          android:layout_alignParentTop="true"
18          android:layout_centerHorizontal="true"
19          android:textColor="@android:color/holo_blue_light"
20          android:textSize="18dp"
21          android:text="COLA DE MENSAJES" />
22
```

```
23      <ListView
24          android:id="@+id/listViewMSG"
25          android:layout_width="match_parent"
26          android:layout_height="wrap_content"
27          android:layout_below="@+id/textMSGList"
28          android:layout_marginTop="1dp"
29          android:layout_marginBottom="200dp"
30          android:background="@android:color/darker_gray" >
31
32      </ListView>
33
34      <TextView
35          android:id="@+id/textLocations"
36          android:layout_width="wrap_content"
37          android:layout_height="wrap_content"
38          android:layout_alignParentBottom="true"
39          android:layout_centerHorizontal="true"
31          android:layout_marginBottom="175dp"
40          android:text="LOCALIZACIONES"
41          android:textColor="@android:color/holo_blue_light"
42          android:textSize="18dp" />
43
44      <ListView
45          android:id="@+id/listViewLocations"
46          android:layout_width="match_parent"
47          android:layout_height="wrap_content"
48          android:layout_alignTop="@+id/textLocations"
49          android:layout_marginTop="27dp"
50          android:layout_marginBottom="40dp"
51          android:background="@android:color/darker_gray" >
52      </ListView>
53
54      <Button
55          android:id="@+id/cancelMaster"
56          android:layout_width="match_parent"
57          android:layout_height="30dp"
58          android:layout_alignParentBottom="true"
59          android:background="@android:color/holo_blue_light"
60          android:text="CIERRA MAESTRO" />
61
62  </RelativeLayout>
```

Fragmento 43 : Programa Layout de actividad *Master*

9.1.9 Actividad Master programa Java

```
1   package com.ebtlocator;
2
3   import java.io.IOException;
4   import java.io.InputStream;
5   import java.io.OutputStream;
6   import java.util.UUID;
7
8   import android.app.Activity;
9   import android.bluetooth.BluetoothAdapter;
10  import android.bluetooth.BluetoothServerSocket;
```

```
11  import android.bluetooth.BluetoothSocket;
12  import android.content.Intent;
13  import android.os.Bundle;
14  import android.os.CountDownTimer;
15  import android.view.View;
16  import android.view.View.OnClickListener;
17  import android.widget.AdapterView;
18  import android.widget.ArrayAdapter;
19  import android.widget.Button;
20  import android.widget.ListView;
21  import android.widget.Toast;
22  import android.widget.AdapterView.OnItemClickListener;
23
24  public class Master extends Activity {
25
26          private Button serverBtn;
27        //Turn on master mode button
28          private Button outServerBtn;
29        //Turn off master mode button
30          private ListView myListMSG;
31        //List accessory of interface
32          private ListView myListLocations;
33        //List accessory of interface
34          private BluetoothAdapter myBluetoothAdapter;
35        //Bluetooth adapter to obtains self characteristic
36          private ArrayAdapter<String> BTArrayAdapter1;
37        //Adapter between list view and other generated lists
38          private ArrayAdapter<String> BTArrayAdapter2;
39        //Adapter between list view and other generated lists
31          private ArrayAdapter<BeaconData> finalDevArray;
40        //Generated list of information received from beacons
41          private ArrayAdapter<Coordinates> locationList;
42          private BluetoothSocket mBluetoothSocket = null;
43        //Communication bluetooth socket
44          private UUID bluetoothUUID = UUID.fromString("00001111-0000-
45  1000-18000-001011111111");      //Required key for socket connection
46                                  //
47          private String message = "blank";
48        //Initialization of interchanged message
49
50          @Override
51          protected void onCreate(Bundle savedInstanceState) {
52             super.onCreate(savedInstanceState);
53             setContentView(R.layout.activity_master);
54
55             // take an instance of BluetoothAdapter - Bluetooth radio
56             myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
57
58            myListMSG = (ListView)findViewById(R.id.listViewMSG);
59            myListLocations =
60                  (ListView)findViewById(R.id.listViewLocations);
61
62           finalDevArray = new ArrayAdapter<BeaconData>(this,
63                           android.R.layout.simple_list_item_1);
64           locationList = new ArrayAdapter<Coordinates>(this,
65                           android.R.layout.simple_list_item_1);
66
67           BTArrayAdapter1 = new ArrayAdapter<String>(this,
68                           android.R.layout.simple_list_item_1);
```

```
69                BTArrayAdapter2 = new ArrayAdapter<String>(this,
70                                android.R.layout.simple_list_item_1);
71
72                myListMSG.setAdapter(BTArrayAdapter1);
73                myListMSG.setOnItemClickListener(newOnItemClickListener() {
74
75                   @Override
76                   public void onItemClick(AdapterView<?> parent, View view,
77                                                   int position, long id) {
78
79                   }
80                });
81
82                myListLocations.setAdapter(BTArrayAdapter2);
83                myListLocations.setOnItemClickListener(new
84                                                    OnItemClickListener() {
85
86                    @Override
87                    public void onItemClick(AdapterView<?> parent, View
88                                                view, int position, long id) {
89
90                    }
91                });
92
93                final CountDownTimer serverTimer = new
94                                                  CountDownTimer(11000, 500) {
95
96                   public void onTick(long millis_to_end) {
97
98                       AcceptThread accept =new AcceptThread("miServidor");
99                       accept.run();
100
101                      extractMatrix(message);
102
103                      BTArrayAdapter1.clear();
104                      for(int i = 0 ; i < finalDevArray.getCount() ; ++i) {
105
106                         String cad1 = finalDevArray.getItem(i).getOrigin()
107                          + " " + finalDevArray.getItem(i).getMac() + " " +
108                                        finalDevArray.getItem(i).getSig();
109                         BTArrayAdapter1.add(cad1);
110                         BTArrayAdapter1.notifyDataSetChanged();
111
112                         BeaconData tempDev = new
113                         BeaconData(finalDevArray.getItem(i).getOrigin(),
114    finalDevArray.getItem(i).getMac(), finalDevArray.getItem(i).getSig());
115
116                         if(finalDevArray.getCount() > 1)
117                            for(int j = 1 ; j < finalDevArray.getCount() ;
118                                                                  ++j) {
119
120                               if(tempDev.getOrigin() !=
121                                     finalDevArray.getItem(j).getOrigin() &
122               tempDev.getMac().equals(finalDevArray.getItem(j).getMac())) {
123                                  int tempDist = 0;
124                                  int tempRad1 = 0, tempRad2 = 0;
125
126                                  if(tempDev.getOrigin() >
127                                 finalDevArray.getItem(j).getOrigin()) {
```

```
128                         tempDist = tempDev.getOrigin();
129                         tempRad2 = tempDev.getSig();
130                         tempRad1 = finalDevArray.getItem(j).getSig();
131                                   }else {
132                         tempDist = finalDevArray.getItem(j).getOrigin();
133                         tempRad2 = finalDevArray.getItem(j).getSig();
134                         tempRad1 = tempDev.getSig();
135                                   }
136
137                                 obtainCordinates(finalDevArray.getItem(j)
138                                 .getMac(), tempDist, tempRad1, tempRad2);
139                                 break;
140                             }
141                         }
142                     }
143
144                 }
145
146                 public void onFinish() {
147                             this.start();
148                 }
149         };
150
151         serverTimer.start();
152
153         outServerBtn = (Button)findViewById(R.id.cancelMaster);
154         outServerBtn.setOnClickListener(new OnClickListener() {
155
156              @Override
157              public void onClick(View v) {
158                  serverTimer.cancel();
159                  BTArrayAdapter1.clear();
160                  finalDevArray.clear();
161
162                  BTArrayAdapter2.clear();
163
164               Intent mi_intento = new Intent(Master.this,Initial.class);
165                  startActivity(mi_intento);
166               }
167           });
168
169       }
170
171
172   ////////////////////////////////////////////////////////////////////////
173   ////////////////////////////////////////
174   //             CLASS FUNCTIONS
175   ////////////////////////////////////////////////////////////////////////
176   ////////////////////////////////////////
177
178   //MESSAGE MANAGE AND STORAGE FUNCTION
179   //IF THE IMPUT MESSAGE CONTAINS OR BEGINS WITH '/' READS THE COMPLETE
180   MESSAGE
181   //IF THE CHARACTER IS '%' MEANS THAT THE NEXT INFORMATION IS THE NAME
182   OF THE BEACON TRANSMITTER
183   //IF THE CHARACTER IS '$' MEANS THAT THE NEXT INFORMATION IS THE
184   ADDRESS OF THE FOUND DEVICE
185   //IF THE CHARACTER IS '&' MEANS THAT THE NEXT INFORMATION IS THE
186   DISTANCE OF THE FOUND DEVICE
```

```
187   //IF THE CHARACTER IS '/' MEANS THAT THE NODE INFORMATION (BEACON,
188   DEVICE ADDRESS, DISTANCE) IS READY TO STORE IN FINALDEVICELIST
189
190       public void extractMatrix(String msg) {
191
192           String tempBeacon = "";
193           String tempAddress = "";
194           String tempDist = "";
195           int typeData = 0;
196           int devicePointer;
197           boolean deviceHere;
198
199           if(msg.contains("/")) {
200               for (int i = 0; i < msg.length(); ++i ) {
201                  if( msg.charAt (i) == '%')      //tipo de dato baliza
202                      typeData = 1;
203                  else if( msg.charAt (i) == '$') //tipo de dato mac
204                                                    device address
205                      typeData = 2;
206                  else if( msg.charAt (i) == '&') //tipo de dato rssi
207                                                   device
208                      typeData = 3;
209                  else if( msg.charAt (i) == '/') //tipo de dato rssi
210                                                   device
211                      typeData = 4;
212
213                  if(typeData == 1 & msg.charAt (i) != '%')
214                      tempBeacon = tempBeacon + msg.charAt (i);
215                  else if(typeData == 2 & msg.charAt (i) != '$')
216                      tempAddress = tempAddress + msg.charAt (i);
217                  else if(typeData == 3 & msg.charAt (i) != '&')
218                      tempDist = tempDist + msg.charAt (i);
219
220                  if(msg.charAt (i) == '/' & i != 0) {//Carga datos en
221                                                        matriz
222                      devicePointer = 0;
223                      deviceHere = false;
224
225   //Comprueba si el dispositivo ya esta registrado en la matriz
226                     for(int m = 0; m < finalDevArray.getCount() ; ++m) {
227                         if(Integer.parseInt(tempBeacon) ==
228                                  finalDevArray.getItem(m).getOrigin() &
229                   tempAddress.equals(finalDevArray.getItem(m).getMac())) {
230                             deviceHere = true;
231                             devicePointer = m;
232                         }
233                     }
234
235                     tempBeacon.trim();
236                     tempAddress.trim();
237                     tempDist.trim();
238
239                     if(!deviceHere )
240                         finalDevArray.add(new
241                                 BeaconData(Integer.parseInt(tempBeacon),
242                                tempAddress,Integer.parseInt(tempDist)));
243                     if(deviceHere)
244                          +finalDevArray.getItem(devicePointer).
245                                    setSig(Integer.parseInt(tempDist));
```

```
246
247                         tempBeacon = "";
248                         tempAddress = "";
249                         tempDist = "";
250                         typeData = 0;
251                     }
252                 }
253             }
254         }
255
256         public void obtainCordinates(String devMAC, int beaconsDist, int
257                                                         rad1, int rad2) {
258
259             long x = 0;
260             long y = 0;
261
262             if(rad1 + rad2 <= beaconsDist)
263                 locationList.add(new Coordinates(devMAC, rad1, 0));
264             else {
265                 double tempSemiperiod = 0;
266                 double tempArea = 0;
267
268                 tempSemiperiod = (rad1 + rad2 + beaconsDist)/2;
269                 tempArea =
270                         Math.sqrt(Math.abs(tempSemiperiod*(tempSemiperiod -
271           rad1)*(tempSemiperiod - rad2)*(tempSemiperiod - beaconsDist)));
272
273                 y = Math.round(tempArea*2/beaconsDist);
274
275                 x = Math.round(rad1 * Math.cos(Math.asin(y/rad1)));
276
277                 if(x>0 & !locationList.isEmpty()) {
278                     for(int i = 0; i < locationList.getCount() ; ++i) {
279
280                         if(locationList.getItem(i).getMac().equals(devMAC))
281
282
283                             BTArrayAdapter2.remove(BTArrayAdapter2.getItem(i));
284                             BTArrayAdapter2.add(devMAC+" ("+x+","+y +")");
285
286                             BTArrayAdapter2.notifyDataSetChanged();
287
288                             locationList.remove(locationList.getItem(i));
289                             locationList.add(new Coordinates(devMAC, x, y));
290
291                             break;
292                         }
293                         if(i == locationList.getCount()-1) {
294                             BTArrayAdapter2.add(devMAC+" ("+x+","+y+")");
295
296                             BTArrayAdapter2.notifyDataSetChanged();
297
298                           locationList.add(new Coordinates(devMAC, x, y));
299                         }
300                 }else if(x>0 & locationList.isEmpty()) {
301                         BTArrayAdapter2.add(devMAC+" ("+x+","+y+")");
302                         BTArrayAdapter2.notifyDataSetChanged();
303
304                         locationList.add(new Coordinates(devMAC, x, y));
```

```
305                 }
306             }
307         }
308
309
310     ////////////////////////////////////////////////////////////////////
311     ////////////////////////////////////
312     //            COMMUNICATION CLASSES
313     ////////////////////////////////////////////////////////////////////
314     ////////////////////////////////////
315
316     //SERVER CLASS
317     //CONSTRUCTOR       :     CREATE A BLUETOOTH SERVER SOCKET
318     //RUN               :     ACCEPT A CLIENT COMM / SEND A MESSAGE IF
319     SOMEONE IS LISTENING / RECEIVE A MESSAGE FROM CLIENT / CLOSE SOCKETS
320     //CLOSE             :     CANCEL THE SERVER SOCKET
321
322         private class AcceptThread extends Thread {
323
324             private BluetoothServerSocket mServerSocket;
325
326             public AcceptThread(String name) {
327                   mServerSocket = null;
328                   try {
329                       mServerSocket =
330                  myBluetoothAdapter.listenUsingRfcommWithServiceRecord(name,
331                  bluetoothUUID);
332                   } catch (IOException e) {
333                       Toast.makeText(getApplicationContext(),"Server NOT
334                                  AVAILABLE" , Toast.LENGTH_LONG).show();
335                   }
336             }
337
338             public void run() {
339
331                   while (mServerSocket!=null) {
340                       try {
341
342                         mBluetoothSocket = mServerSocket.accept();
343                          if (mBluetoothSocket != null) {
344                               StreamThread mStreamThread = new
345                                                          StreamThread();
346                              mStreamThread.start();
347
348                                String a = message;
349                                byte[] data = a.getBytes();
350                                mStreamThread.write(data);
351
352                              message = mStreamThread.read();
353                              try {
354                               mBluetoothSocket.close();
355                              } catch (IOException e) {}
356
357                              try {
358                               mServerSocket.close();
359                              } catch (IOException e) {}
360                                     mServerSocket = null;
361                         }
362
```

```
363                         } catch (IOException e) {
364                             Toast.makeText(getApplicationContext(),
365            "Server NOT accept a client" , Toast.LENGTH_LONG).show();
366                         }
367                         break;
368                 }
369             }
370 
371         }
372 
373     //STREAM CLASS
374     //CONSTRUCTOR       :     CREATE A COMMUNICATIONS STREAMS (BUFFERS)
375                               INPUT AND OUTPUT
376     //READ              :     EXTRACT BYTES FROM IN BUFFER AND SAVE IN A
377                               STRING VARIABLE
378     //WRITE             :     WRITE BYTES IN THE OUT BUFFER
379 
380         private class StreamThread extends Thread {
381 
382                 private InputStream in;
383                 private OutputStream out;
384 
385                 public StreamThread() {
386                     try {
387                         in = mBluetoothSocket.getInputStream();
388                         out = mBluetoothSocket.getOutputStream();
389                     } catch (IOException e) {}
390                 }
391 
392                 public String read() {
393                     byte[] data = new byte[30000];
394                     int length;
395                     String text = null;
396                     while (true) {
397                         try {
398                             length = in.read(data);
399                             text = new String(data, 0, length);
400 
401                         } catch (IOException e) {
402                          text = "Empty buffer";
403                         }
404                         break;
405                     }
406                     return text;
407                  }
408 
409                  public void write(byte[] data) throws IOException {
410                    try {
411                           out.write(data);
412                        }catch (Exception e) {}
413                  }
414            }
415 
416     ////////////////////////////////////////////////////////////////////////
417     ////////////////////////////////////////
418     //           DATA MANAGE CLASSES
419     ////////////////////////////////////////////////////////////////////////
420     ////////////////////////////////////////
421 
```

```
422   //INFO FROM BEACONS CLASS
423   //CONSTRUCTOR       :      INITIALIZE WHITH TRANSMITTER NAME, ADDRESS OF
424   FOUN DEVICE AND ITS DISTANCE TO TRANSMITTER
425   //GETS              :      OBTAINS THE EXPLICIT OBJECT
426   //SETS              :      ASSIGNS THE VALUE OF A EXPLICIT OBJECT
427
428         private class BeaconData {
429
430               private int origin;
431               private String mac;
432               private int distance;
433
434             public BeaconData(int name, String macadd, int signal) {
435               origin = name;
436                 mac = macadd;
437                 distance = signal;
438             }
439
431             public void setOrigin(int name) {
440               origin = name;
441             }
442
443             public int getOrigin() {
444               return origin;
445             }
446
447             public void setMac(String macadd) {
448               mac = macadd;
449             }
450
451             public String getMac() {
452               return mac;
453             }
454
455             public void setSig(int sig) {
456               distance = sig;
457             }
458
459             public int getSig() {
460               return distance;
461             }
462         }
463
464         private class Coordinates{
465               private long x, y;
466               private String device;
467
468               public Coordinates(String objet, long x, long y) {
469                     device = objet;
470                     this.x = x;
471                     this.y = y;
472               }
473
474               public long getXposition() {
475                     return x;
476               }
477
478               public long getYposition() {
479                     return y;
```

```
480                 }
481
482                 public String getMac() {
483                       return device;
484                 }
485           }
486
487     }
```

Fragmento 44 : Programa Java de actividad *Master*