ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Aplicación para Android: Baby Monitor

AUTOR: PABLO MARCO JORNET

DIRECTOR: JUAN CARLOS SÁNCHEZ AARNOUTSE

Enero / 2016

Autor:	Pablo Marco Jornet
e-mail autor:	pablomarcojornet@gmail.com
Director(es):	Juan Carlos Sánchez Aarnoutse
e-mail director:	juanc.sanchez@upct.es
Codirector(es):	
Titulo del TFG:	Aplicación para Android: Baby Monitor
Descriptores:	

Resumen:

Desarrollo de una aplicación que lleva a cabo el funcionamiento de los dispositivos conocidos popularmente como vigilabebés, aprovechando las funcionalidades que nos proporcionan los Smartphones de última generación.

Estos dispositivos mencionados anteriormente se componen de dos terminales, uno que graba el sonido de la habitación donde está el bebé (algunos incluyen video) y envía el audio por radiofrecuencia al otro terminal que actúa como receptor.

La aplicación usa la interfaz Wifi para la conexión de los teléfonos y el envío del video y audio en tiempo real.

Titulación:	Grado en Ingeniería Telemática
Intensificación:	
Departamento:	Tecnología de la información y las comunicaciones.
Fecha:	Enero - 2016

Índice general

Capítulo 1

Introducción

Los dispositivos móviles son uno de los aspectos tecnológicos que más ha crecido en los últimos años y tienen un papel muy importante en nuestra vida cotidiana ya que casi todo el mundo dispone de un Smartphone que utiliza para trabajar, comunicarse, entretenerse, informarse, etc.

Este auge de los dispositivos móviles unido al reto de enfrentarse a un lenguaje de programación y a una plataforma nueva hicieron muy atractivo el desarrollo de esta aplicación que me parecía el mejor complemente a la formación recibida en el Grado.

A continuación se van a explicar detalladamente todos los problemas surgidos en la implementación de la aplicación, las decisiones tomadas y su justificación.

1.1 Objetivos

El objetivo de este proyecto es desarrollar una aplicación en Android que sirva como un vigilabebés aprovechando una de las últimas novedades del sistema Android, *Wifi Direct* que nos permite la conexión directa entre dos terminales a través de la interfaz Wifi.

Para la utilización de la aplicación se requieren dos dispositivos Android, el primero de los terminales captura el audio y el video y lo envía al otro terminal que lo reproduce en tiempo real.

1.2 Fases del proyecto

1. Decisión de la tecnología usada para la conexión entre los terminales
2. Estudio de la tecnología *Wifi Direct* y desarrollo de la conexión entre los terminales
3. Estudio y desarrollo de la generación, envío y reproducción de un video streaming.

1.3 Organización de la memoria

1. Introducción y objetivos.
2. Desarrollo de la aplicación y experiencia del trabajo con Android. Toma de decisiones de la herramienta a utilizar y de las características idóneas que nos ofrece el sistema Android
3. Implementación de la aplicación. Especificaciones del código, clases utilizadas y su función.
4. Manual de uso de la aplicación. Explicación detallada para que el usuario desarrolle la mejor experiencia posible.
5. Conclusión y ampliaciones futuras.

Capitulo 2

Contexto y Tecnología

2.1 Sistema Operativo Android

Android es un sistema operativo basado en Linux que, inicialmente, fue diseñado para dispositivos móviles con pantalla táctil aunque actualmente se ha extendido a una variedad de dispositivos mucho más amplia.

Fue creado por una firma denominada Android Inc. que despertó el interés de Google que en 2005 decidió comprar la empresa y en 2007 creó la *Open Handset Alliance*, conglomerado de 48 compañías dedicadas a la fabricación de hardware o software relacionado con el desarrollo de dispositivos móviles. Este conjunto de empresas abanderado por Google apostaron por Android como su sistema operativo genérico para móviles.

Android representa el liderazgo en el sector de los dispositivos móviles estando presente en más del 50% de los terminales vendidos, superando notablemente el número de ventas de los otros sistemas.

2.2 Arquitectura

Como podemos observar en la imagen 1 la arquitectura del S.O Android está formada por 5 componentes:

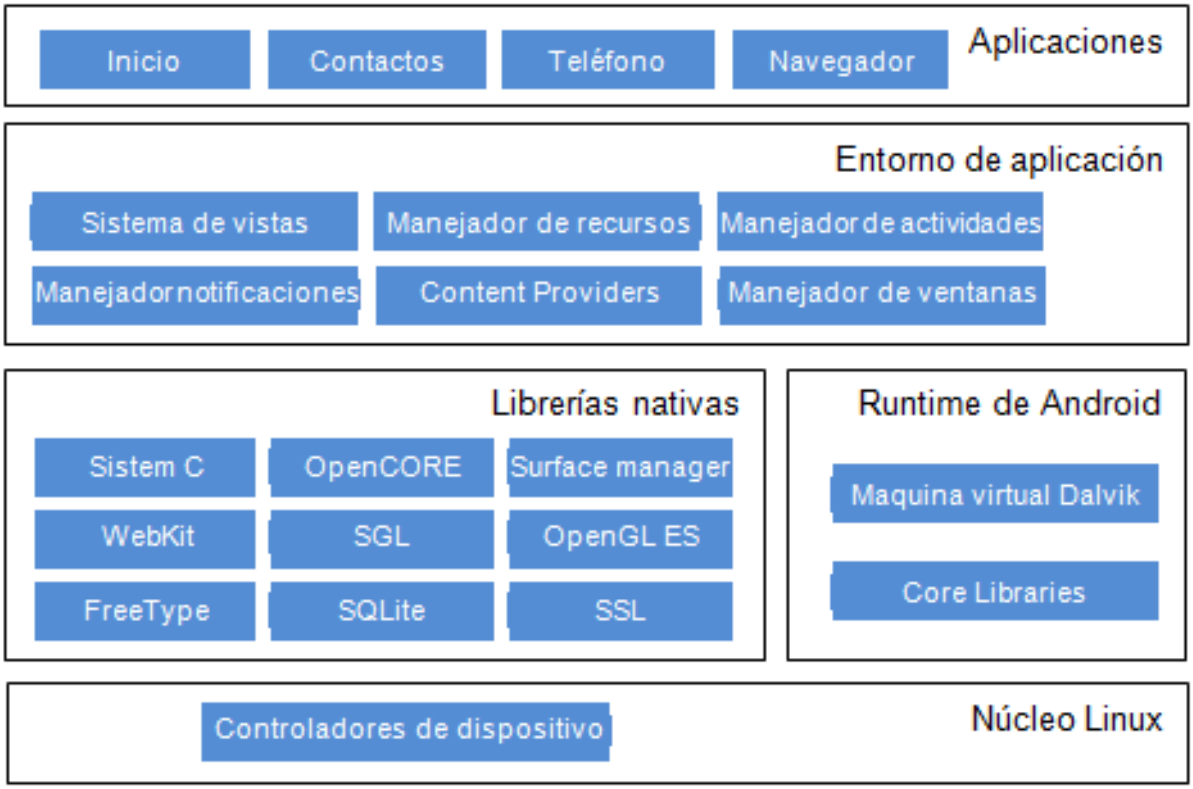

Imagen 1. Arquitectura del sistema Android

- Núcleo Linux. Es el nexo de unión entre el hardware y el software del sistema. Es la capa inferior y se encarga de la gestión de memoria, recursos, seguridad, etc.
- Las librerías de Android están escritas en C y C++ que tienen como objetico proporcionar funcionalidad a las aplicaciones para que se que desarrollen de forma más eficiente.
- El *Runtime Android* es un conjunto de librería de bajo nivel que implementan casi todas las funcionalidades disponibles a las que se puede acceder desde cualquier programa Java. El sistema incluye una maquina virtual de Java denominada *Dalvik* que Google diseñó para dispositivos con recursos limitados.
- Entorno de aplicación. Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos a través de la maquina virtual *Dalvik*.
- Aplicaciones. Constituyen el último nivel de la arquitectura del sistema operativo Android. Incluye una serie de aplicaciones como cliente de correo electrónico, aplicación para SMS, calendario, mapas, navegador, contactos, etc. Todas estas aplicaciones están escritas en lenguaje Java.

2.3 Características

Entre otras, las principales características actuales de los dispositivos que emplean Android son:

- Navegador
- Bluetooth y Wi-Fi
- GSM, EDGE, 3G y 4G
- Soporte SQLite para bases de datos
- Soporte multimedia para los siguientes formatos: WebM, H.263, H.264, MPEG-4 SP, AMR, AAC, MP3, MIDI, WAV, JPEG, PNG, GIF, BMP, etc.
- Soporte para streaming
- Máquina Virtual Dalvik
- Cámara, GPS, brújula, acelerómetro
- Pantalla táctil

Capítulo 3

Entorno y características de la aplicación

3.1 Introducción a Android

En esta sección vamos a introducir unos conceptos básicos que serán utilizados en futuros capítulos y que creemos conveniente detallar para la comprensión del trabajo realizado:

- El archivo *AndroidManifest.xml* es imprescindible y único en una aplicación de Android, se encarga de definir la información y la estructura de la aplicación. Tiene un elemento <application> en el cual habrán tantos elementos <activity> como clases Activity o actividades tengamos en la aplicación.
  Además, en este archivo aparece la información de la aplicación: nombre, versión de Android, icono, etc. Los permisos de la aplicación también aparecen en el *AndroidManifest.xml*. En el Capítulo 4 podemos observar el archivo *AndroidManifest.xml* de nuestra aplicación.
- El concepto actividad. Cuando hablamos de actividades en la aplicación nos referimos a una clase de la aplicación que hereda de la clase *Activity*. Una actividad corresponde a una ventana de nuestra aplicación y tiene asociada una interfaz gráfica que se define mediante un archivo .xml.
  Una actividad puede estar en tres estados: activa, cuando está visible e interactuando con el usuario, en pausa, cuando este visible pero no está interactuando con el usuario, y detenida cuando deja de ser visible. A continuación, en la imagen 2 mostramos el ciclo de vida de una actividad

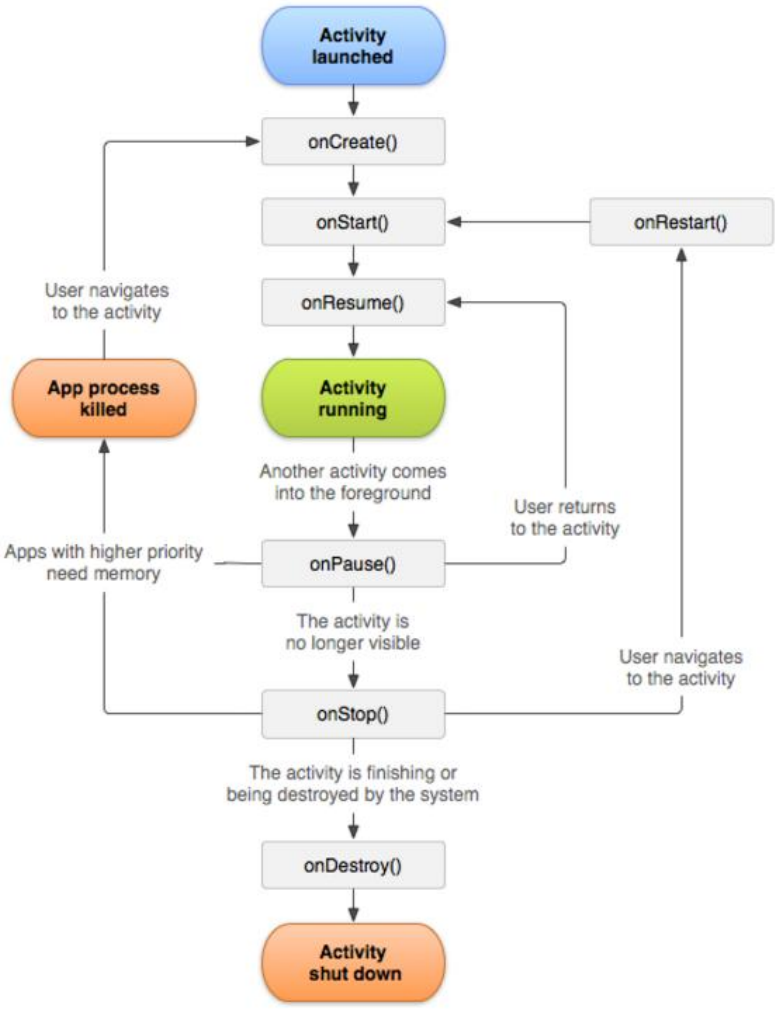

Imagen 2. Ciclo de vida de una actividad. Imagen obtenida de la API de Android.

- El concepto de *intent*. Los *intents* son unos elementos muy característicos de Android y de mucha importancia a la hora de implementar una aplicación. Estos elementos son un tipo de objetos que se encargan de lanzar una nueva actividad y nos permiten la navegación entre ventanas de nuestra aplicación. Durante la fase de explicación de la implementación veremos varios ejemplos del uso de *intents*.
- *Broadcast Receiver*. Un *Broadcast Receiver* es una clase que se encargar de escuchar los eventos globales generados por el sistema y tomar las decisiones correspondientes. Este tipo de eventos no van dirigidos a ninguna aplicación determinada sino que se generan para cualquier aplicación que necesite contar con esta información. En nuestra aplicación este tipo de elemento jugará un papel importante y su implementación será detallada en capítulos siguientes.

3.2 Entorno y características de la aplicación

Al abordar este proyecto se tuvieron que tomar varias decisiones para iniciar el trabajo. Lo primero era elegir la plataforma que utilizaríamos para la implementación de la aplicación, las dos opciones eran el uso de la herramienta Eclipse con el SDK de Android o usar el software desarrollado por Google, Android Studio. Se ha elegido esta última principalmente porque Google ha anunciado que va a dejar de actualizar el SDK de Eclipse y, por tanto, ya no se podrían utilizar para las posteriores versiones de Android que se vayan presentando.

Imagen 3. Logo de la herramienta Android Studio

La otra decisión importante que se tuvo que tomar fue la tecnología a utilizar para la conexión de los dos terminales, las dos opciones que se presentaban eran el *Bluetooth* y la interfaz Wifi. Se optó por el Wifi debido a las características de la aplicación, se entiende que los dos móviles estarán como mínimo en distintas habitaciones lo que hacía que la distancia permitida por el Bluetooth fuera insuficiente. La limitación de la distancia hizo que se utilizáramos la interfaz Wifi para conectar los terminales, además estudiando las posibilidades del Wifi en los nuevos terminales y en las nuevas versiones de Android se descubrió *Wifi Direct*, un protocolo que se ajustaba perfectamente a las características de nuestra aplicación.

Wifi Direct es un protocolo que establece una conexión p2p entre dos terminales sin necesidad de que ningún router o punto de acceso intervenga en la comunicación. Este protocolo nos proporciona una tasa de transferencia mayor que mediante *Bluetooth*, algo muy interesante para nuestra aplicación debido a que vamos a transferir video en tiempo real. El uso de *Wifi Direct* hace que sea necesario que los dos terminales sean compatibles con este protocolo y que lo tengan activado antes de ejecutar la aplicación.

Capítulo 4

Implementación

Vamos a proceder a detallar todos los elementos de la aplicación uno a uno y a explicar los aspectos más importante de cada uno. Se añadirán fragmentos de código para apoyar las explicaciones y para ayudar a entender mejor los problemas que han surgido durante la implementación y cómo se han resuelto.

4.1 Estructura de la aplicación

En primer lugar vamos a mostrar la imagen 4 que representa la estructura completa de la aplicación que nos permite ver cómo se organizan los distintos componentes y que nos ayudará a entender cómo se relacionan entre ellos.

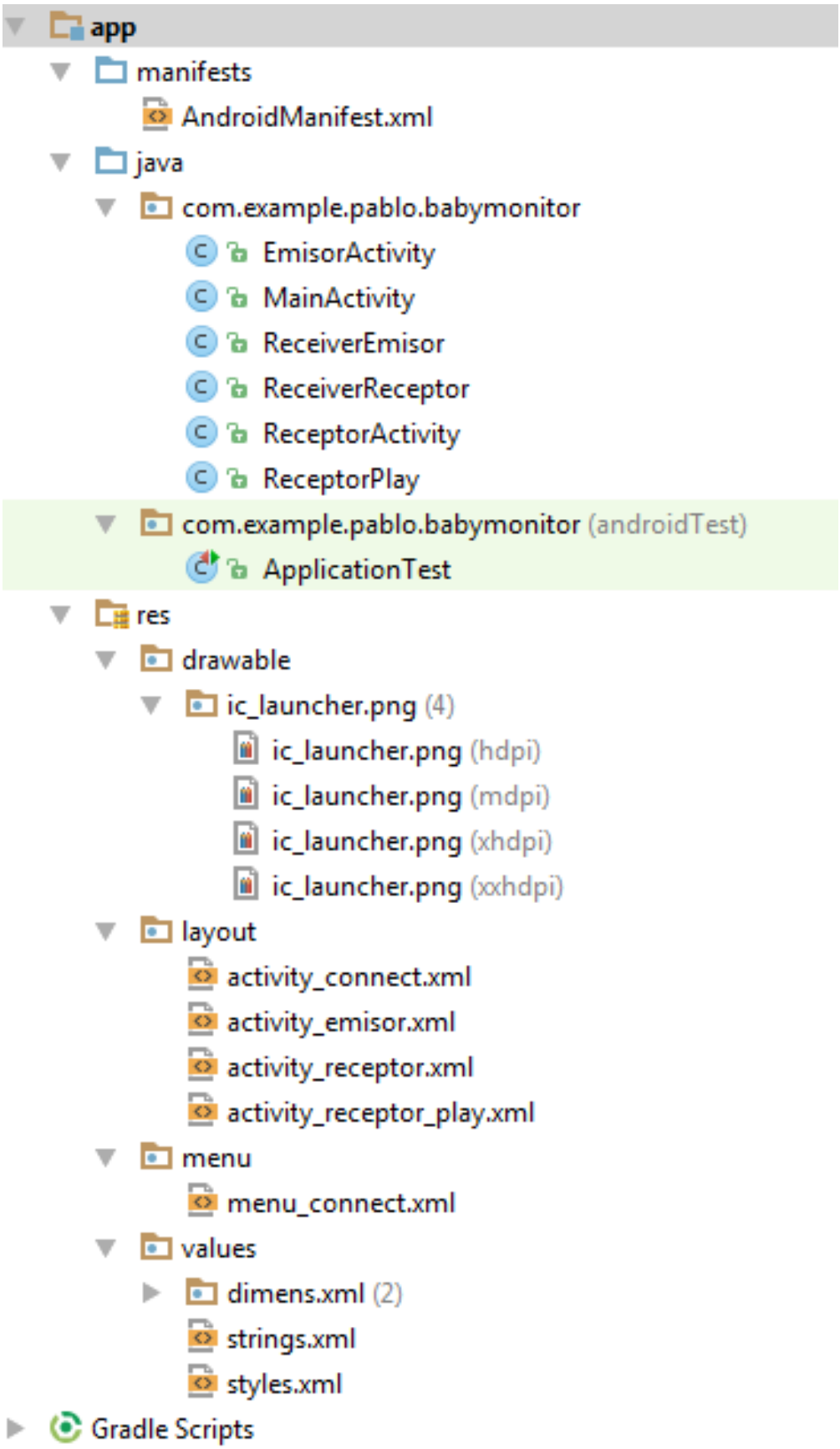

Imagen 4. Estructura Completa

Ésta es la estructura de carpetas desplegadas que componen nuestra aplicación. En la carpeta *manifest* se almacena el archivo *AndroidManifest.xml*, esencial en Android dado que se encarga de la configuración de la aplicación. En este fichero, que podemos observar en la imagen 5, se indican los permisos que le asignamos a la aplicación y la declaración de las distintas actividades que la componen.

```
<uses-sdk android:minSdkVersion="14" />

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="BabyMonitor"
    android:theme="@style/AppTheme">
    <activity
        android:name=".MainActivity"
        android:label="BabyMonitor">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
```

Imagen 5 Fragmento AndroidManifest.xml

En la carpeta *java* se encuentran las distintas clases que forman parte de la aplicación: *EmisorActivity*, *MainActivity*, *ReceiverEmisor*, *ReceiverReceptor*, *ReceptorActivity* y *ReceptorPlay* que posteriormente explicaremos detalladamente. Las clases *EmisorActivity*, *MainActivity*, *ReceptorActivity* y *ReceptorPlay* son las actividades de nuestra aplicación, las cuales tienen asociadas un fichero .xml donde se desarrolla la interfaz gráfica de las pantallas que corresponden a cada actividad. Las otras dos clases, *ReceiverEmisor* y *ReceiverReceptor*, no tienen asociada una interfaz gráfica y se utilizan de apoyo para las tareas de conexión entre los dos terminales.

Por último, aparece la carpeta *res* donde se guardan todos los recursos de la aplicación. En esta carpeta se encuentra otra carpeta llamada *layout*, donde están los ficheros .xml de cada actividad y que definen la interfaz gráfica de la aplicación, además, en esta carpeta res también aparece: la carpeta *drawable*, donde se almacenan las imágenes que podamos necesitar para la aplicación,

la carpeta *menu*, que contiene la descripción de los elementos de los menús, y la carpeta *raw*, que alberga el contenido multimedia.

4.2 Clases de la aplicación

En este apartado explicaremos el código y la interfaz gráfica de cada clase, vamos a organizar el apartado en dos líneas, emisor y receptor. En cada parte detallaremos la interfaz gráfica, las actividades y las clases necesarias para desarrollar cada parte.

Antes de comenzar con la división mencionada, en la imagen 6 vamos a mostrar la clase *MainActivity* con su interfaz gráfica, esta clase no tiene una gran complejidad ya que su única función es la de elegir la función de cada terminal, emisor o receptor.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:background="@drawable/bebe"
    tools:context=".MainActivity">


    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Emisor"
        android:id="@+id/BtnEmisor"
        android:layout_marginRight="46dp"
        android:layout_marginEnd="46dp"
        android:layout_alignParentBottom="true"
        android:layout_toLeftOf="@+id/BtnReceptor"
        android:layout_toStartOf="@+id/BtnReceptor"
        android:layout_marginBottom="76dp" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Receptor"
        android:id="@+id/BtnReceptor"
        android:layout_alignTop="@+id/BtnEmisor"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="63dp"
        android:layout_marginEnd="63dp" />
```

Imagen 6. Interfaz gráfica de MainActivity

Imagen 7. Pantalla de la actividad MainActivity

La imagen 7 muestra la interfaz gráfica de esta actividad, como podemos observar la pantalla se compone de dos botones que se encargan de elegir que rol tendrá cada terminal, en la imagen 8 veremos la implementación de los dos botones.

```
public class MainActivity extends ActionBarActivity {
    private Button btnEmisor;
    private Button btnReceptor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_connect);

        btnEmisor = (Button) findViewById(R.id.BtnEmisor);
        btnReceptor = (Button) findViewById(R.id.BtnReceptor);
        btnEmisor.setOnClickListener((v) -> {
                Intent intent = new Intent(MainActivity.this, EmisorActivity.class);
                startActivity(intent);
        });
        btnReceptor.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, ReceptorActivity.class);
                startActivity(intent);
            }
        });
```

Imagen 8. Código de MainActivity

En este fragmento de código de la implementación se observa perfectamente la función de los dos botones. En primer lugar, se declaran las variables de los botones y los métodos de escucha del evento *clic* del botón. Cuando se hace clic en cualquiera de los botones se crea un *intent* donde se indica la actividad a la que nos llevará ese botón, y una vez que tenemos el *intent* lo utilizamos para llamar al método *startActivity* que se encarga de lanzar la actividad correspondiente.

4.2.2 Clases Emisor

En primer lugar, en la imagen 9, mostramos el archivo .xml asociado a la actividad *EmisorActivity* y que se encarga de la interfaz gráfica de ésta.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.pablo.babymonitor.EmisorActivity">

    <SurfaceView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/surfaceView"
        android:layout_gravity="center"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Imagen 9. Archivo activity_emisor.xml

Observamos que en este fichero sólo hay un elemento de tipo *SurfaceView* que es una superficie de dibujo la cual será utilizada para que el *MediaRecorder* muestre la imagen grabada por la cámara. En la imagen 10 podemos ver la representación de este archivo.

Imagen 10. Representación del archivo activity_emisor.xml

En esta parte de la aplicación encontramos dos clases, la actividad *EmisorActivity* y una clase *ReceiverEmisor* que realiza la función de *Broadcast Listener* y que se encarga de escuchar y manejar los *intents* del Wi-Fi P2p. Un *Broadcast Receiver* es un componente destinado a recibir los evento globales del sistema, cuando se genera uno de estos eventos se llama al método *onReceive* que lleva implementado este componente.

```
public class ReceiverEmisor extends BroadcastReceiver {
    private WifiP2pManager mManager;
    private WifiP2pManager.Channel mChannel;
    private EmisorActivity mActivity;
    WifiP2pManager.PeerListListener peerListListener;

    public ReceiverEmisor(WifiP2pManager manager, WifiP2pManager.Channel channel, EmisorActivity activity){
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }
    public void onReceive(Context context, Intent intent){
        String action = intent.getAction();

        if(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)){
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if(state == WifiP2pManager.WIFI_P2P_STATE_ENABLED){
                Toast toast1 = Toast.makeText(context, "Wifi P2P habilitado", Toast.LENGTH_SHORT);
                toast1.show();
            }else{
                Toast toast2 = Toast.makeText(context, "Error, Wifi P2P no habilitado", Toast.LENGTH_SHORT);
                toast2.show();
                Log.e("LogsAndroid", "error Wifip2p");
            }
        } else if(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)){
            if (mManager != null) {
                mManager.requestPeers( mChannel, peerListListener);
            }

        } else if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)){

        } else if(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)){

        }
    }
}
```

Imagen 11. Implementación ReceiverEmisor

En esta clase *ReceiverEmisor* mostrada en la imagen 11 lo que hacemos es implementar un *Broadcast Receiver* cuya función es recibir los *intents* que se generan cuando se producen cambios en la variables *WIFI_P2P_STATE_CHANGED_ACTION*, *WIFI_P2P_PEERS_CHANGED_ACTION*, *WIFI_P2P_CONNECTION_CHANGED_ACTION*, *WIFI_P2P_THIS_DEVICE_CHANGED_ACTION*. Estas variables nos informan de los cambios relacionados con *Wi-Fi Direct* que se producen cuando la actividad *EmisorActivity* está trabajando. Esta actividad manda los *intents* necesarios a la clase *ReceiverEmisor* que evalúa que evento se ha producido y realiza las acciones necesarias para que la conexión sea exitosa. En este caso las tareas de las que se encarga esta clase son las de comprobar si *Wi-Fi Direct* está encendido e informar al usuario mediante un mensaje. La otra función de esta clase es recibir el evento que se produce al llamar al método *discoverPeers* que se encarga de buscar los dispositivos con *Wi-Fi Direct* activo. Una vez recibido este evento, el *ReceiverEmisor* llama a la función *requestPeers* que genera la lista de dispositivos disponibles para que la actividad *EmisorActivity* pueda consultarla.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_emisor);

    surface = (SurfaceView) findViewById(R.id.surfaceView);
    surface.getHolder().addCallback(this);
    surface.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    mManager = (WifiP2pManager)getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel =  mManager.initialize(this, getMainLooper(), null);
    mReceiver = new ReceiverEmisor(mManager,  mChannel,this);
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
```

Imagen 12. Primer fragmento de la clase EmisorActivity

Ahora ya pasamos a desglosar la implementación de la clase *EmisorActivity*, en este primer fragmento mostrado en la imagen 12 lo que hacemos es definir el objeto *SurfaceView* que es el encargado de mostrar lo capturado por la cámara. También definimos los objetos necesarios para gestionar la conexión p2p mediante *Wi-Fi Direct*. En esta clase construimos el objeto *mManager*, manejador de la conexión, el canal para la comunicación y el escuchador de los eventos globales, la clase *ReceiverEmisor* detallada anteriormente.

Por último, configuramos los *intents* para informar al *ReceiverEmisor* de los eventos globales del sistema generados por el protocolo *Wi-Fi Direct*.

```
mManager.discoverPeers( mChannel, new WifiP2pManager.ActionListener() {
    public void onSuccess() {
        Toast toast1 = Toast.makeText(getApplicationContext(), "Busqueda exitosa", Toast.LENGTH_SHORT);
        toast1.show();
    }

    public void onFailure(int reasonCode) {
        Toast toast1 = Toast.makeText(getApplicationContext(), "Error en la busqueda", Toast.LENGTH_SHORT);
        toast1.show();
        Log.e("LogsAndroid", "error discover peers");
    }
});
WifiP2pManager.PeerListListener peerListListener = new WifiP2pManager.PeerListListener() {
    @Override
    public void onPeersAvailable(WifiP2pDeviceList peersList) {
        peers.clear();
        peers.addAll(peersList.getDeviceList());
    }
};
WifiP2pDevice device = (WifiP2pDevice) peers.get(0);
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
```

Imagen 13. Segundo fragmento de la clase EmisorActivity

En la parte de la clase que vemos en la imagen 13 se buscan los dispositivos disponibles para realizar la conexión p2p. Para eso, se llama al método *discoverPeers*. Este método que se encarga de buscar los dispositivos para realizar la conexión solo informa de si la búsqueda ha

sido exitosa o no, luego para obtener la lista de dispositivos tenemos que escuchar el evento global generado en la variable *WIFI_P2P_PEERS_CHANGED_ACTION* y en el *ReceiverEmisor* obtener esta lista mediante el método *requestPeers* que mediante el método *OnPeersAvailable* notificará a la actividad de que hay una lista de dispositivos disponible. Dentro de este método obtenemos dicha lista y construimos los objetos *WifiP2pDevice* y *Wifip2pConfig* para proceder a establecer la conexión.

```
mManager.connect( mChannel, config, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        mManager.requestConnectionInfo( mChannel, new WifiP2pManager.ConnectionInfoListener() {
            @Override
            public void onConnectionInfoAvailable(WifiP2pInfo info) {
                InetAddress address = info.groupOwnerAddress;
                try{
                    Socket socket = new Socket();
                    int port = 8080;
                    socket.bind(null);
                    socket.connect(new InetSocketAddress(address, port));


                    ParcelFileDescriptor pfd = ParcelFileDescriptor.fromSocket(socket);
                    mMediaRecorder = new MediaRecorder();
                    mCamera.unlock();
                    mMediaRecorder.setCamera(mCamera);
                    mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
                    mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
                    mMediaRecorder.setOutputFormat(8);
                    mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
                    mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.DEFAULT);
                    mMediaRecorder.setOutputFile(pfd.getFileDescriptor());
                    mMediaRecorder.setPreviewDisplay(surface.getHolder().getSurface());
                    mMediaRecorder.prepare();
                    mMediaRecorder.start();
```

Imagen 14. Tercer fragmento de la clase EmisorActivity

Una vez llegados a este punto comienza el código que aparece en la imagen 14 donde se llama al método *connect* que establecerá la conexión. Este método nos notificará si la conexión ha sido exitosa, en ese caso, nos disponemos a abrir un socket entre los dos dispositivos, para ello obtenemos la dirección del otro dispositivo y abriremos el socket. Ahora nos disponemos a grabar y enviar el video al otro terminal.

El primer paso es crear un objeto *ParcelFileDescriptor* que es el que usaremos para enviar el streaming. Después creamos el objeto *MediaRecorder*, que será utilizado para capturar el video y guardarlo en el *ParcelFileDescriptor*. Para ello, desbloqueamos la cámara, configuramos las características del video y llamamos a los métodos prepare y start para iniciar la grabación.

En la imagen 15 podemos observar cómo funciona el objeto *MediaRecorder*.

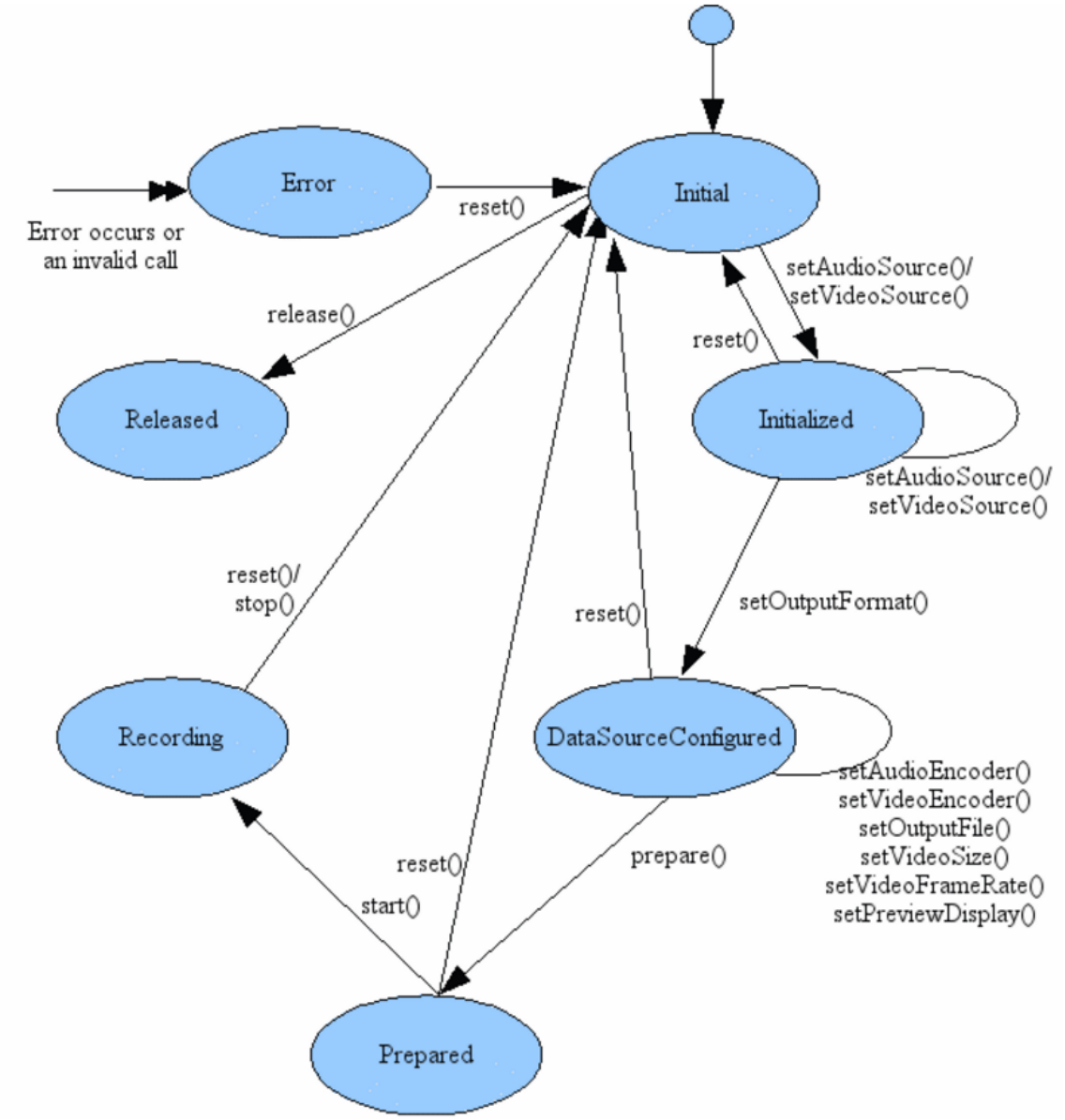

Imagen 15. Máquina de estados MediaRecorder. Imagen obtenida de la API de Android

4.2.3 Clases Receptor

En esta segunda parte de la aplicación, tenemos dos actividades, *ReceptorActivity* y *ReceptorPlay*, por lo tanto, también tenemos dos archivos .xml para las interfaces gráficas de las actividades, y además, una clase *ReceiverReceptor* que, como en la parte del emisor hace la función del *Broadcast Receiver* que ha sido explicada anteriormente.

En primer lugar, en la imagen 16 se va a mostrar el archivo *activity_receptor.xml*, que es el que se encarga de la interfaz gráfica. Como vamos a observar es una actividad muy sencilla en el aspecto gráfico ya que solo tenemos un elemento *TextView* en el que aparecerá un mensaje que informa al usuario de que se están llevando a cabo las tareas de conexión.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.pablo.babymonitor.ReceptorActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Conectando..."
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Imagen 16. Fichero activity_receptor.xml

Imagen 17. Representación del fichero activity_receptor.xml

A continuación, vamos a detallar la implementación de la actividad asociada a esta representación gráfica mostrada en la imagen 17.

```
public class ReceptorActivity extends ActionBarActivity {
    TextView textView;
    WifiP2pManager mManager;
    WifiP2pManager.Channel mChannel;
    BroadcastReceiver mReceiver;
    IntentFilter mIntentFilter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receptor);

        textView = (TextView)findViewById(R.id.textView);

        mManager = (WifiP2pManager)getSystemService(Context.WIFI_P2P_SERVICE);
        mChannel = mManager.initialize(this, getMainLooper(), null);
        mReceiver = new ReceiverReceptor(mManager,mChannel,this);
        mIntentFilter = new IntentFilter();
        mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
        mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
        mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
        mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    }
    protected void onResume(){
        super.onResume();
        registerReceiver(mReceiver, mIntentFilter);
    }
    protected void onPause(){
        super.onPause();
        unregisterReceiver(mReceiver);
    }

}
```

Imagen 18. Código de la clase ReceptorActivity

En esta actividad observada en la imagen 18 lo que hacemos es definir los objetos *mManager, mChannel y mReceiver*, estos son los necesarios para llevar a cabo la conexión a través del *Wi-Fi Direct*. Después de esto, lo que se lleva a cabo es la configuración de los *intents* para que el *ReceiverReceptor* pueda manejar los eventos globales que se produzcan sobra las variables *WIFI_P2P_STATE_CHANGED_ACTION, WIFI_P2P_PEERS_CHANGED_ACTION, WIFI_P2P_CONNECTION_CHANGED_ACTION, WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.*

```
public class ReceiverReceptor extends BroadcastReceiver {

    private WifiP2pManager mManager;
    private WifiP2pManager.Channel mChannel;
    private ReceptorActivity mActivity;

    public ReceiverReceptor( WifiP2pManager manager, WifiP2pManager.Channel channel,ReceptorActivity activity){
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    public void onReceive(Context context, Intent intent){
        String action = intent.getAction();

        if(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)){
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if(state == WifiP2pManager.WIFI_P2P_STATE_ENABLED){
                Toast toast1 = Toast.makeText(context, "Wifi P2P habilitado", Toast.LENGTH_SHORT);
                toast1.show();
            }else{
                Toast toast2 = Toast.makeText(context, "Error, Wifi P2P no habilitado", Toast.LENGTH_SHORT);
                toast2.show();
                Log.e("LogsAndroid", "error wifip2p receptor");
            }
        } else if(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)){

        } else if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)){
            if (mManager == null) {
                return;
            }

            NetworkInfo networkInfo = (NetworkInfo) intent
                    .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);

            if (networkInfo.isConnected()) {
                Intent intent2 = new Intent(mActivity, ReceptorPlay.class);
                mActivity.startActivity(intent2);
            }


        } else if(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)){
```

Imagen 19. Código de la clase ReceiverReceptor

En la imagen 19 podemos observar que la estructura de esta clase es la misma que la del *ReceiverEmisor*, se implementa el método *onReceive* que será llamado cuando se produzca algún evento global del sistema, la diferencia en esta clase está en la condición que determina si la variable *WIFI_P2P_CONNECTION_CHANGED_ACTION*, en la siguiente imagen podemos observarle más claramente.

```
} else if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)){
    if (mManager == null) {
        return;
    }

    NetworkInfo networkInfo = (NetworkInfo) intent
            .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);

    if (networkInfo.isConnected()) {
        Intent intent2 = new Intent(mActivity, ReceptorPlay.class);
        mActivity.startActivity(intent2);
    }
```

Imagen 20. Fragmento del código de la clase ReceiverReceptor

Este fragmento del código de la imagen 20 se ejecutará cuando cambie la variable *WIFI_P2P_CONNECTION_CHANGED_ACTION*, en este caso lo que hacemos es comprobar si ya se ha establecido la conexión y si los dos terminales se conectan correctamente se crea un *intent* para lanzar la actividad *ReceptorPlay* a través del método *startActivity*.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.pablo.babymonitor.ReceptorPlay">

    <SurfaceView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/surfaceView2"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Imagen 21. Archivo activity_receptor_play

Imagen 22. Representación del archivo activity_receptor_play

La de la imagen 21 es interfaz gráfica de la clase *ReceptorPlay*, al igual que en la de la clase *EmisorActivity*, solo hay un elemento del tipo *SurfaceView* donde se mostrará el video grabado y enviado por el otro terminal.

Por último, se va mostrar y explicar la implementación de la clase *ReceptorPlay*.

```
public class ReceptorPlay extends ActionBarActivity implements  SurfaceHolder.Callback{
    MediaPlayer mMediaPlayer;
    SurfaceView surface;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receptor_play);

        surface = (SurfaceView)findViewById(R.id.surfaceView);
        surface.getHolder().addCallback(this);
        surface.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        try {
            ServerSocket serverSocket = new ServerSocket(8080);
            Socket client = serverSocket.accept();

            ParcelFileDescriptor pfd = ParcelFileDescriptor.fromSocket(client);
            mMediaPlayer = new MediaPlayer();
            mMediaPlayer.setDataSource(pfd.getFileDescriptor());
            mMediaPlayer.setDisplay(surface.getHolder());
            mMediaPlayer.prepare();
            mMediaPlayer.start();
```

Imagen 23. Código de la clase ReceptorPlay

En la imagen 23 observamos que lo primero que se hace en esta clase es iniciar el objeto *SurfaceView* y ponerlo en marcha para mostrar el streaming de video recibido, el siguiente paso es iniciar el socket en el mismo puerto que el emisor y llamar al método *accept* para esperar la petición de conexión del otro terminal. Con el método *accept*, la actividad se queda en espera hasta que se realiza la conexión, una vez conectados se crea un objeto *ParcelFileDescriptor* dónde se encontrarán los datos que se envían por el socket y se crea e inicia el objeto *MediaPlayer* que se encarga de reproducir el video a partir del *ParcelFileDescriptor* mediante el método *setDataSource*. Una vez establecido el origen de los datos a reproducir, se indica que se van a mostrar en la *SurfaceView* con el método *setDisplay*, con la configuración completada se llama a los métodos *prepare* y *start* para iniciar la reproducción, a continuación en la imagen 24 se muestra un esquema del funcionamiento del *MediaPlayer*.

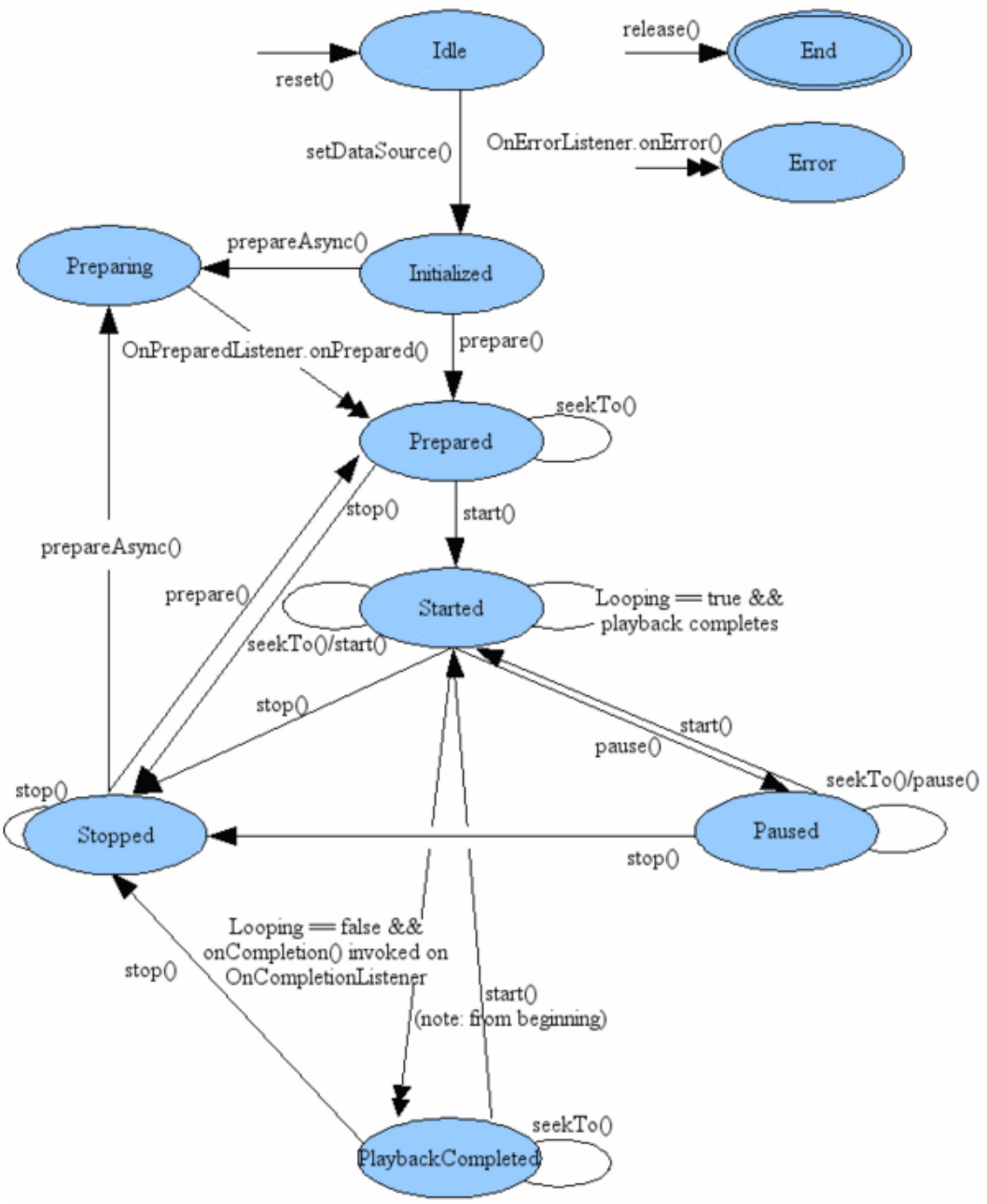

Imagen 24. Máquina de estados del MediaPlayer. Imagen obtenida de la API de Android

Capítulo 5

Manual de usuario

Este capítulo de la memoria es una guía para que el usuario tenga un manual para la correcta utilización de la aplicación, ya que, al ser una versión beta se necesita tener varias cosas en cuenta a la hora de utilizarla. Antes de empezar debemos aclarar que al estar en fase de desarrollo, la aplicación no se puede encontrar en *Google Play Store* y, por tanto, no sólo debemos indicar cómo utilizar la aplicación sino también cómo instalarla.

5.1 Instalación

Para instalar una aplicación que no se encuentra en el *Google Play Store* hay que seguir unos pasos previos que te permitan la instalación:

1. En primer lugar, hay que conseguir el archivo .apk que será el que ejecutaremos para instalar la aplicación, para ello, debemos guardarlo en la memoria del dispositivo desde el ordenador o descargándolo desde la red
2. El segundo paso es ir al menú de Ajustes -> Seguridad y hay que marcar la pestaña de Orígenes Desconocidos que se encuentra en el apartado de Administración de dispositivos. Esto permite instalar aplicaciones que no provienen del *Google Play Store.* Ver imagen 25.
3. Por último, si el dispositivo no tiene un explorador de archivos debemos instalarlo. A continuación, buscamos donde está almacenado el archivo .apk de la aplicación y lo ejecutamos.

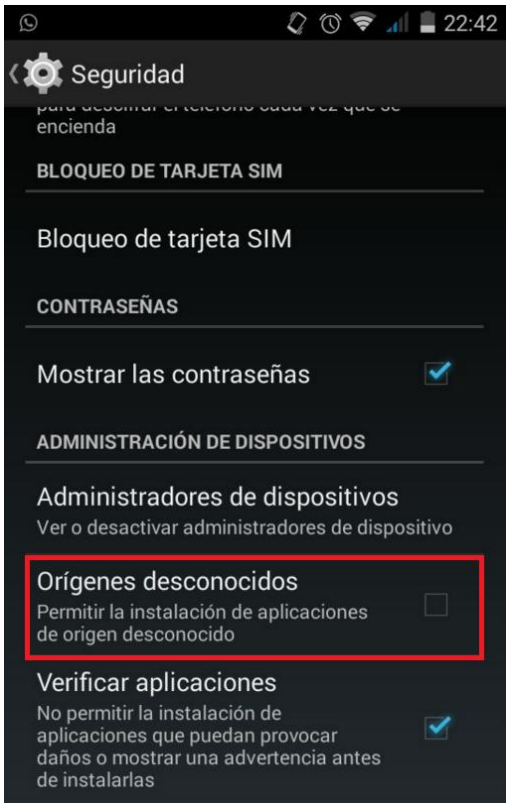

Imagen 25. Captura del menú Ajustes->Seguridad

5.2 Utilización

Antes de arrancar la aplicación debemos tener en cuenta que para usarla nuestro terminal debe ser compatible con *Wi-Fi Direct*, y en ese caso activarlo. Para la realizar esta comprobación debemos entrar en Ajustes-> Conexiones inalámbricas y marcar la casilla de *Wifi Direct*.

Imagen 26. Pantalla de la aplicación

Cuando arrancamos la aplicación aparece esta pantalla de la imagen 26 en la que observamos dos botones, pulsaremos el botón *Emisor* para el terminal que se encarga de grabar. Por otro lado, se pulsará el botón *Receptor* para el terminal que se encarga de recibir el video streaming y reproducirlo.

Imagen 27. Captura de la aplicación

En último lugar, durante la conexión entre los terminales, en el dispositivo que actuará como receptor aparecerá un cuadro de diálogo mientras estamos en la pantalla que se muestra en la imagen 27. Este cuadro nos pregunta si queremos aceptar la petición de conexión *Wi-Fi Direct*, aceptamos y si la conexión se realiza de manera exitosa, la grabación, el envío y la reproducción comenzarán automáticamente.

Capitulo 6

Conclusión y futuras mejoras

Como conclusión personal me gustaría resaltar que la realización de esta aplicación ha sido un reto para mí que, aunque duro, ha sido muy interesante y motivador.

Elegí trabajar en la programación Android en mi Trabajo Fin de Grado porque era algo nuevo para mí que me parecía que complementaba y enriquecía los conocimientos adquiridos durante estos cuatro años. Sabía que el lenguaje sería muy similar al Java, estudiado previamente en el Grado, lo que me ayudó mucho a suavizar la curva de aprendizaje que normalmente suele ser más abrupta al abordar un proyecto como este en una plataforma nueva para mí.

La mayor dificultad que he encontrado en la realización de esta aplicación ha sido como estructurarla, la decisión de organizar el código en más o menos clases, incluir bloques de código en métodos para mejor organización de éste... Estas cuestiones se me han plateado porque soy un programador bastante inexperto y por eso mi estilo no se ajuste mucho al concepto de buena programación, pero como nota positiva me quedo con que este trabajo ha despertado en mí el gusto por la programación y quiero seguir mejorando en esta disciplina.

Por último, me gustaría dejar constancia de que esta aplicación es una versión beta y reconocer que me hubiera gustado presentar un trabajo más maduro, más detallista y más pulido pero debido a factores ajenos he tenido que finalizar el proyecto antes de lo previsto.

Como futuras mejoras se plantean las siguientes:

- Mejorar la interfaz gráfica.
- Mejorar la gestión y la comunicación de los errores que se puedan producir, sobre todo en el proceso de conexión entre los dos terminales.
- Añadir el código que permita habilitar el *Wi-Fi Direct* sin que el usuario tenga que ocuparse de eso previamente
- Añadir funcionalidades a la reproducción y grabación del video como, por ejemplo, botones de pausar, parar y reanudar.

Capitulo 7

Bibliografía

https://developer.android.com. (s.f.).

https://es.wikipedia.org/wiki/Android. (s.f.).

www.android.com. (s.f.).