Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Cliente Android para recepción de imágenes JPEG y detección de patrones

Autor: Pedro Cano Vázquez

Tutor: Antonio Jesús Sierra Collado

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Cliente Android para recepción de imágenes JPEG y detección de patrones

Autor:
Pedro Cano Vázquez

Tutor:
Antonio Jesús Sierra Collado
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado: Cliente Android para recepción de imágenes JPEG y detección de patrones

Autor: Pedro Cano Vázquez

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia

Agradecimientos

La realización de este trabajo supone el final de mis estudios de Grado. Simplemente quiero dar las gracias a todas las personas que han depositado su confianza en mí. Especialmente a mis padres, que han trabajado duro durante estos años para que yo pudiese estudiar.

Gracias también a mi tutor, Antonio Jesús Sierra Collado, por guiarme en la realización de este trabajo. En general, gracias a todos mis profesores por haberme formado para ser ingeniero.

Pedro Cano Vázquez

Sevilla, 2015

Resumen

En la actualidad, los teléfonos móviles son tan utilizados para acceder a Internet desde cualquier lugar como para realizar llamadas telefónicas. Es por ello que las redes de datos que conectan los Smartphones a Internet se están desarrollando a gran velocidad. Desde que apareció GSM (*Global System for Mobile* Communications) con una pobre tasa de 9.6 Kbps, hasta la actualmente implantada HSDPA (*High Speed Downlink Packet Access*) con tasas de hasta 14.4 Mbps o la futura LTE (Long-Term Evolution) con tasas de hasta 100 Mbps y que ya está implantada en algunas zonas de España.

De igual manera, los Sistemas Operativos para Smartphones han evolucionado a una velocidad de vértigo en la última década. El utilizado por la mayoría de los fabricantes es Android, debido a que es un Sistema Operativo de código libre perteneciente a una de las empresas más cotizadas del momento, Google.

En este proyecto se lleva a cabo el diseño, implementación y prueba de una aplicación para Smartphones con Sistema Operativo Android. La función principal de esta aplicación es comunicarse con un servidor de vídeo con tecnología JMF (Java Media Framework) a través de la red HSDPA para descargar las imágenes capturadas por el servidor en el momento de cada petición. De esta manera se obtiene una secuencia de imágenes que se mostrará en la pantalla del Smartphone. La fluidez de la secuencia dependerá del tamaño de las imágenes y de la cobertura de red 3G disponible. Además, se utiliza la librería OpenCV (Open source Computer Vision). Esta librería de visión artificial permite agregar la funcionalidad de reconocimiento de patrones a la aplicación. En concreto, se utiliza para el reconocimiento facial en las imágenes descargadas del servidor. Por último, se estudia una aplicación práctica para este proyecto, un servicio de videovigilancia de bajo coste. Para ello, se utiliza el dispositivo Raspberry Pi, un ordenador de tamaño reducido. La antigüedad de la tecnología JMF obliga además a buscar un servidor alternativo para poner en práctica este apartado, que será MJPEG-Streamer. Tras las pruebas en un escenario real, se comparan los resultados obtenidos utilizando ambos servidores y además se comparan con los resultados obtenidos en un proyecto anterior [2] que utilizaba la red GPRS (General Packet Radio Service) y la tecnología J2ME (Java 2 Micro Edition).

Abstract

Nowadays, mobile phones are used to access to the Internet as much as for calls. Because of this fact, the data networks which connect Smartphones to the Internet are being improved rapidly. Since the first GSM (Global System for Mobile Communications) architecture with a 9.6 Kbps rate to the current HSDPA (High Speed Downlink Packet Access) network with rates until 14.4 Mbps or the oncoming LTE (Long-Term Evolution) network with rates until 100 Mbps and which is already installed in some areas of Spain.

In the same way, the Smartphone Operating Systems have evolved too fast in the last decade. Android is used by the majority of Smartphone manufacturers because it is an open source Operating System from Google which is currently one of the most quoted enterprises.

In this proyect, we carry out the design, implementation and test of an application for Android OS Smartphones. The main function of this application is to communicate with a JMF (Java Media Framework) video server through HSDPA net to download the captured images in each request moment. In this way, we get a sequence of images which will be shown on the Smartphone screen. The sequence fluidity depends on the images size and the available 3G coverage. Moreover, the OpenCV (Open source Computer Vision) library is used in this proyect. This computer vision library allows to add pattern recognition to the application. Specifically, it is used for face recognition in downloaded images. Finally, we are going to study a practical implementation for the proyect which is a low cost video monitoring service. In order to do this, we use Raspberry Pi, which is a small computer. Furthermore, we will need to look for an alternative video server because of the fact that JMF is an old technology. The new video server will be MJPEG-Streamer. After testing the practical implementation, the results for both servers will be compared with eachother and with the results of a previous proyect [2] which use GPRS (General Packet Radio Service) net and J2ME (Java 2 Micro Edition) technology.

Índice

Índice de Tablas

Índice de Figuras

Acrónimos

3GPP	3erd Generation Partnership Project
8PSK	Eight-Phase Shift Keying
AMC	Adaptive Modulation and Coding
AMPS	Advanced Mobile Phone System
API	Application Programming Interface
ARM	Advanced RISC Machine
ART	Android Runtime
BSD	Berkeley Software Distribution
BSC	Base Station Controller
BTS	Base Transceiver Station
CDMA	Code Division Multiple Access
DCS	Digital Cellular System
DS-CDMA	Direct-Sequence Code Division Multiple Access
DVM	Dalvik Virtual Machine
EDGE	Enhanced Data Rates for Global Evolution
FDD	Frequency-division duplexing
GGSN	Gateway GPRS Support Node
GMSK	Gaussian Minimun Shift Keying
GPIO	General Purpose Input Output
GPRS	General Packet Radio Services
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications

HARQ	Hybrid Automatic Repeat request
HSCSD	High-Speed Circuit-Switched Data
HSDPA	High Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
J2ME	Java 2 Micro Edition
JDK	Java Development Kit
JIMI	Java Image Management Interface
JMF	Java Media Framework
JRE	Java Runtime Enviroment
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MSC	Mobile service Switching Center
NDK	Native Development Kit
PCS	Personal Communications Service
PDC	Personal Digital Cellular
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random Access Memory
RLC	Radio Link Control
SDK	Software Developers Kit
SGSN	Serving GPRS Support Node
SSH	Secure SHell
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTRA	Universal Terrestrial Radio Access
UTRAN	Universal Terrestrial Radio Access Network

UVC	USB Video Class
W-CDMA	Wideband Code Division Multiple Access

1. INTRODUCCIÓN

En la actualidad, el sistema operativo para Smartphones dominante es Android. Su rápido crecimiento desde que se lanzó la primera versión se debe a su gran comunidad de programadores, su amplia documentación y su licencia de código abierto. Dispone de un gran número de aplicaciones en su mercado virtual, *Google Play Store*. La filosofía de los Smartphones es la de un terminal móvil conectado siempre a Internet. Esto ha favorecido la evolución y desarrollo de las redes de datos móviles, para proporcionar a los usuarios tasas de descarga cada vez más altas, retardos muy pequeños y la mayor cobertura posible.

En este proyecto se lleva a cabo el diseño, implementación y prueba de una aplicación para Android. La función principal de esta aplicación será comunicarse con un servidor de vídeo para mostrar al usuario las imágenes capturadas por el servidor en el momento de cada petición, obteniendo una secuencia de imágenes que se irá mostrando por pantalla conforme sean descargadas. Además, se utiliza una librería de visión artificial para agregar la funcionalidad de reconocimiento de patrones a la aplicación. En concreto, la librería se utilizará para el reconocimiento facial en las imágenes descargadas del servidor. Finalmente, se estudia una aplicación práctica para el proyecto, un servicio de videovigilancia de bajo coste. Se analizan las ventajas y desventajas de la idea propuesta y se pone a prueba en un escenario real.

Este primer capítulo contiene los objetivos que se desean cumplir con la realización del proyecto, además de mostrar los componentes y arquitectura del sistema estudiado. Por último, se muestra un esquema de la organización de esta memoria.

1.1. Objetivos

A continuación se marcan los objetivos iniciales del proyecto:

- El objetivo principal de este proyecto es la implementación de una aplicación Android que utilizará la red HSDPA (High Speed Downlink Packet Access) para comunicarse con un servidor de vídeo, obteniendo un flujo de imágenes estáticas que serán mostradas al usuario de la aplicación.
- Se desea estudiar el rendimiento de la aplicación en la descarga de imágenes (tiempo medio entre imágenes consecutivas) para comparar estos resultados con los obtenidos en un proyecto anterior [2], y comprobar la evolución con respecto al uso de la red GPRS (General Packet Radio Services) para la descarga de datos.
- Proporcionar la funcionalidad de reconocimiento de patrones a la aplicación. Para ello, se desea utilizar la librería de visión artificial OpenCV (Open source Computer Vision). En concreto, el objetivo es que la aplicación pueda reconocer patrones faciales en las imágenes descargadas del servidor.
- Diseño y prueba de una aplicación práctica para el proyecto. Un servicio de videovigilancia de bajo coste, especialmente dirigido a propietarios de pequeños establecimientos y empresas. Se desea probar en un escenario real y analizar las ventajas y desventajas del mismo.

1.2. Componentes del sistema

El escenario inicial del proyecto es un caso Cliente-Servidor. El servidor, una vez ejecutado, permanece a la espera de peticiones de clientes. Cuando llega una petición, el servidor envía la respuesta oportuna a dicha

petición. Las peticiones del cliente pueden variar dependiendo de los parámetros pasados en la petición, al igual que la respuesta puede variar en función de la petición realizada. A continuación se describen las funciones de ambos componentes del proyecto:

- **Servidor de vídeo:** El servidor de vídeo utilizado ha sido extraído de un antiguo proyecto fin de carrera [11]. Este servidor utiliza las tecnologías JMF (Java Media Framework) para la captura de imágenes y audio mediante dispositivos de vídeo y JIMI (Java Image Management Interface) para la manipulación de imágenes. Gracias a JIMI, el servidor puede modificar las imágenes antes de enviarlas al cliente. De esta forma, el cliente puede elegir aspectos de la imagen como la resolución o la calidad, afectando por supuesto al tamaño de la misma. Para la aplicación práctica del proyecto, se utiliza un servidor de vídeo distinto. Esto es debido a la incompatibilidad de JMF con Raspberry Pi, que será el dispositivo donde se almacenará el servidor de vídeo en el servicio de videovigilancia. El servidor de vídeo elegido para esta parte del proyecto es MJPG-Streamer. Este servidor, de código libre, obtiene las imágenes capturadas por un dispositivo de vídeo compatible y las envía como respuesta a las peticiones que lleguen de los clientes. En este caso, no disponemos de la flexibilidad de elección de resolución o calidad de imagen, ya que estos parámetros son fijados al ejecutar el servidor y no pueden cambiar durante la ejecución.
- **Cliente Android:** Será el encargado de acceder al servidor y descargar las imágenes para mostrarlas al usuario. Este proceso se llevará a cabo a través de las redes de datos móviles de tercera generación, en concreto HSDPA. Dependiendo del servidor utilizado, el usuario podrá cambiar algunos parámetros de las imágenes o no. Además, incluye la funcionalidad de detección facial utilizando la librería de visión artificial OpenCV. La aplicación es compatible con cualquier Smartphone que disponga de la versión 3.0 de Android o superior. Será necesario tener instalada también la aplicación OpenCV Manager, disponible en Google Play, para el correcto funcionamiento de la detección facial.

1.3. Arquitectura del sistema

El sistema está compuesto por dos elementos, un servidor y un cliente. Se utilizan dos servidores distintos, dependiendo del apartado del proyecto donde nos encontremos. En la primera parte, el servidor se ejecuta en un ordenador normal y corriente, mientras que el servidor de la aplicación práctica se ejecutará en Raspberry Pi. El cliente estará instalado en un Smartphone Android con conexión a Internet y la aplicación OpenCV Manager instalada.

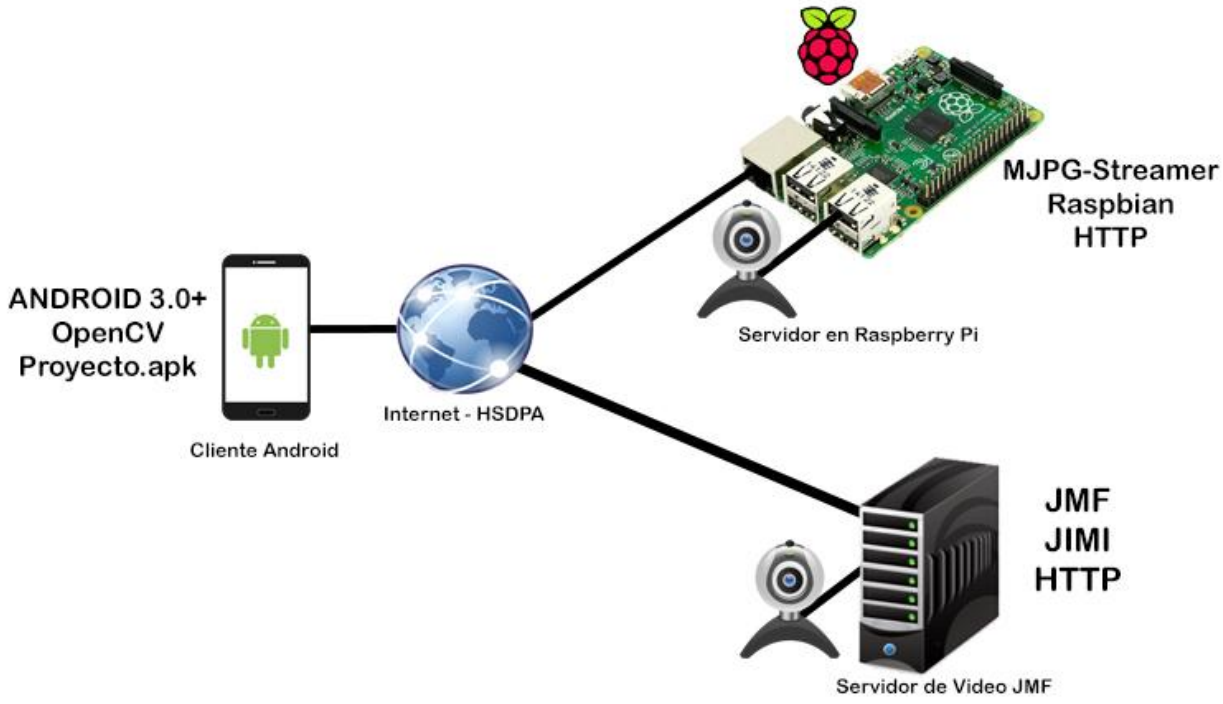

Figura 1. Arquitectura del sistema

En la Figura 1 puede observarse la Arquitectura del sistema. A la izquierda se encuentra el cliente, instalado en un Smartphone con Sistema Operativo Android, en una versión 3.0 del mismo o superior. Además, este dispositivo debe tener instalada la aplicación OpenCV Manager, para poder utilizar las funcionalidades de la librería de visión artificial.

También puede observarse el servidor de vídeo con JMF. Dicho servidor también utiliza la tecnología JIMI para la manipulación de imágenes y HTTP (Hypertext Transfer Protocol) como protocolo del servidor web para atender peticiones.

De igual modo, se observa el dispositivo Raspberry Pi de la aplicación práctica. El Sistema Operativo instalado es Raspbian, una adaptación de Debian para Raspberry Pi. El servidor de vídeo es MJPG-Streamer, que también utiliza el protocolo HTTP para atender peticiones.

Estamos ante un caso típico de servicio cliente-servidor. El cliente realizará peticiones al servidor y éste enviará las respuestas pertinentes. En este caso, el cliente pide al servidor las imágenes que está capturando en el momento de la petición y el servidor responde enviando la imagen estática capturada en ese instante. Tras recibir la imagen, el cliente Android la procesará y le aplicará la detección facial antes de mostrarla por pantalla.

1.4. Esquema de la memoria

La memoria del proyecto está dividida en capítulos. A continuación se puede observar el contenido de cada capítulo, empezando por el capítulo actual:

- **Capítulo 1**. Introducción: En este apartado se encuentran las motivaciones y los objetivos del proyecto. Además se proporciona una visión general de la estructura y componentes del mismo.
- **Capítulo 2**. Tecnología: Esta sección está dedicada al estudio de las tecnologías utilizadas para la realización del proyecto, indicando la implicación de cada una dentro de éste.

- **Capítulo 3**. Trabajo realizado: Aquí se describe el diseño y la implementación del cliente Android desarrollado. También se estudia la aplicación práctica propuesta para el proyecto: qué es necesario; cómo se llevará a cabo; y el presupuesto necesario para el servicio.
- **Capítulo 4**. Resultados obtenidos: En este apartado se muestran las pruebas llevadas a cabo sobre los distintos escenarios propuestos y se comparan los resultados obtenidos para ambos servidores. También se hará una comparación con los resultados obtenidos en otro proyecto donde se realiza la conexión a internet del terminal móvil mediante redes de datos móviles de segunda generación.
- **Capítulo 5**. Conclusiones: En esta sección se plantean las conclusiones de la realización del proyecto, así como las posibles vías futuras de mejora para el mismo.
- **Bibliografía.** Índice de los documentos consultados para la realización del proyecto.
- **Anexos**. Aquí se encuentran los manuales de usuario necesarios para la instalación de los distintos elementos del sistema y el código fuente del cliente Android desarrollado.

2. TECNOLOGÍA

Tras la introducción del proyecto, donde se han planteado los objetivos del mismo y se ha dado una visión global del contexto en el que se encuentra, es necesario exponer la tecnología utilizada para la implementación del proyecto. En este capítulo, se muestran las características y la implicación de cada una de las tecnologías elegidas para la realización del proyecto, así como la importancia que tiene cada una dentro del mismo.

El capítulo comienza con una subsección dedicada a Android, donde se expone su historia y sus características. Después se habla del lenguaje de programación utilizado para la implementación de la aplicación, el lenguaje Java. También se menciona el entorno de desarrollo utilizado, Eclipse. Más tarde se muestran las características de la librería de visión artificial OpenCV (Open source Computer Vision) y los módulos que serán utilizados para el reconocimiento de patrones. Para terminar, se describen: el protocolo HTTP (Hypertext Transfer Protocol), utilizado en la comunicación entre cliente y servidor; los sistemas móviles de tercera generación, que serán el canal de comunicación entre el cliente y el servidor; y el servidor de vídeo utilizado en la primera parte del proyecto.

2.1. Android

2.1.1. Introducción

Android es un sistema operativo inicialmente desarrollado por Android Inc., empresa dedicada a la producción de aplicaciones para terminales móviles. En 2005, Google adquiere la compañía y empieza a trabajar en la creación de la máquina virtual Dalvik (una máquina virtual Java optimizada para móviles). En 2007 se crea el consorcio Open Handset Alliance con el objetivo de desarrollar estándares abiertos para móviles, promoviendo el diseño y la difusión de la plataforma Android. En noviembre de este mismo año se lanza una primera versión del Android SDK (Software Development Kit). El primer móvil con Android (T-Mobile G1) aparece en el año 2008 y en octubre del mismo año, Google libera el código fuente de Android, principalmente bajo licencia de código abierto Apache. Ese mismo mes, se abre Android Market para la descarga de aplicaciones. En abril de 2009, Google lanza la versión 1.5 del Android SDK. A finales de año sale la versión 2.0 y durante el 2010 las versiones 2.1, 2.2 y 2.3 consolidándose como uno de los sistemas operativos para terminales móviles más utilizados. En 2011 se lanzan las versiones 3.x específica para tabletas y 4.0 tanto para móviles como para tabletas. Durante este año, Android alcanza una cuota de mercado superior al 50% convirtiéndose en la plataforma para móviles más importante. En 2012, Google sustituye el Android Market por Google Play Store, un portal donde acceder a la descarga tanto de aplicaciones como de contenidos. En este mismo año aparecen las versiones 4.1 y 4.2 del SDK. En 2013 aparece la versión 4.4 y a finales de 2014 aparece la última versión en la actualidad del Android SDK, la 5.0.

Actualmente, la versión más utilizada es la 4.4 seguida por las versiones 4.1.x y 4.2.x. La distribución de las versiones del SDK de Android puede observarse en la Figura 2 y la Tabla 1.

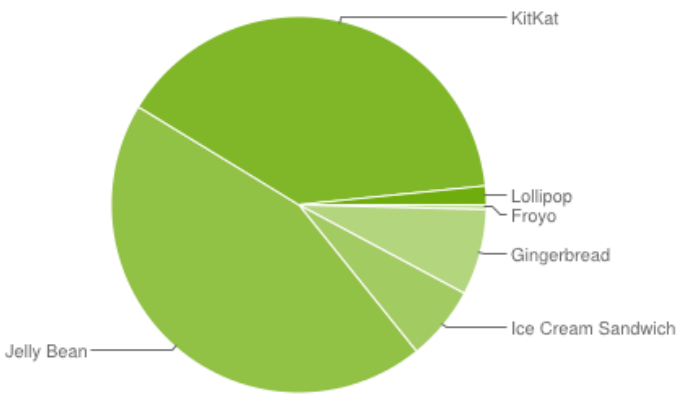

Figura 2. Gráfica de distribución de las versiones del SDK

Versión	Codename	API	Distribución
2.2	Froyo	8	0.4%
2.3.3-2.3.7	Gingerbread	10	7.4%
4.0.3-4.0.4	Ice Cream Sandwich	15	6.4%
4.1.x	Jelly Bean	16	18.4%
4.2.x		17	19.8%
4.3		18	6.3%
4.4	KitKat	19	39.7%
5.0	Lollipop	21	1.6%

Tabla 1. Porcentaje de distribución de las versiones del SDK

Como puede observarse, las versiones de Android para Smartphones más utilizadas en la actualidad son la 4.4 y las 4.1.x y 4.2.x. La última versión, la 5.0, se encuentra aún en expansión, ya que hace menos de un año que apareció.

2.1.2. **Características**

A continuación, se marcan las características principales del Sistema Operativo Android. La mayoría de estas características marcan la diferencia entre Android y los demás Sistemas Operativos actuales de plataformas móviles, explicando así su gran popularidad entre desarrolladores y usuarios:

- Plataforma realmente abierta. El desarrollador puede usar y modificar el sistema de forma gratuita.
- Portabilidad. La mayoría de las aplicaciones son desarrolladas en Java, lo que permite su ejecución en una gran variedad de dispositivos gracias al concepto de máquina virtual.
- Filosofía de dispositivo siempre conectado a internet.
- Gran cantidad de servicios incorporados. Entre otros, localización GPS, bases de datos con SQL, navegador web o reconocimiento y síntesis de voz.
- Seguridad. Los programas se encuentran aislados unos de otros y cada aplicación dispone de una serie de permisos que limitan su rango de actuación.
- Optimizado para dispositivos con pocos recursos.

- Alta calidad de gráficos y sonido. Gráficos vectoriales suavizados, gráficos en 3 dimensiones basados en OpenGL (Open Graphics Library), una librería para gráficos en 2D y 3D. Incluye los codecs estándar más comunes de audio y vídeo como H.264, MP3, AAC, etc.

2.1.3. Arquitectura de Android

La arquitectura de Android está formada por varias capas. En la Figura 3 se muestran estas capas de forma gráfica. En las siguientes subsecciones se describen cada una de las capas de manera individual.

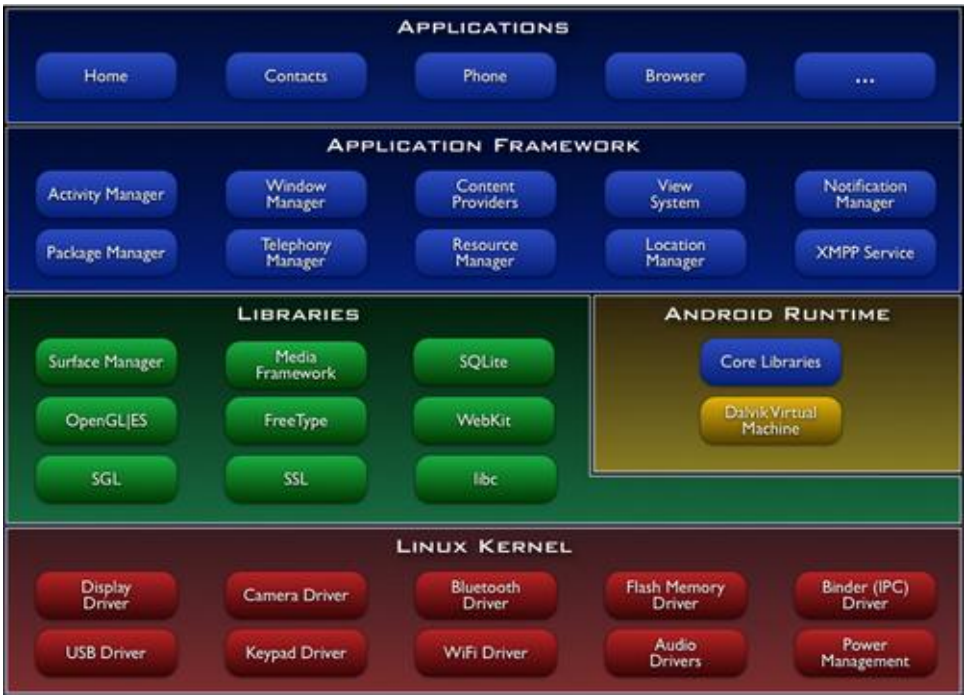

Figura 3. Arquitectura de Android.

2.1.3.1. El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos. Además, actúa como capa de abstracción entre el hardware y el software.

2.1.3.2. Runtime de Android

Está basado en el concepto de máquina virtual utilizado en Java. Teniendo en cuenta las limitaciones en recursos de los dispositivos donde se pensaba ejecutar Android (Poca batería, poca memoria, procesador limitado, pantallas con poca resolución), Google creó la máquina virtual Dalvik para responder mejor a estas carencias.

Está formado por el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java y la máquina virtual Dalvik. Algunas características de la DVM (Dalvik Virtual Machine) son:

- Convierte los ficheros *.class (clase JAVA compilada) a formato ".dex" (Dalvik EXecutable) mediante la herramienta "dx" incluida en el SDK de Android.
- Ejecuta las aplicaciones en formato ".dex" y el código es interpretado en cada ejecución de la aplicación.

- Técnicamente no es una JVM (Java Virtual Machine). No opera con Java bytecode sino que está basada en registros.
- Cada aplicación se ejecuta en su propio proceso Linux con su propia instancia de la DVM.
- Utiliza compilación JIT (Just In Time). Cada vez que se ejecuta la aplicación, se compila el código a medida que va siendo necesaria su utilización. Este tipo de compilación nos permite utilizar poca memoria.

En la versión 4.4 del Android SDK se puede sustituir el uso de DVM por ART (Android Runtime). Este nuevo entorno de ejecución presenta las siguientes características:

- Nuevo formato de compilado. Ficheros ELF (Executable and Linkable Format) con extensión ".oat".
- Utiliza compilación AOT (Ahead Of Time). Se compila la aplicación completa en el momento de la instalación.
- Necesita más memoria y el tiempo de instalación es mayor pero reduce el uso del procesador al ejecutar las aplicaciones, reduciendo el consumo de batería.

A partir de la versión 5.0, ART reemplaza por completo el uso de DVM.

2.1.3.3. Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes. Están compiladas en el código nativo del procesador.

- Sistem C library: una derivación de la librería de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- Media Framework: librería basada en PacketVideo's OpenCORE. Soporta codecs de reproducción y grabación de multitud de formatos de audio, vídeo e imágenes (MPEG4, H.264, MP3, AAC, AMR, JPG y PNG).
- Surface Manager: maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- WebKit: soporta un moderno navegador web utilizado en el navegador Android y en la vista webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- SGL (Scalable Graphics Library): motor de gráficos 2D.
- Librerías 3D: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- FreeType: fuentes en bitmap y renderizado vectorial.
- SQLite: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- SSL: proporciona servicios de encriptación Secure Socket Layer.

2.1.3.4. Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localizacion, servicios, barra de notificaciones, etc.).

Permite la reutilización de componentes de manera más simple. Las aplicaciones pueden publicar sus capacidades y hacer uso de las capacidades de otras aplicaciones.

Los servicios más importantes que incluye son:

- Views: vistas. Parte visual de los componentes.
- Resource Manager: gestiona los elementos que forman parte de la aplicación y que están fuera del código, por ejemplo imágenes.
- Activity Manager: gestiona el ciclo de vida de las aplicaciones y permite navegar entre ellas.

- Notification Manager: gestiona el uso de alertas personalizadas en la barra de estado.
- Content Providers: proporciona acceso a datos de otras aplicaciones.

2.1.3.5. Aplicaciones

En esta capa se ubican las aplicaciones instaladas en el dispositivo Android.

Normalmente, las aplicaciones Android están escritas en Java. Para su desarrollo se utiliza el Android SDK. También pueden desarrollarse aplicaciones utilizando C/C++ haciendo uso del Android NDK (Native Development Kit).

2.2. Java

2.2.1. Introducción

Java es un lenguaje de programación de propósito general y de alto nivel. Es concurrente (un mismo programa puede generar varios procesos que se ejecutan simultáneamente) y orientado a objetos (el programa está compuesto por entidades que interactúan entre ellas por medio de mensajes que modifican su estado).

Fue desarrollado a principios de los 90 por un grupo dirigido por James Gosling, de la empresa Sun Microsystems. El enfoque principal del proyecto era el de crear un lenguaje de programación basado en el paradigma de la orientación a objetos, cuyos programas pudieran ser ejecutados en múltiples sistemas operativos sin necesidad de recompilar el código.

2.2.2. Características de Java

El compilador de Java genera, a partir del código fuente de un programa, un código intermedio llamado bytecode. Este código es interpretado y ejecutado por la JVM instalada en la plataforma destino. De esta forma se consigue la ejecución en cualquier plataforma del programa original sin necesidad de modificarlo.

Las características principales de la programación orientada a objetos son:

- Abstracción. Los objetos se ven entre sí como agentes abstractos que pueden modificar su estado o comunicarse con otros objetos sin revelar cómo es el proceso internamente.
- Encapsulamiento. Los objetos están aislados del exterior, evitando que se acceda a sus datos o se modifique su contenido sin permiso.
- Herencia. Las clases se encuentran relacionadas entre sí, formando una jerarquía de clasificación. Mediante la herencia, los objetos pueden compartir de forma automática los métodos y datos de las clases a las que pertenecen.
- Polimorfismo. Permite hacer uso de varios métodos distintos que comparten el mismo nombre. Se puede diferenciar un método de otro por el parámetro pasado (sobrecarga) o haciendo un método distinto en varias clases, bajo el mismo nombre (reemplazo).

En cuanto a la estructura del código, éste está formado por clases. Una clase es la generalización de un tipo específico de objeto. En una clase se definen las características (atributos) y comportamientos (métodos) de los objetos de esa clase. Los métodos determinan el comportamiento del objeto cuando recibe un mensaje, operando sobre los atributos del propio objeto o enviando mensajes a otros objetos.

Para la ejecución de programas Java, es necesario tener instalado el JRE (Java Runtime Enviroment). Este entorno de ejecución está compuesto por la JVM, clases del núcleo de la plataforma Java y bibliotecas de la plataforma Java de soporte. Para el desarrollo de aplicaciones Java, en cambio, es necesario el JDK (Java Development Kit) que está compuesto por el compilador y depurador Java además de otras herramientas de

programación. El JDK lleva integrado el JRE.

En este proyecto, utilizaremos el lenguaje Java para llevar a cabo la implementación de nuestro cliente Android. Además, tanto el servidor de vídeo que utilizamos como el proyecto del que partimos están escritos en Java.

2.3. Eclipse

2.3.1. Historia

En noviembre de 2001 se creó el Consorcio Eclipse, un proyecto de código abierto formado por las empresas Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft y Webgain bajo la dirección de IBM. A últimos de 2003, este consorcio había crecido hasta estar formado por unos 80 miembros.

El 2 de febrero de 2004, el Consorcio Eclipse pasa a ser independiente de IBM y se convierte en una organización sin ánimo de lucro formada por las empresas: HP, QNX, IBM, Intel, SAP, Fujitsu, Hitachi, Novell, Oracle, Palm, Ericsson y RedHat, además de algunas universidades e institutos tecnológicos. A partir de este momento, el objetivo de la organización es convertir Eclipse en un organismo independiente que impulsará la evolución de la plataforma para beneficiar a los desarrolladores de software y a los usuarios finales.

2.3.2. ¿Qué es Eclipse?

Eclipse es un IDE (Integrated Development Environment). Es un software que proporciona diversas herramientas para facilitar la tarea de los desarrolladores de aplicaciones software en cualquier lenguaje de programación. Entre las distintas herramientas, Eclipse incluye un compilador, un intérprete, un editor de código fuente y un depurador. Está desarrollado por completo en lenguaje java, por lo que es necesario tener un JRE instalado previamente en el sistema.

2.3.3. Características de Eclipse

Eclipse se basa en el uso de plug-ins. Un plug-in es una funcionalidad adicional que debemos instalar si queremos tener acceso a ella. La descarga del entorno básico de Eclipse incluye algunos de los plug-ins más básicos para poder empezar a programar en Java. Si quisiéramos escribir y compilar código en C++ (por ejemplo) deberíamos descargar e instalar el plug-in CDT (C/C++ Development Tooling) de Eclipse. Para la realización de este proyecto, debemos preparar nuestro entorno para poder desarrollar la aplicación Android. Para ello, será necesario descargar e instalar el plug-in ADT (Android Development Tool) desarrollado por Google. Si quiere obtener más información acerca de la instalación del ADT en Eclipse lea el Anexo A. Manuales de Usuario.

En Eclipse, el elemento principal es el *Workbench*. Desde aquí tendremos acceso a todas las herramientas necesarias para llevar a cabo nuestra labor de programación. El *Workbench* está compuesto por una o más perspectivas, donde encontramos editores y vistas como el *Project Explorer* que muestra el contenido de nuestro espacio de trabajo seleccionado (en inglés *workspace*, es el directorio donde estamos trabajando con Eclipse).

Se puede crear un nuevo proyecto desde cero, indicando el lenguaje en el que será desarrollado. Eclipse montará automáticamente el sistema de ficheros de nuestro proyecto para que sea acorde al lenguaje especificado ahorrando dicha tarea al programador. También es posible importar un proyecto existente a nuestro *workspace* y trabajar sobre él.

Otra tarea que se facilita con el uso de Eclipse es la de compilación y ejecución. Para compilar sólo es necesario un click (Build Project). Eclipse comenzará el proceso, alertándonos mediante la Vista de Consola de los errores y avisos que pudiera generar la compilación de nuestro proyecto. Así mismo, podemos ejecutar nuestro proyecto (si compila con éxito primero) mediante un click, indicando cómo debe ejecutarse (aplicación Java, aplicación Android, aplicación C, …).

En este proyecto, Eclipse sirve de gran ayuda para enlazar la librería de OpenCV con nuestra aplicación Android, permitiendo el uso de los métodos y características de la librería para el reconocimiento facial. Los tutoriales que ofrece la propia documentación de dicha librería se basan en el uso de Eclipse, por lo que descartamos el uso de otros IDE para desarrollo de aplicaciones Android, como por ejemplo Android Studio.

2.4. OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de código abierto para la visión artificial. La visión artificial es un subcampo de la inteligencia artificial, cuyo propósito es programar un ordenador para que sea capaz de analizar las características de una imagen o escena.

OpenCV se construyó para proporcionar una estructura común para la visión artificial y para acelerar su uso en los productos comerciales. Siendo un producto bajo licencia BSD (Berkeley Software Distribution), OpenCV hace que utilizar y modificar su código sea fácil para las empresas.

La librería consta de más de 2500 algoritmos optimizados, incluyendo un amplio conjunto de algoritmos clásicos y actuales de visión artificial. Estos algoritmos se pueden utilizar para el reconocimiento facial, identificación de objetos, seguir el movimiento de objetos, identificar imágenes dentro de otra imagen a partir de una base de datos, etc. Con una comunidad de más de 47000 usuarios y más de 7 millones de descargas, esta librería se utiliza en empresas, grupos de investigación y organismos gubernamentales.

Originalmente, OpenCV fue escrita en C++. Actualmente, la librería dispone de interfaces en C++, C, Python, Java y MATLAB y existe soporte para Windows, Linux, Android y Mac OS.

En este proyecto, utilizaremos algunos módulos de la librería para el reconocimiento facial en las imágenes descargadas desde el servidor de vídeo. En concreto, utilizaremos los siguientes módulos:

- Core. Es el módulo principal de la librería. Está compuesto por funciones y estructuras de datos definidas que son utilizadas por los demás módulos.
- Android. Este módulo se utiliza para inicializar los recursos necesarios para utilizar la librería OpenCV y gestionar la cámara del terminal.
- Objdetect. Sirve para la detección y clasificación de los objetos que aparecen en una imagen.

Para la ejecución de una aplicación Android que utiliza la librería OpenCV, es necesario instalar en el terminal la aplicación *OpenCV Manager* proporcionada por OpenCV y accesible desde Google Play.

2.5. HTTP

2.5.1. **Introducción**

El Protocolo de Transferencia de HiperTexto (Hypertext Transfer Protocol) es un protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP. La especificación completa del protocolo HTTP 1/0 está recogida en el RFC 1945. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como el World Wide Web.

2.5.2. **Características y funcionamiento**

Está soportado sobre los servicios de conexión TCP/IP. Durante su funcionamiento, un proceso servidor escucha en un puerto de comunicaciones TCP (por defecto, el 80) y espera las solicitudes de conexión de los clientes Web. Una vez que se establece la conexión, el protocolo TCP se encarga de mantener la comunicación y garantizar un intercambio de datos libre de errores.

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan. Cada objeto Web es conocido por su URL (Uniform Resources Locator).

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME (Multipurpose Internet Mail Extensions). De esta forma, el protocolo puede intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, byte a byte, y la identificación MIME permitirá que el receptor trate adecuadamente los datos.

Principales características de HTTP:

- Permite la transferencia de objetos multimedia. El contenido de cada objeto intercambiado está identificado por su clasificación MIME.
- Tiene 3 métodos principales definidos (además de otros 5 menos utilizados) para identificar la acción que se desea efectuar sobre el recurso identificado.
    - GET: pide una representación del recurso especificado.
    - POST: envía datos al servidor para que sean procesados por el recurso identificado. Los datos van incluidos en el cuerpo de la petición.
    - HEAD: pide una respuesta similar a la correspondiente a una petición GET, pero sin el cuerpo de la respuesta. Es útil para obtener información de encabezado sin necesidad de transportar todo el contenido.
- Cada operación HTTP implica una conexión con el servidor, que es liberada al término de la misma. En la actualidad (mediante el procedimiento HTTP keep-alive), se puede mantener la conexión activa durante un periodo de tiempo, de forma que pueda ser utilizada en sucesivas transacciones.
- Es un protocolo sin estado. Cada petición de un cliente a un servidor es independiente de las anteriores.

El proceso petición-respuesta se da en los siguientes pasos:

- El cliente Web separa las distintas partes de la URL, identificando la dirección del servidor, el puerto (80 por omisión) y el objeto requerido del servidor.
- Se abre una conexión TCP/IP con el servidor, en el puerto correspondiente.
- Se realiza la petición. Para ello, se indica el método (GET, POST, HEAD, …), la dirección del objeto requerido, la versión del protocolo empleada (por lo general HTTP/1.0) y un conjunto variable de información, que incluye datos sobre el navegador, datos opcionales para el servidor, etc.
- El servidor devuelve la respuesta al cliente. La respuesta está formada por un código de estado y el tipo de dato MIME de la información de retorno, seguida de la propia información.
- Se cierra la conexión TCP, a menos que se utilice el modo HTTP Keep Alive.

El diálogo con los servidores HTTP se establece a través de mensajes formados por líneas de texto, cada una

de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la correspondiente respuesta. En la Figura 4 se puede observar el formato de estos mensajes.

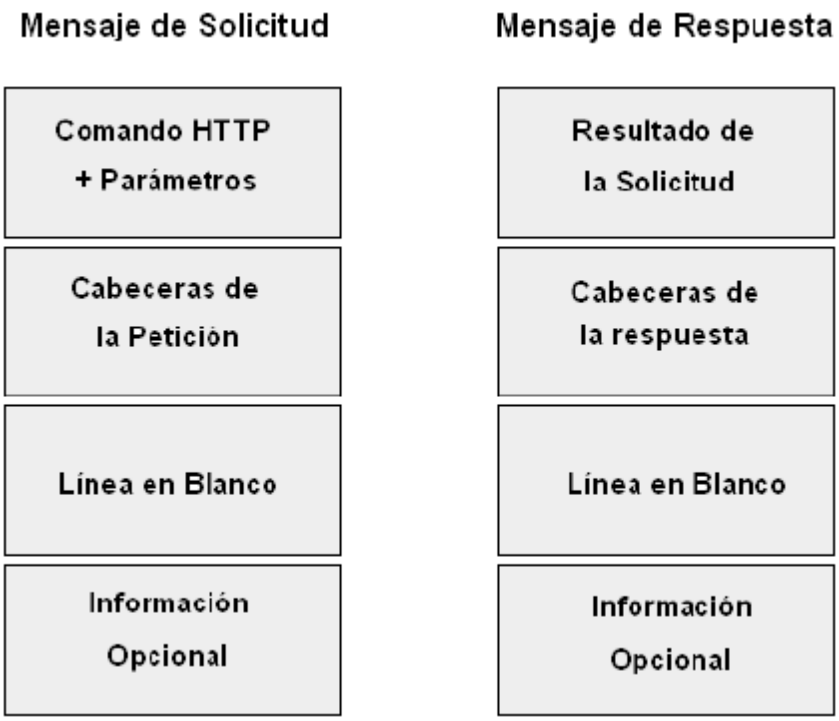

Figura 4. Tipos de mensajes HTTP.

La primera línea del mensaje de solicitud contiene el comando que se solicita al servidor HTTP. La primera línea del mensaje de respuesta contiene el resultado de la operación, que es un código numérico que permite conocer el éxito o fracaso de la operación. Después aparece, para ambos tipos de mensajes, un conjunto de cabeceras (unas obligatorias y otras opcionales) que condicionan y matizan el funcionamiento del protocolo.

La separación entre cada línea del mensaje se realiza con un par CR-LF (retorno de carro más nueva línea). El final de las cabeceras se indica con una línea en blanco, tras la cual se pueden incluir los datos transportados por el protocolo, por ejemplo, el documento HTML (HyperText Markup Language) que devuelve un servidor.

2.5.3. **Comandos del protocolo HTTP**

El protocolo HTTP consta de los siguientes comandos:

1. GET. Se utiliza para recoger cualquier tipo de información del servidor. Como resultado, el servidor HTTP envía el documento correspondiente a la URL solicitada. Este comando puede ir acompañado de una cabecera con los siguientes parámetros:
    - Accept. Indica los formatos de archivo que puede soportar el cliente.
    - Accept-Charset. Indica los conjuntos de caracteres que acepta.
    - Accept-Encoding. Qué tipo de codificación acepta.
    - Accept-Language. En qué lenguaje se hará la respuesta de la cabecera.
    - Authorization. Clave para tener acceso a lugares restringidos donde se requiere nombre de contraseña y el formato de autorización empleado.

- Connection. Especifica opciones requeridas para una conexión.
- Date. Representa la fecha y la hora a la que se envió el comando.
- From. Campo opcional que incluye la dirección de correo electrónico del usuario del cliente.
- Host. Especifica la dirección del host del cliente y el puerto de conexión que ha empleado.
- If-Modified-Since. Condición de que sólo se responda a la solicitud si la fecha de última modificación del objeto es superior a la indicada en su parámetro.
- If-UnModified-Since. Condición de que sólo se responda a la solicitud si la fecha de última modificación del objeto no ha cambiado a partir de la fecha indicada en su parámetro.
- Proxy-Authorization. Identificarse a un proxy.
- Referer. Contiene la URL de la página que le ha dado acceso a la que está solicitando. Esto puede interesarle a los autores de páginas web para saber en qué lugares existen enlaces de su página.
- Transfer-Encoding. Indica el tipo de codificación aplicada del contenido que responderá el servidor.
- Upgrade. Especifica qué protocolos suplementarios soporta el cliente, si también soportara FTP u otros.
- User-Agent. Especifica el nombre del cliente y su versión así como el sistema operativo que utiliza el cliente.
- Via. Sirve para obtener la ruta de routers que ha recorrido el mensaje.

2. HEAD. Es similar al comando GET y puede usar las mismas cabeceras, pero se utiliza para obtener como respuesta del servidor únicamente las cabeceras. Es utilizado por los gestores de cachés de páginas o los servidores proxy, para conocer cuándo es necesario actualizar la copia que se mantiene de un fichero.
3. POST**.** Se usa para hacer peticiones en las que el servidor destino acepta el contenido de la petición como un nuevo subordinado del recurso pedido. El método POST se creó para cubrir funciones como la de enviar un mensaje a grupos de usuarios, dar un bloque de datos como resultado de un formulario a un proceso de datos o añadir nuevos datos a una base de datos, entre otras. La función llevada a cabo por el método POST está determinada por el servidor y suele depender de la URI (Uniform Resource Identifier) de la petición. El resultado de la acción realizada por el método POST puede ser un recurso que no sea identificable mediante una URI.
4. PUT. Permite guardar el contenido de la petición en el servidor bajo la URI de la petición. Si esta URI ya existe, entonces el servidor considera que esta petición proporciona una versión actualizada del recurso. Si la URI indicada no existe y es válida para definir un nuevo recurso, el servidor puede crear el recurso con esa URI. El servidor responde con un código 2xx que indica el resultado de la operación.
5. DELETE. Este comando se usa para que el servidor borre el recurso identificado por la URI de la petición. No se garantiza al cliente que la operación se lleve a cabo aunque la respuesta sea satisfactoria.
6. TRACE. Se usa para saber si existe el receptor del mensaje y usar la información para hacer un diagnóstico. En las cabeceras, el campo "Via" sirve para obtener la ruta que sigue el mensaje. Mediante el campo Max-Forwards se limita el número de pasos intermedios que puede tomar. Esto es útil para evitar bucles entre los proxy.

Cuando el servidor envía la respuesta al cliente, le envía la información solicitada y una cabecera con una serie campos, estos campos son:

- Age. Tiempo transcurrido desde que se creó la respuesta.
- Allow. Especifica qué comandos puede utilizar el cliente respecto al objeto pedido.
- Expires. Fecha en la que caducará el objeto adjunto.
- Last-Modified. Fecha de la última modificación del objeto.
- Location. Se usa para redirigir la petición a otro objeto. Informa sobre la dirección exacta del objeto que se ha accedido.
- Proxy-Authenticate. Indica el esquema de autentificación en caso de que un proxy la requiera.
- Public. Da una lista de los comandos que reconoce el servidor.
- Retry-After. Si no puede ofrecer un objeto provisionalmente, indica la fecha para que se vuelva a hacer la petición.
- Server. Especifica el tipo y versión del servidor.
- Vary. Indica que hay más de una posible respuesta a la petición pero solo escoge una.
- Warning. Específica información adicional sobre el estado de la respuesta.
- WWW-Autenticate. Usado cuando se accede a un lugar restringido, sirve para informar de las maneras de autentificarse que soporta el objeto del servidor.

Además, hay otros parámetros de información de cabeceras que son válidos tanto en los mensajes de petición de los clientes como en los de respuesta del servidor:

- Content-Type. Descripción MIME de la información que contiene el mensaje para dar el tratamiento conveniente a los datos que se envían.
- Content-Length. Longitud en bytes de los datos enviados.
- Content-Encoding. Especifica en qué formato están codificados los datos enviados.
- Date. Fecha local de la operación. Las fechas deben incluir la zona horaria en que reside el sistema que genera la operación. Por ejemplo: Sunday, 12- Dec-96 12:21:22 GMT+01. No existe un formato único en las fechas; incluso es posible encontrar casos en los que no se dispone de la zona horaria correspondiente, con los problemas de sincronización que esto produce. Los formatos de fecha a emplear están recogidos en los RFC 1036 y 1123.
- Pragma. Permite incluir información variada relacionada con el protocolo HTTP en la petición o respuesta que se está realizando. Por ejemplo, un cliente envía un Pragma: no-caché para informar de que desea una copia nueva del recurso especificado.

2.5.4. **Códigos de estado**

Ante cada transacción con un servidor HTTP, éste devuelve un código numérico que informa sobre el resultado de la operación como primera línea del mensaje de respuesta.

Los códigos de estados visualizados en la Tabla 2 están clasificados en cinco categorías:

- 1xx. Mensajes informativos. En HTTP/1.0 no están definidos, y están reservados para un futuro uso.
- 2xx. Mensajes asociados con operaciones realizadas correctamente.

- 3xx. Mensajes de redirección, que informan de operaciones complementarias que se deben realizar para finalizar la operación.
- 4xx. Errores del cliente; el requerimiento contiene algún error, o no puede ser realizado.
- 5xx. Errores del servidor, que no ha podido llevar a cabo una solicitud.

Código	Descripción
200 OK	Operación realizada satisfactoriamente.
201 Created	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. Este nuevo objeto ya está disponible. Puede ser utilizado en sistemas de edición de documentos.
202 Accepted	La operación ha sido realizada correctamente, y como resultado se ha creado un nuevo objeto, cuya URL de acceso se proporciona en el cuerpo de la respuesta. El nuevo objeto no está disponible por el momento. En el cuerpo de la respuesta se debe informar sobre la disponibilidad de la información.
204 No Content	La operación ha sido aceptada, pero no ha producido ningún resultado de interés. El cliente no deberá modificar el documento que está mostrando en este momento.
301 Moved Permanently	El objeto al que se accede ha sido movido a otro lugar de forma permanente. El servidor proporciona, además, la nueva URL en la variable Location de la respuesta. Algunos browsers acceden automáticamente a la nueva URL. En caso de tener capacidad, el cliente puede actualizar la URL incorrecta, por ejemplo, en la agenda de bookmarks.
302 Moved Temporarily	El objeto al que se accede ha sido movido a otro lugar de forma temporal
304 Not Modified	Cuando se hace un GET condicional, y el documento no ha sido modificado, se devuelve este código de estado.
400 Bad Request	La petición tiene un error de sintaxis y no es entendida por el servidor.
401 Unauthorized	La petición requiere una autorización especial, que normalmente consiste en un nombre y clave que el servidor verificará. El campo WWW-Autenticate informa de los protocolos de autentificación aceptados para este recurso.
403 Forbidden	Está prohibido el acceso a este recurso. No es posible

	utilizar una clave para modificar la protección.
404 Not Found	La URL solicitada no existe.
500 Internal Server Error	El servidor ha tenido un error interno, y no puede continuar con el procesamiento.
501 Not Implemented	El servidor no tiene capacidad, por su diseño interno, para llevar a cabo el requerimiento del cliente.
502 Bad Gateway	El servidor, que está actuando como proxy o pasarela, ha encontrado un error al acceder al recurso que había solicitado el cliente.
503 Service Unavailable	El servidor está actualmente deshabilitado, y no es capaz de atender el requerimiento.

Tabla 2. Lista de códigos HTTP

En este proyecto, nuestro cliente Android realizará peticiones HTTP al servidor de vídeo para obtener la imagen JPEG (estática) que está transmitiendo en el momento que recibe la petición. Es decir, el protocolo de comunicación entre cliente y servidor será HTTP.

2.6. Sistemas móviles de Tercera Generación. 3G

2.6.1. Generación anterior

La segunda generación de sistemas móviles celulares supuso un gran cambio con respecto a la primera generación. Se llevó a cabo el paso de radiotransmisión analógica a radiotransmisión digital y se aumentó la capacidad de la red gracias a la división simultánea del canal entre varios usuarios.

Hay 4 estándares principales para los sistemas 2G: *Global System for Mobile Communications* (GSM); *Digital AMPS* (D-AMPS); *Code Division Multiple Access* (CDMA) IS-95; y *Personal Digital Cellular* (PDC). De estos cuatro estándares, GSM ha sido el más extendido y utilizado.

GSM utiliza, principalmente, la banda de 900 MHz aunque hay otras alternativas. Las más importantes son *Digital Cellular System* 1800 (DCS-1800; también conocida como GSM-1800) y *Personal Communications Service* 1900 (PCS-1900; también conocida como GSM-1900). La última sólo se utiliza en Norte América y en Chile. El objetivo de la utilización de una nueva banda de frecuencia es cubrir la falta de capacidad en la banda de 900 MHz. La banda de 1800 MHz puede albergar una mayor cantidad de usuarios, por lo que es especialmente popular en las zonas de mayor densidad de población. La cobertura de esta banda, sin embargo, a veces es menor que la ofrecida por las redes que utilizan la banda de 900 MHz. Por esta razón se utilizan móviles de banda dual, que usan una banda u otra dependiendo de la cobertura disponible.

Entre la segunda y la tercera generación encontramos la 2.5G, que incluye mejoras en las redes 2G hasta hacer que ofrezcan casi las mismas capacidades que las planificadas para las redes 3G. En general, un sistema GSM 2.5G incluye al menos una de las siguientes tecnologías: *High-Speed Circuit-Switched Data* (HSCSD); *General Packet Radio Services* (GPRS); y *Enhanced Data Rates for Global Evolution* (EDGE).

El principal problema de GSM es su pobre tasa de transferencia, de 9.6 Kbps. Una tasa tan baja que hace que, por ejemplo, navegar por la Web sea una tarea desesperante. HSCSD es el modo mas fácil de acelerar las

cosas. Esto significa que, en vez de un intervalo de tiempo, una estación móvil utilice varios intervalos para una conexión de datos. En las implementaciones comerciales desarrolladas, normalmente el máximo de intervalos es cuatro. La tasa total es simplemente el número de intervalos de tiempo multiplicado por la tasa de un intervalo (9.6 Kbps). Ésta es una forma relativamente barata de mejorar las capacidades de la red, puesto que sólo necesita mejoras de software en la red (además de nuevos teléfonos que soporten HSCSD). Pero HSCSD tiene inconvenientes. El mayor problema es el uso de los escasos recursos radio. HSCSD funciona por conmutación de circuitos, por lo que asigna los intervalos de tiempo utilizados (sin que puedan ser utilizados por otro) aunque no se esté transmitiendo nada en un determinado momento. Por otro lado, esta misma característica hace de HSCSD una buena opción para las aplicaciones en tiempo real, donde los retardos deben ser bajos. El problema es que la mayoría de los usuarios utilizan estos servicios en zonas donde las redes móviles ya están congestionadas, por lo que HSCSD no serviría para mejorar la situación. Otro problema que tuvo HSCSD es que los fabricantes no estaban muy interesados en implementarlo.

La siguiente solución es GPRS. Con esta tecnología, la tasa puede llegar hasta los 115 Kbps o más si nos olvidamos de la corrección de errores. Sin embargo, con la protección adecuada de los datos, la cuota de 115 Kbps es el máximo teórico en condiciones óptimas con 8 intervalos de tiempo (en sentido descendente). Una buena aproximación para el rendimiento en condiciones normales es 10 Kbps por intervalo de tiempo. Pero lo que es más importante que el rendimiento es que GPRS utiliza conmutación de paquetes, de tal forma que los recursos radio sólo permanecen asignados cuando hay algo que enviar. GPRS es especialmente recomendable para las aplicaciones que no son en tiempo real (no necesitan un retardo bajo), como el correo electrónico o la navegación Web. Sin embargo, no es buena elección para las aplicaciones en tiempo real porque utiliza técnicas de contienda y, por ello, no puede garantizar un retardo máximo.

La implementación de un sistema GPRS es mucho más cara que la de un sistema HSCSD. Se necesitan nuevos componentes en la red, además de modificaciones en los existentes. Sin embargo era un paso necesario hacia unas mejores capacidades en GSM con respecto al tráfico de datos, que estaba cogiendo fuerza frente al tráfico de voz. Además, para los operadores que quisieran operar con 3G en el futuro, GPRS era paso muy significativo puesto que el núcleo de las redes 3GPP (3rd Generation Partnership Project) se basan en la combinación de los núcleos de las redes GSM y GPRS.

La tercera opción para mejorar GSM es EDGE (Enhanced Data Rates for GSM Evolution). La idea es utilizar el por entonces nuevo esquema de modulación *Eight-Phase Shift Keying* (8PSK), con el que se obtenía el triple de tasa con respecto a GSM. Para su implementación, sólo se necesitan mejoras de software en las estaciones base. No reemplazaba la antigua modulación *Gaussian Minimun Shift Keying* (GMSK) sino que coexistía con ella. De esta forma, los usuarios que no necesitasen la mejora de servicio proporcionada por EDGE no tendrían que cambiar su terminal. El gran problema que presentaba EDGE es que la modulación 8PSK sólo era eficaz a corta distancia, por lo que la modulación GMSK seguía siendo necesaria para cubrir grandes zonas de cobertura.

Enhanced GPRS (EGPRS) es la combinación de EDGE y GPRS. La tasa máxima que ofrece esta fusión, utilizando 8 intervalos de tiempo (con la protección de datos adecuada) es de 384 Kbps. Pero esta tasa sólo se consigue cuando se utilizan todos los recursos radio de una portadora y sólo si la estación móvil está cerca de la estación base. La combinación entre EDGE y HSCSD (ECSD) proporciona tasas que triplican las de la tecnología HSCSD estándar.

De todas ellas, la tecnología más utilizada fue GPRS, que más tarde evolucionó hacia UMTS (la tecnología 3G utilizada en Europa). En la siguiente figura podemos ver la arquitectura simplificada de la red GSM con GPRS.

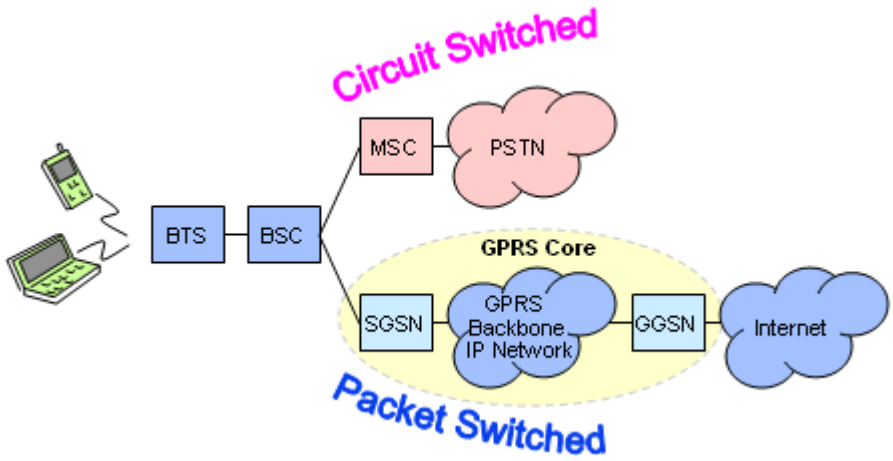

Figura 5. Arquitectura simplificada de la red GSM con GPRS.

Los elementos que aparecen son:

- Base Transceiver Station (BTS). Son las estaciones base, que se comunican por radio con los móviles de los usuarios y con otras estaciones base.
- Base Station Controller (BSC). Controlan los radiocanales de varias BTS y gestionan los traspasos entre estaciones base (roaming).
- Mobile service Switching Center (MSC). Frontera entre los BSC y la red telefónica conmutada (PSTN). Encamina llamadas y gestiona la movilidad y los traspasos entre BSC para terminales en su área de servicio. Conmuta circuitos a 64 Kbps.
- Serving GPRS Support Node (SGSN). Frontera entre los BSC y el núcleo de la red GPRS. Es el responsable de la entrega de los paquetes provenientes de la red GPRS a los terminales destino en su área de servicio, además de gestionar la movilidad y las sesiones dentro de la misma.
- Gateway GPRS Support Node (GGSN). Frontera entre la red GPRS e Internet. Es el encargado de encapsular los paquetes hacia los SGSN destino, así como de acceder a redes externas (como Internet).

2.6.2. **UMTS**

El mismo año que GSM se lanzó al mercado, la ETSI comenzó a trabajar en la estandarización de la siguiente generación de redes de telecomunicaciones móviles. El nuevo sistema se llamó *Universal Mobile Telecommunications System* (UMTS). Más tarde, las compañías de telecomunicaciones más importantes se unieron creando el programa 3GPP (3erd Generation Partnership Project). El objetivo de este proyecto era producir las especificaciones para un sistema 3G basado en la interfaz radio UTRA (UMTS Terrestrial Radio Access) y el núcleo de red de GSM/GPRS. La *Universal Terrestrial Radio Access* (UTRA) utiliza la tecnología W-CDMA, con un ancho de banda de 5 MHz y los métodos DS-CDMA (Direct-Sequence Code Division Multiple Access) y FDD (Frequency-division duplexing) para acceso al canal y duplexación respectivamente.

Las principales mejoras introducidas con UMTS son: el aumento significativo de la tasa de transmisión de datos (hasta 2 Mbps de bajada en condiciones óptimas); y el roaming internacional, proporcionando un método de traspaso de un terminal entre dos redes UMTS distintas (si los operadores de ambas redes tienen un convenio en vigor) evitando que el usuario quede incomunicado por falta de cobertura de su operador. Además, los terminales 3G disponen de la capacidad de cambiar a modo 2G cuando no sea posible conectar a una red 3G, proporcionando un área de cobertura mayor.

En la siguiente figura podemos ver un esquema de la arquitectura de UMTS.

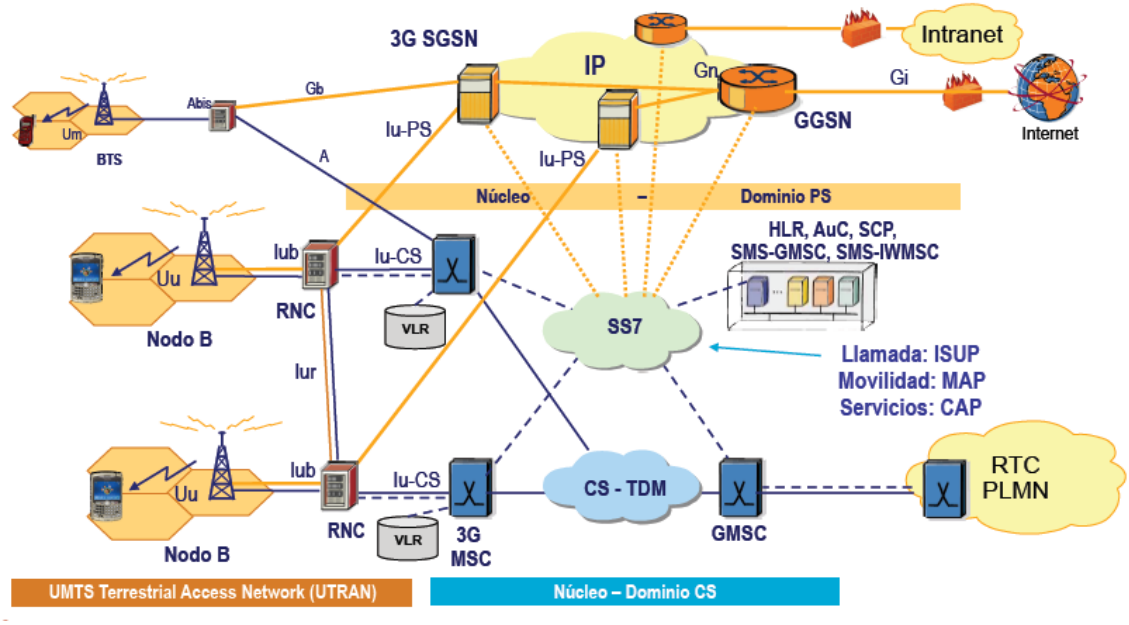

Figura 6. Arquitectura UMTS.

En la figura observamos el cambio en el nombramiento de los elementos que forman parte del acceso a la red.

- Nodo B. Representa las BTS de la anterior generación. Sus funciones son de nivel de capa física (modulación, control de recursos radio, codificación de canal, …).
- Radio Network Controller (RNC). Sustituye a la BSC de GSM. Controla un grupo de Nodos-B y se encarga de tareas como la asignación de códigos, control de admisión, gestión de traspaso suave o control de congestión.

Los demás elementos son iguales a los de la red GSM/GPRS.

El *Visitor Location Register* (VLR) contiene información acerca de las estaciones móviles (como el IMSI, que identifica al abonado o el MSISDN, que contiene el número telefónico) que se encuentran en el área del MSC (Mobile services Switching Center) al que pertenece el registro. La información del registro varía conforme las estaciones móviles abandonan y se agregan al MSC.

El GMSC es un MSC que hace de pasarela entre la red telefónica conmutada y los demás MSCs de la red. Se encarga de encaminar las llamadas entrantes hacia los MSCs apropiados consultando el registro HLR (Home Location Register). El registro HLR es similar al VLR, solo que su información no cambia. Dispone de la misma información además de otros campos como el MSC donde se encuentra la estación móvil o el VLR donde se encuentra registrada actualmente.

2.6.3. HSDPA

La tecnología *High Speed Downlink Packet Access* (HSDPA) se diseñó para mejorar la capacidad de transmisión en el enlace de descarga de los sistemas 3G para dar soporte a las aplicaciones multimedia, que necesitan un ancho de banda mayor que el ofrecido por UMTS/WCDMA.

En el esquema de HSDPA, la idea es añadir un nuevo canal de banda ancha compartido para la descarga,

optimizado para la transferencia de datos a gran velocidad.

HSDPA es una combinación de técnicas donde todas contribuyen a la mejora de la capacidad del canal de descarga. Un dispositivo con soporte HSDPA versión 5 dispone de las funciones *Hybrid Automatic Repeat request* (HARQ) y *Adaptive Modulation and Coding* (AMC). En versiones posteriores se añaden las técnicas de *Fast Cell Selection* (FCS) y *Multiple Input Multiple Output* (MIMO).

HARQ se encarga de la detección y corrección de errores. En el nuevo esquema de capas de HSDPA, los buffers de retransmisión de HARQ se encuentran en la nueva entidad lógica MAC-hs, justo encima de la capa física. En la versión 99 (anterior), la función de retransmisión era trabajo de la capa RLC (Radio Link Control). Podemos ver la diferencia en la siguiente figura.

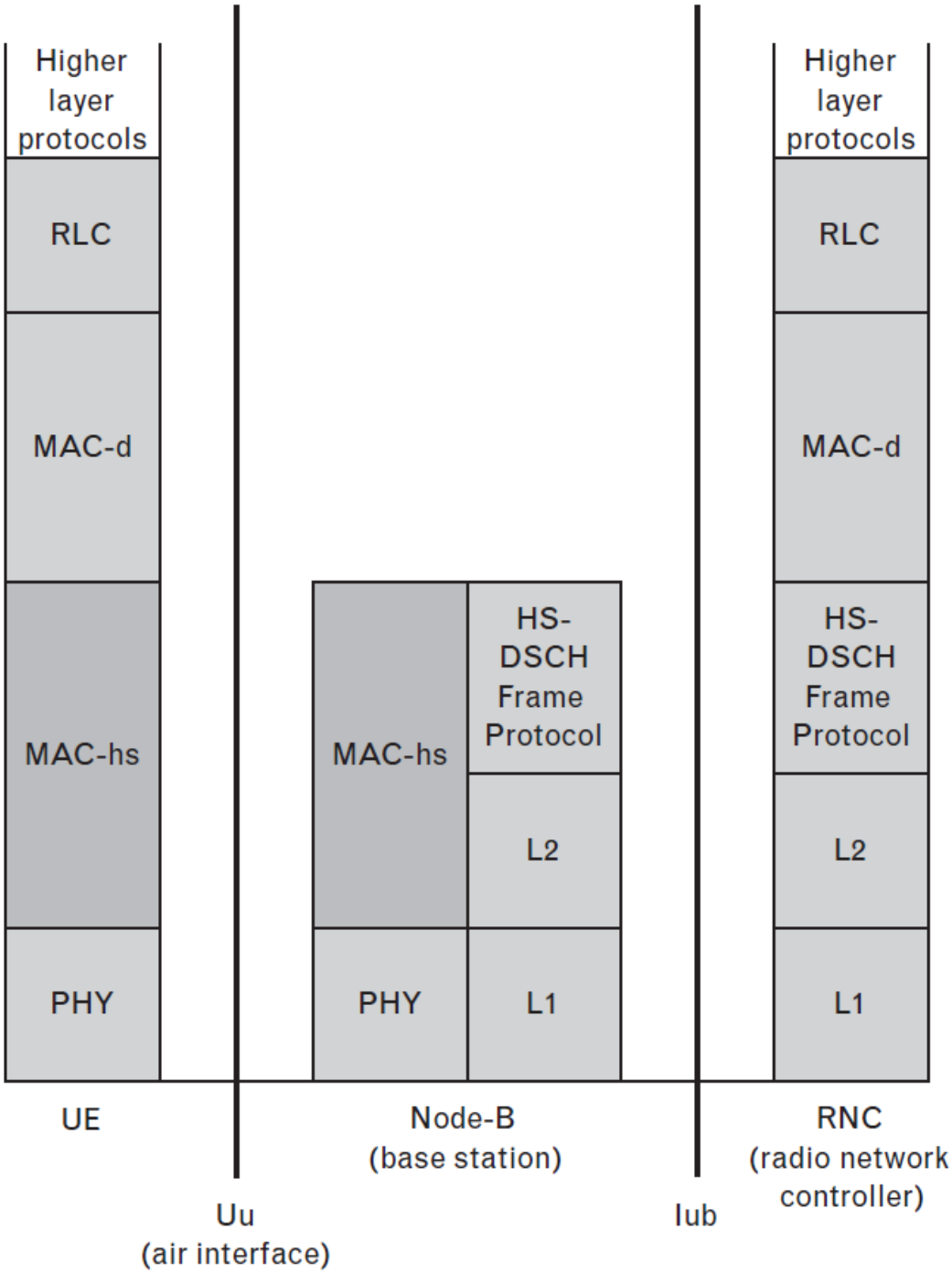

Figura 7. Esquema de capas de UTRAN en HSDPA.

Las combinaciones de HARQ están basadas en la redundancia incremental. Es decir, si la transmisión falla, los datos recibidos (corruptos) se almacenan en un buffer. Las transmisiones exitosas introducirán más redundancia y se combinan con los datos que ya estaban en el buffer. Esta operación se repite hasta que se considera que los datos del buffer son correctos o hasta que se alcanza el máximo de retransmisiones. Además, para que HARQ sea más eficiente, se utiliza un TTL (Time To Live) más pequeño. Este nuevo TTL ocupa tres intervalos de tiempo (2ms), a diferencia de los 15 intervalos (10 ms) utilizados en los otros canales físicos. Gracias a esto, el tiempo de detección de fallos en la transmisión se reduce notablemente.

AMC adapta el formato de transporte (modulación utilizada, por ejemplo) sobre el canal compartido dependiendo de las condiciones de calidad del canal. Las condiciones de calidad del canal se controlan constantemente, pudiendo cambiar el formato de transporte en cada trama. De esta forma, en áreas donde se den buenas condiciones, la UTRAN podrá utilizar una modulación de mayor nivel y menos redundancia, mientras que en zonas de peores condiciones se utilizará una modulación más robusta y los datos transmitidos llevarán más información de redundancia. Las modulaciones empleadas en la versión 5 son QPSK y 16QAM. En versiones posteriores se introduce el uso de otras modulaciones, como la 64QAM, para mejorar aún más las prestaciones.

HSDPA introduce tres nuevos tipos de canales: HS-DSCH es el canal compartido para la descarga de alta velocidad; HS-SCCH para transportar información de control; y HS-DPCCH para el envío de los ACK cuando los datos se han recibido y han sido procesados por la capa MAC-hs.

En condiciones óptimas, la tasa máxima de descarga con la tecnología HSDPA versión 5 es de 14.4 Mbps. Un valor considerablemente mayor que el obtenido con UMTS/WCDMA y mucho mayor que los valores obtenidos con tecnología 2G.

En este proyecto utilizaremos la red HSDPA para conectar con el servidor de vídeo. Compararemos los resultados obtenidos en la descarga de secuencias de imágenes con el proyecto realizado en J2ME utilizando la red GPRS.

2.7. Servidor para captura de imágenes y video con Java.

El servidor de imágenes utilizado en este proyecto pertenece al Proyecto Fin de Carrera: "Servidor para Captura de Imágenes y Video con Tecnología Java (JMF y JIMI)" realizado por Diego Ruiz Macías. El texto original del proyecto está disponible en la web de la biblioteca para su libre consulta. Por ello, en esta sección describiremos las funciones que nos interesan para el desarrollo de este proyecto, sin profundizar en aspectos técnicos del servidor ni del código de su implementación.

Java Media Framework (JMF) es una API (Application Programming Interface) de Java que permite trabajar con contenido multimedia en las aplicaciones Java, facilitando el acceso a dispositivos de captura de datos multimedia y el tratamiento de la información obtenida de dichos dispositivos. La idea es utilizar esta API para obtener los datos multimedia que van a transmitirse desde la fuente que nos convenga (Webcam o dispositivo de vídeo externo).

Java Image Management Interface (JIMI) es otra API de Java para la lectura, escritura y manipulación de archivos de imagen en múltiples formatos, como GIF, JPEG, PNG, o BMP entre otros. El papel de esta herramienta es el de manipular las imágenes obtenidas a través del dispositivo de vídeo mediante JMF según las peticiones del cliente. En nuestro proyecto, haremos uso de las capacidades de redimensionado para ofrecer varios tipos de resolución de imagen además de modificar la calidad de imagen para reducir el volumen de descarga.

En el proyecto se hace uso de *Real-Time Transport Protocol* (RTP) para el transporte de flujo de vídeo a través de Internet. El cliente de vídeo fue implementado mediante un applet Java que se carga desde el servidor a través del navegador web del cliente. Para nuestro proyecto, no necesitaremos conocer más acerca de este apartado.

No utilizaremos el protocolo RTP ya que no vamos a trabajar con imágenes de vídeo, sino con imágenes estáticas. Utilizaremos el protocolo HTTP para comunicarnos con el servidor y obtener las imágenes capturadas por el dispositivo de vídeo configurado por el servidor.

La siguiente imagen representa el esquema del escenario en este proyecto.

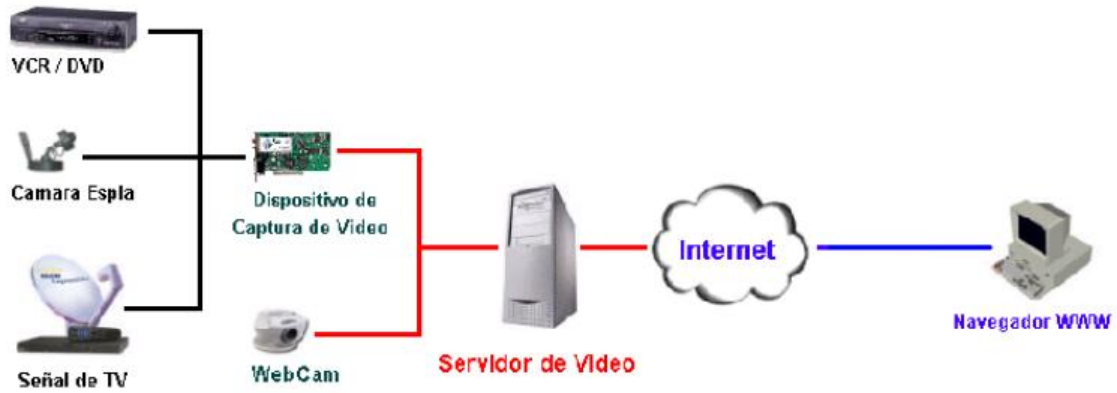

Figura 8. Escenario para Servidor JMF y JIMI

En este proyecto, se implementará un cliente que se ubicará en un terminal móvil (Android). El cliente se conectará utilizando la red HSDPA al servidor para obtener, en secuencia, las imágenes que se están transmitiendo en cada momento.

3. TRABAJO REALIZADO

En la sección anterior se trató la tecnología utilizada para la realización de este proyecto. Se analizó su función dentro del mismo y por qué fueron elegidas. A continuación se muestra un breve resumen de las tecnologías utilizadas:

- Android. Sistema Operativo para terminales móviles más utilizado en la actualidad. Tiene una muy buena documentación, lo que facilita el desarrollo de aplicaciones.
- Java. Es el lenguaje de programación más utilizado para desarrollar aplicaciones Android.
- Eclipse. Entorno donde se ha realizado la implementación de la aplicación Android para este proyecto.
- OpenCV. Librería de visión artificial de código libre. Utilizada en el proyecto para añadir la funcionalidad de detección de patrones en imágenes.
- HTTP (Hypertext Transfer Protocol). Protocolo de comunicación con el servidor de vídeo.
- HSDPA (High-Speed Downlink Packet Access). Tecnología perteneciente a la tercera generación de redes de datos móviles. Será el canal de comunicación entre cliente y servidor.

La siguiente sección muestra el trabajo realizado en este proyecto. Está dividida en dos partes. La primera parte muestra la estructura y el funcionamiento del cliente Android implementado para la recepción de las imágenes desde el servidor, haciendo uso de la red HSDPA. La segunda parte se trata de una propuesta de aplicación práctica para el proyecto. Se describe la idea en detalle, presentando las posibles ventajas que puede proporcionar y las desventajas que puede presentar la aplicación práctica estudiada. Además, se muestra qué es necesario para llevar a cabo la idea y cómo se ha instalado en un escenario real.

3.1. Cliente Android

El cliente que se ha desarrollado para este proyecto se encuentra instalado en un Smartphone con Sistema Operativo Android. En concreto, se trata de una aplicación que interactúa con el usuario antes de comunicarse con el servidor, para obtener los parámetros de la conexión (IP y puerto de escucha) y de las imágenes que serán descargadas (Resolución y calidad de imagen).

La aplicación Android que actúa de cliente, realiza peticiones al servidor desde el cual se sirven imágenes captadas mediante la tecnología JMF (Java Media Framework) desde una WebCam o desde algún dispositivo de captura de vídeo conectado al servidor. A cada imagen, una vez descargada, se le aplica un proceso de detección facial utilizando métodos de la librería OpenCV. Tras esto, la imagen se muestra en la pantalla del Smartphone, sustituyendo la imagen descargada anteriormente justo antes de comenzar la descarga de la siguiente imagen. La Figura 9 muestra el esquema de estados del cliente.

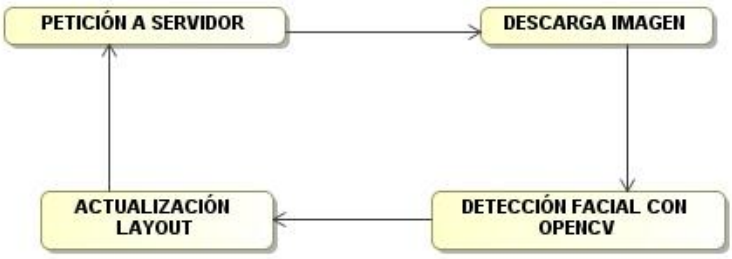

Figura 9. Esquema básico de estados del Cliente Android

El intervalo de tiempo transcurrido desde que se muestra una imagen recibida hasta que se reemplaza por la siguiente que ha sido descargada dependerá de dos factores:

- La velocidad de descarga de datos del Smartphone donde se encuentre instalado el cliente. Esta velocidad dependerá de la cobertura 3G que esté disponible en el área donde esté situado el mismo. Si disponemos de cobertura HSDPA obtendremos una mayor tasa que si sólo disponemos de cobertura UMTS/WCDMA.
- El tamaño de las imágenes que son descargadas desde el servidor, que depende de la resolución y la calidad de imagen seleccionadas en la descarga. Estos parámetros son configurables gracias a que el servidor utiliza la tecnología JIMI (Java Image Management Interface) para la manipulación de las imágenes. De esta forma, si elegimos una resolución de 320x240 obtendremos un mejor rendimiento que si elegimos una resolución de 640x480. Si la resolución de la pantalla del Smartphone donde se encuentra instalado el cliente es pequeña, será preferible elegir una resolución de imagen de 320x240 ya que se ajustará bastante a la resolución de la pantalla y además tendremos una secuencia más fluida de imágenes. Si por lo contrario el Smartphone dispone de una pantalla de muy alta resolución (como 1080x1920), será preferible elegir una resolución mayor para obtener unas imágenes más nítidas aunque la secuencia sea algo más lenta. En cuanto a la calidad de imagen, si elegimos una calidad del 100% tendremos una imagen mucho más pesada que si elegimos una calidad del 70%. De nuevo habrá que decidir si es preferible una mayor fluidez en la secuencia de imágenes o una mejor imagen.

3.1.1. Estructura de la aplicación

La aplicación está formada por 4 clases:

- MainActivity.java
- Conexion.java
- ConexionActivity.java
- TestActivity.java.

En la Figura 10 puede observarse la estructura interna de la aplicación. Se puede observar como las clases MainActivity y Conexión heredan de la clase Activity, de la API de Android. TestActivity y ConexionActivity heredan de la clase Conexion, y ambas tienen una clase interna llamada procesadoPeticion. La clase Conexion hace uso de los módulos android, objdetect y core de la librería OpenCV.

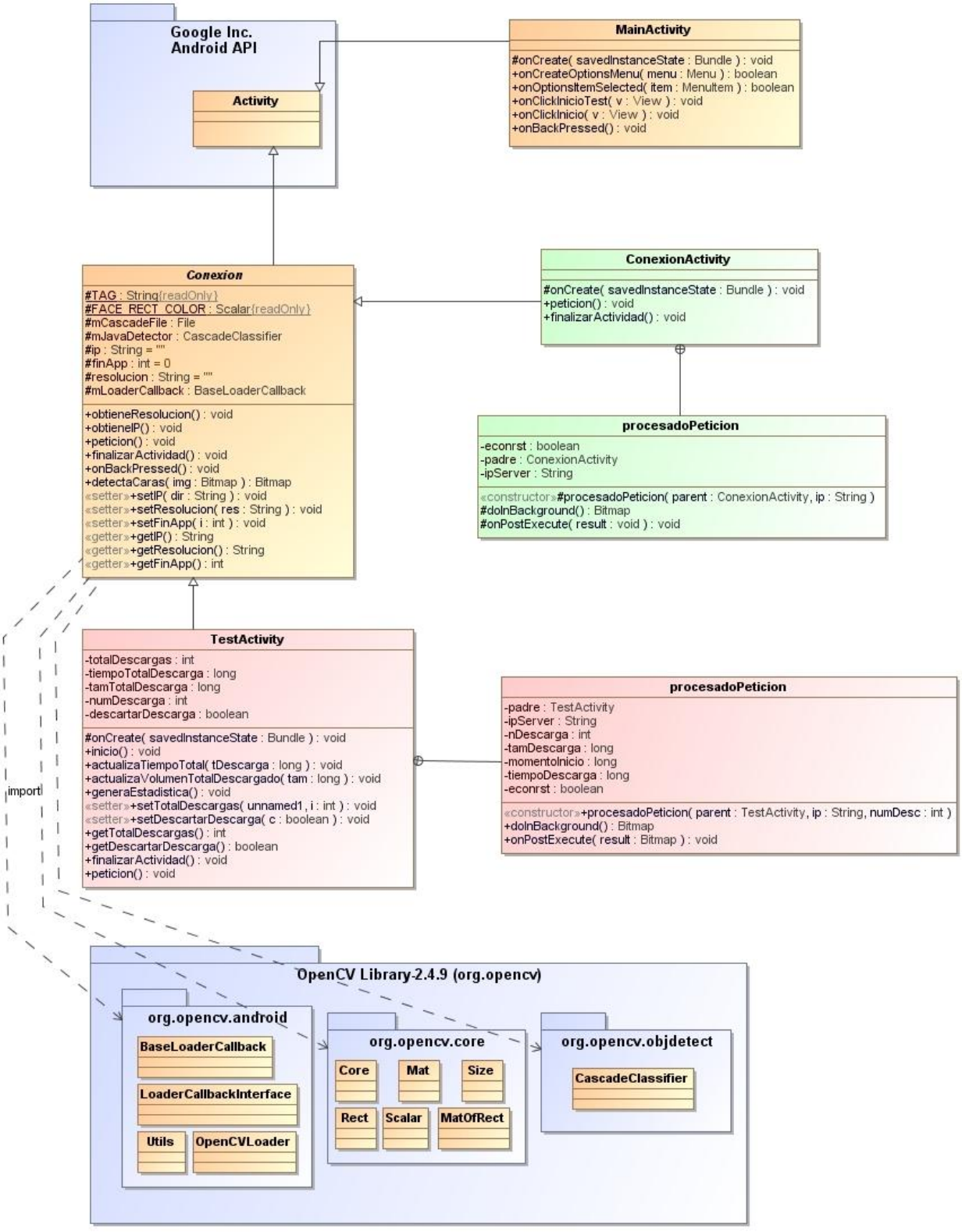

Figura 10. Estructura interna de la aplicación.

3.1.1.1. MainActivity.java

La clase MainActivity hereda de la clase Activity, de la API de Android. Las actividades son clases que interactúan con el usuario. Cada actividad está asociada a un layout, donde se define la presentación de la misma de cara al usuario. La clase MainActivity es la primera actividad que se lanza cuando se abre la aplicación del cliente. Es una clase sencilla, provista de un conjunto de botones que permitirán al usuario

lanzar las demás actividades para poder acceder a las imágenes emitidas por el servidor.

Sobreescribe los siguientes métodos de la clase Activity:

- *onCreate* para fijar el layout utilizado
- *onCreateOptionsMenu* y *onOptionsItemSelected* para la creación y gestión de un pequeño menú de opciones donde se podrá obtener información de la aplicación
- *onBackPressed* para forzar la finalización de la actividad cuando se pulse "atrás" en el Smartphone.

Además dispone de los métodos *onClickInicioTest* y *onClickInicio* para lanzar las actividades TestActivity y ConexionActivity respectivamente, cuando se pulse sobre uno de los botones del layout.

La siguiente figura muestra la estructura interna de esta clase.

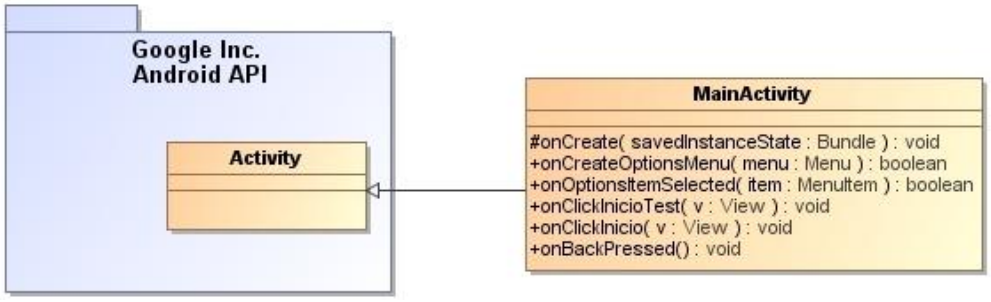

Figura 11. Estructura interna de la clase MainActivity

3.1.1.2. Conexion.java

La clase Conexion es una clase abstracta, que hereda de la clase Activity. Su objetivo es agrupar los métodos y variables comunes entre las clases TestActivity y ConexionActivity, propiciando la reutilización de código. Los métodos que implementa esta clase son:

- *obtieneResolucion* para guardar la resolución de imagen elegida por el usuario.
- *obtieneIP* para almacenar la IP del servidor de video.
- *detectaCaras* para detectar caras frontales en una imagen pasada como parámetro y dibujar un recuadro alrededor de las caras detectadas, devolviendo la imagen modificada. Para este proceso, el método hace uso de la librería de OpenCV.

La clase Conexion sobreescribe el método *onBackPressed* para garantizar la finalización de la actividad cuando el usuario pulsa el botón "atrás" del Smartphone. Además, en esta clase se definen dos métodos pero no se implementan. Estos métodos son *peticion* y *finalizarActividad* que serán implementados por las clases herederas. Es necesaria su presencia aunque no estén implementados en esta clase porque son llamados desde otros métodos de la propia clase, presentando un comportamiento distinto dependiendo de la clase heredera. En cuanto a las variables, merece especial mención la variable *mLoaderCallback* de tipo BaseLoaderCallback (definido en la librería de OpenCV). Esta variable es la responsable de que la librería de OpenCV esté preparada para su posterior uso, y de cargar los ficheros necesarios para inicializar el detector de patrones.

La siguiente figura muestra la estructura interna de esta clase.

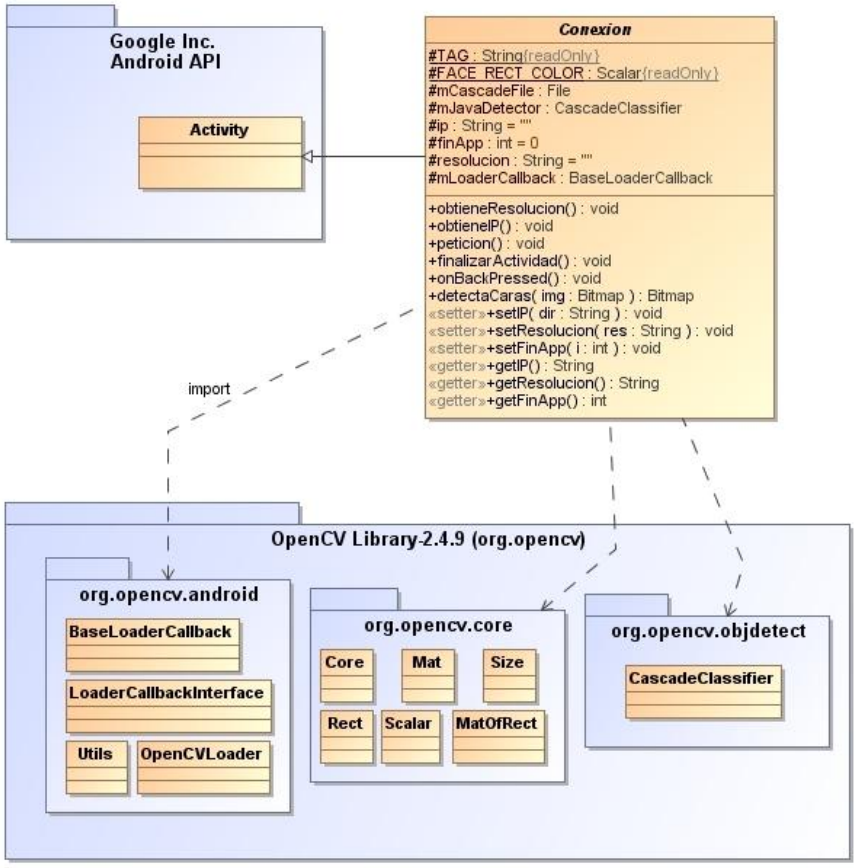

Figura 12. Estructura interna de la clase Conexión

3.1.1.3. ConexionActivity.java

La clase ConexionActivity hereda de la clase Conexión. Su función es conectarse con el servidor de vídeo cuya dirección IP ha sido proporcionada por el usuario. En caso de fallo en la conexión, se volverá a pedir la IP del servidor para intentar conectar de nuevo. Una vez que la conexión se ha realizado con éxito, comenzará un bucle de peticiones al servidor para obtener una secuencia de imágenes que se irán mostrando en el layout de la actividad. El bucle terminará cuando el usuario pulse el botón "atrás", volviendo a la actividad MainActivity. Si durante la transmisión ocurre algún fallo de conexión, se intentará reconectar con la misma IP que se encuentra almacenada. Si falla el intento de reconexión, se alertará del error y se volverá a pedir la IP del servidor. El layout de esta actividad está compuesto por un único elemento, un ImageView donde se irán mostrando las imágenes cuando sean descargadas y se les haya aplicado la detección facial.

Esta clase sobreescribe los siguientes métodos:

- *onCreate* para la selección del layout y cargar la librería de OpenCV.
- *peticion* para la ejecución de las tareas en segundo plano provistas por la clase interna *procesadoPeticion*.
- *finalizarActividad* para forzar la finalización de la actividad cuando se pulsa el botón "atrás", evitando que queden procesos ejecutándose en segundo plano.

ConexionActivity tiene una clase interna, *procesadoPeticion*, que hereda de la clase *AsyncTask* de la API de Android. Esta clase se utiliza para realizar tareas en segundo plano, evitando que el hilo principal quede bloqueado. En concreto, en esta aplicación se utiliza para la descarga de las imágenes desde el servidor y la detección facial en dichas imágenes, mostrando el resultado en el layout.

Las instancias de clases de tipo AsyncTask deben sobreescribir, al menos, los métodos *doInBackground* y *onPostExecute* para definir las tareas a realizar en segundo plano cuando se ejecuta el método *execute* de la propia clase, y las tareas a realizar antes de volver al hilo principal. A continuación se describe la utilidad de estos métodos en nuestro caso:

- El método *doInBackground* servirá para conectar con el servidor y descargar la imagen, comprobando si hay algún fallo en la conexión. La imagen descargada es pasada como parámetro al método *onPostExecute.*
- En el método *onPostExecute* se realizará el proceso de detección facial sobre la imagen descargada. Después se actualizará la imagen mostrada en el layout antes de volver al hilo principal para ejecutar otra clase *procesadoPeticion*. Si falla la conexión, se le notificará al usuario y volverá al hilo principal para pedir de nuevo la IP del servidor o para intentar reconectar con la misma IP, si es posible.

La siguiente figura muestra la estructura interna de esta clase.

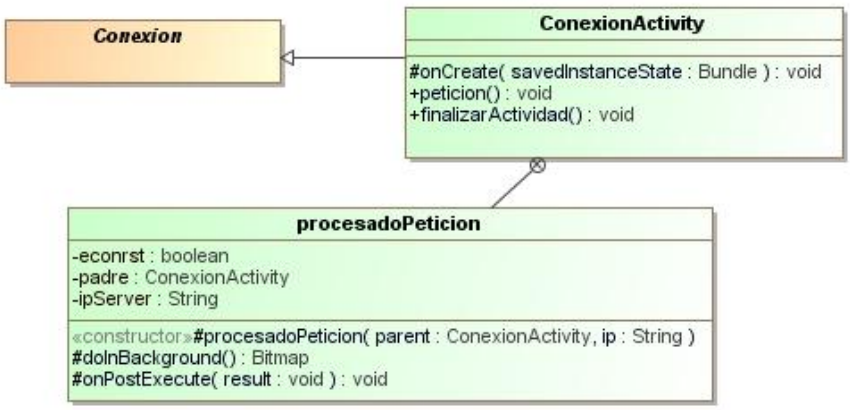

Figura 13. Estructura interna de la clase ConexionActivity

3.1.1.4. TestActivity.java

La clase TestActivity, al igual que la clase anterior, hereda de la clase Conexion. Esta clase sirve para realizar un test de la aplicación, sobre una secuencia de N imágenes descargadas del servidor. Tras realizar la descarga de las imágenes, se muestra un cuadro de diálogo donde aparecen los resultados del test (tiempo medio entre imágenes consecutivas y tamaño medio de cada descarga). La estructura es similar a la estructura de la clase ConexionActivity, añadiendo las funcionalidades que proporcionarán las estadísticas del test. En cuanto al layout de esta actividad, además de tener un ImageView donde irán apareciendo las imágenes descargadas, dispone de un TextView, situado encima del ImageView, donde podremos ver el avance de la secuencia y el tamaño de la imagen actual.

La clase TestActivity sobreescribe los siguientes métodos:

- *onCreate* para la selección del layout, cargar la librería de OpenCV y llamar al método *inicio.*
- *peticion* para la ejecución de las tareas en segundo plano provistas por la clase interna *procesadoPeticion.*
- *finalizarActividad* para forzar la finalización de la actividad cuando se pulsa el botón "atrás", evitando que queden procesos ejecutándose en segundo plano.

Además, dispone de los siguientes métodos propios:

- *inicio* para dar comienzo al test, pidiendo en primer lugar el número de imágenes que deberán descargarse en la secuencia y procediendo después a tomar los valores para la resolución y la IP del servidor

- *actualizaTiempoTotal* para actualizar el tiempo total entre imágenes consecutivas, cada vez que se descarga una nueva imagen, para su posterior tratamiento.
- *actualizaVolumenTotalDescargado* para actualizar el volumen de datos descargados, cada vez que se descarga una nueva imagen, para su posterior tratamiento.
- *generaEstadística* para calcular el tiempo medio entre imágenes consecutivas y el tamaño medio de cada imagen, mostrando los resultados al usuario en un cuadro de diálogo.

Al igual que la anterior clase, TestActivity dispone de una clase interna llamada *procesadoPeticion*. Esta clase tiene las mismas funciones que en el caso de ConexionActivity, además de otras funciones como el cálculo del tiempo de descarga en segundo plano y la obtención del tamaño de los archivos descargados. También se encarga de controlar cuándo se llega al número total de imágenes a descargar, procediendo a generar las estadísticas pertinentes sobre el test realizado.

La siguiente figura muestra la estructura interna de esta clase.

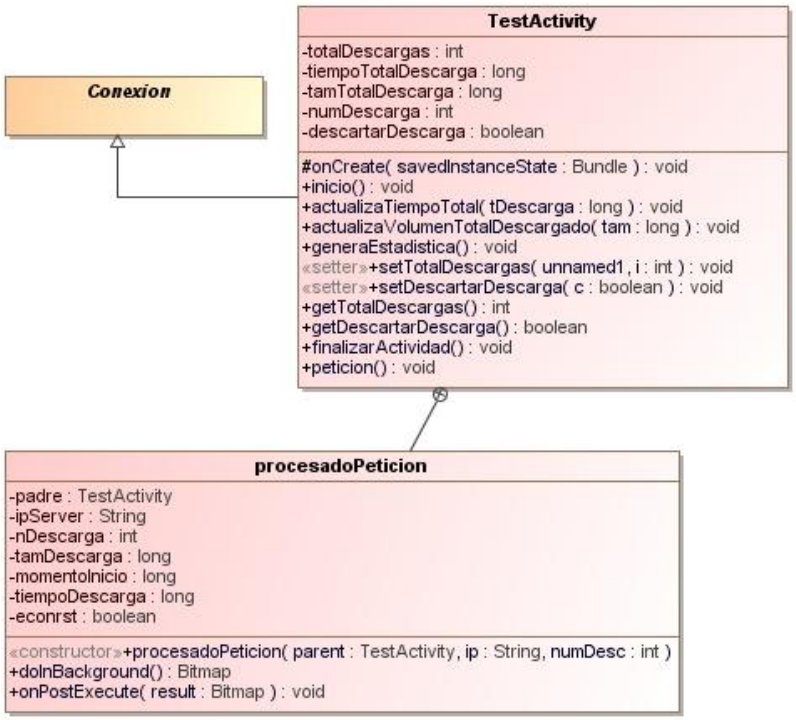

Figura 14. Estructura interna de la clase TestActivity

3.1.2. **Flujo de la aplicación**

En esta subsección se presentará un diagrama de flujo de la aplicación, mostrando su ciclo de vida y el paso entre actividades. Se empezará con un diagrama de flujo global de la aplicación, y después se mostrará el diagrama de flujo interno de cada actividad implicada en el ciclo de vida de la aplicación.

3.1.2.1. Diagrama de flujo global

Como puede observarse en la Figura 15, la actividad que se lanza justo al iniciar la aplicación es MainActivity. Esta actividad será la encargada de lanzar las demás actividades y de finalizar la aplicación según las interacciones del usuario.

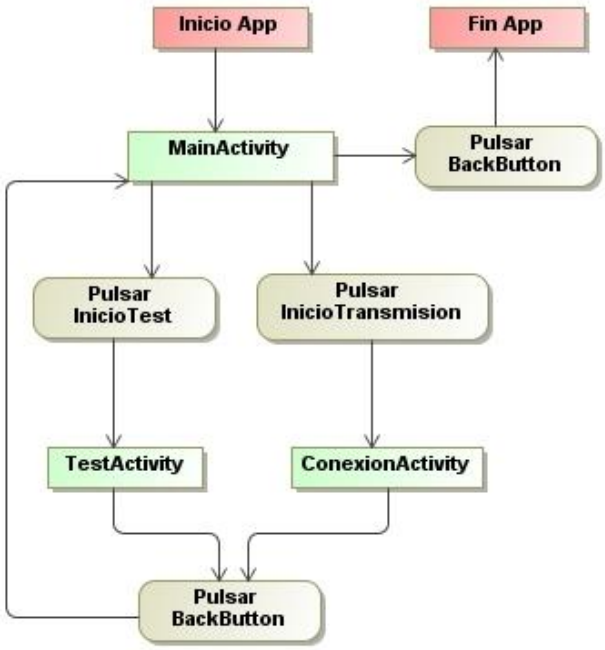

Figura 15. Diagrama de flujo global.

El layout de MainActivity está compuesto por dos botones. Al pulsar uno de ellos, MainActivity se encargará de crear y lanzar la actividad correspondiente al botón pulsado (TestActivity o ConexionActivity). Si el usuario pulsa el botón "atrás", MainActivity procederá a destruir la aplicación.

Tanto TestActivity como ConexionActivity podrán destruirse mientras estén ejecutándose, pulsando el botón "atrás" del Smartphone, eliminándose de la pila de actividades. La aplicación volverá entonces a la siguiente actividad de la pila, que siempre será MainActivity. Estas actividades interactuarán con el usuario al inicio, para recoger los parámetros necesarios para su funcionamiento. Una vez establecida la conexión, sólo se volverá a interactuar con el usuario en caso de fallo de conexión o en caso de finalización del test (sólo en TestActivity). A continuación se expondrá con más detalle el flujo interno de cada actividad.

3.1.2.2. Flujo de MainActivity

Cuando se inicia la aplicación, se crea una instancia de la actividad MainActivity. En el momento en que es creada, se establece el layout que será mostrado por pantalla y se crea el menú de opciones, situado en la barra de acciones. Este menú de opciones constará de un solo elemento, cuyo título es "Sobre…", que servirá para desplegar un cuadro de diálogo con el nombre de este trabajo, así como el nombre de su autor y el tutor del trabajo.

Una vez realizado el paso anterior, la actividad quedará a la espera de interacciones por parte del usuario. Básicamente, las acciones que pueden llevarse a cabo en este punto son:

- Pulsar el botón "atrás" del Smartphone, que supondrá la destrucción de la actividad y de la instancia de la aplicación.
- Pulsar sobre el icono del menú de opciones. Se desplegará entonces el único elemento del que dispone dicho menú, permitiendo al usuario acceder a la información de la aplicación ya descrita.
- Pulsar sobre uno de los botones del layout. En este caso, MainActivity creará una instancia de la clase TestActivity o ConexionActivity, dependiendo del botón pulsado. Después de crear la actividad, procederá a lanzarla y MainActivity pasará al segundo puesto en la pila de actividades. Cuando la

actividad lanzada finalice, MainActivity volverá a ocupar el primer puesto en la pila de actividades y volverá al estado inicial, a la espera de una nueva interacción del usuario.

En la Figura 16 se muestra el diagrama de flujo para esta actividad.

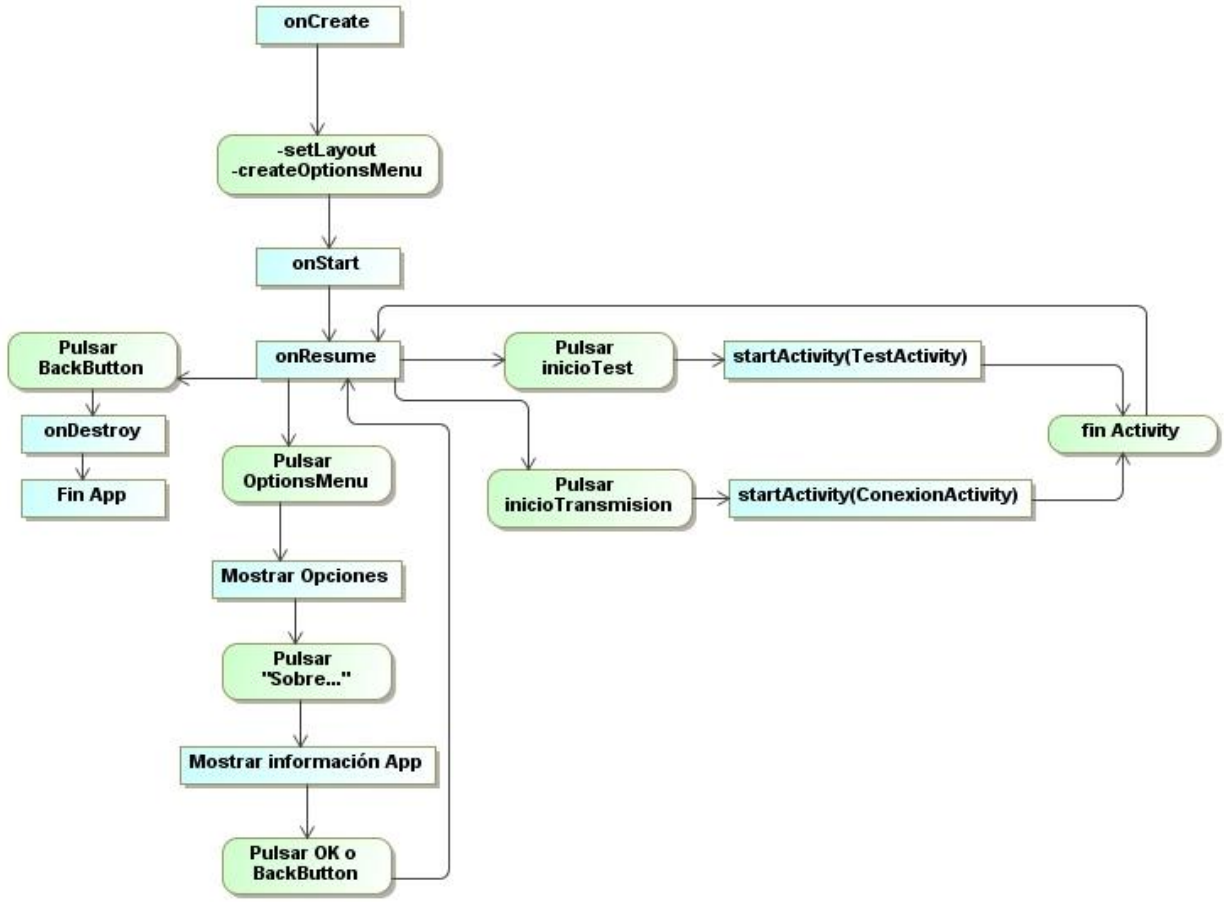

Figura 16. Diagrama de flujo de MainActivity

3.1.2.3. Flujo de ConexionActivity

Cuando se crea la actividad ConexionActivity, lo primero que se hace es configurar el layout y cargar la librería de OpenCV, para poder manejar después el detector con las imágenes que se descarguen. Cuando la actividad se crea correctamente, se procede a obtener los parámetros de la conexión.

En primer lugar, aparecerá un cuadro de diálogo con varios valores para la resolución. El usuario deberá elegir la opción que le parezca más oportuna. En caso de pulsar el botón "Cancelar" del diálogo o el botón "atrás" del Smartphone, se cerrará el cuadro de diálogo y se procederá a la destrucción de la actividad. Si se pulsa el botón "Aceptar", se comprobará que se haya seleccionado alguna opción, en cuyo caso se almacenará el valor escogido. En caso contrario, se volverá a mostrar el mismo cuadro de diálogo, a la espera de una nueva interacción del usuario.

Una vez escogido un valor para la resolución, se creará un nuevo cuadro de diálogo con un campo de texto vacío. En esta ocasión se deberá introducir la dirección IP del servidor. Si se pulsa el botón "Cancelar" del diálogo o el botón "atrás" del Smartphone ocurrirá lo mismo que con el cuadro de diálogo anterior. En caso de pulsar el botón "Aceptar", se comprobará que el campo de texto no esté vacío, en cuyo caso se almacenará el contenido del campo de texto como la dirección IP para las peticiones que se realizarán después. En caso contrario, se volverá a mostrar el mismo cuadro de diálogo, a la espera de una nueva interacción del usuario.

Ahora se procederá a realizar la primera petición al servidor. Si no se obtuviera respuesta o si ocurriese algún error durante esta acción, se mostraría una notificación por pantalla, utilizando la clase *Toast* de la API de Android. La clase Toast permite mostrar mensajes en la pantalla, y se utilizan para proporcionar información al usuario acerca del estado de algún proceso activo en la aplicación. Por ejemplo, cuando se descarga un archivo desde Internet, en pantalla aparece un mensaje "Descargando…" que nos informa que el proceso de descarga ha comenzado.

Tras mostrar la notificación, se lanzaría de nuevo el cuadro de diálogo donde se pedía la dirección IP del servidor. En caso de obtener una respuesta satisfactoria, el siguiente paso será extraer la imagen del mensaje HTTP de respuesta, para después aplicarle la detección facial a dicha imagen y actualizar la imagen mostrada en el layout de la actividad, sustituyendo la anterior por la que se acaba de descargar.

Para mayor información sobre el proceso de peticiones al servidor, extracción y detección facial de las imágenes, puede consultar las secciones 3.1.3 y 3.1.4.

Cuando haya acabado el proceso de sustitución de la imagen mostrada en el layout, se repetirá el proceso de Petición-Extracción de imagen-Detección facial-Actualización de layout hasta que el usuario quiera finalizar la actividad, pulsando el botón "atrás" del Smartphone o hasta que se produzca algún fallo en la conexión, tras lo cual se mostrará una notificación por pantalla y se volverá a lanzar el cuadro de diálogo donde se pedía la dirección IP del servidor.

En la Figura 17 se muestra el diagrama de flujo para esta actividad.

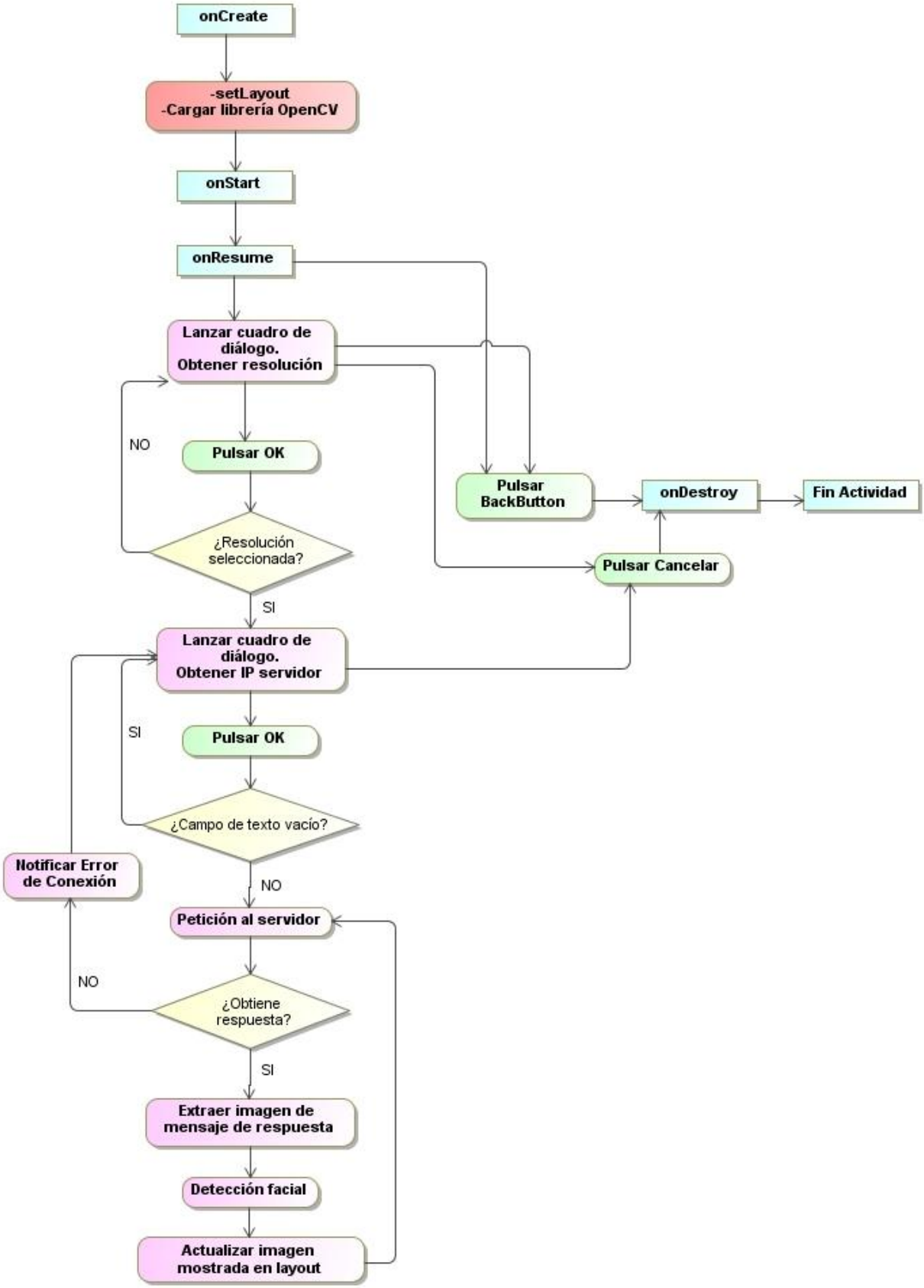

Figura 17. Diagrama de flujo de ConexionActivity

3.1.2.4. Flujo de TestActivity

Dado que esta actividad es similar a ConexionActivity, salvo que proporciona datos acerca del rendimiento obtenido en la conexión con el servidor, el flujo de ambas actividades es prácticamente igual. Por ello, en esta subsección sólo se detallarán las partes exclusivas de esta actividad, indicando su posición en el flujo de la misma.

Cuando la actividad ha sido creada correctamente (se ha establecido el layout y se ha cargado el módulo de OpenCV), se lanza un cuadro de diálogo para que el usuario pueda seleccionar el número total de imágenes que serán descargadas en la secuencia del test. Para ello, este cuadro de diálogo utiliza un widget de la API de Android, llamado *NumberPicker*. Un *NumberPicker* permite al usuario seleccionar un número perteneciente a un intervalo predefinido (en este caso 10-500), deslizando los valores hasta encontrar el deseado o introduciendo directamente el valor mediante el teclado. El valor por defecto es 10 y nunca podrá darse un valor nulo. Es decir, si hacemos click en el valor, lo borramos y le damos a aceptar, se almacenará el valor que estuviese antes del borrado. Esto se debe a que el *NumberPicker* almacena el valor por la posición del intervalo en el que se encuentra. En caso de pulsar el botón "Cancelar" del cuadro de diálogo o el botón "atrás" del Smartphone, se cerrará el cuadro de diálogo y se procederá a la destrucción de la actividad. En caso de pulsar "Aceptar", se almacenará el valor seleccionado y se procederá a obtener la resolución deseada para el test y la dirección IP del servidor de la misma forma que en la clase ConexionActivity.

Cuando se obtiene respuesta del servidor, además de la imagen contenida en el mensaje de respuesta, TestActivity obtiene el tamaño de la misma y se almacena para generar después los resultados del test.

A la hora de actualizar el layout, además de actualizar la imagen, se actualiza el valor del tamaño de la imagen mostrado para que sea acorde a la imagen actual.

Tras actualizar el layout, TestActivity calcula el tiempo transcurrido entre la última actualización del layout y la actual y lo almacena para generar después los resultados del test. Después incrementa el número de descargas realizadas mediante una variable de control. Si dicha variable aún no ha alcanzado el valor estipulado, se procede a realizar una nueva petición. En caso contrario, TestActivity deja de hacer peticiones al servidor y genera las estadísticas del test para mostrarlas por pantalla mediante un nuevo cuadro de diálogo. Aquí aparecerá el tiempo medio entre imágenes y el tamaño medio de las imágenes descargadas durante el test. Tanto si el usuario pulsa el botón "OK" del cuadro de diálogo como si pulsa el botón "atrás" del Smartphone, se cerrará el cuadro de diálogo y se procederá a la destrucción de la actividad.

Para mayor información del proceso de obtención de los resultados del test puede consultar la sección 3.1.5.

Durante la secuencia de descarga para el test, el usuario puede volver a MainActivity en cualquier momento pulsando el botón "atrás" del Smartphone, lo que provocará la destrucción de la actividad.

En la Figura 18 se muestra el diagrama de flujo para esta actividad.

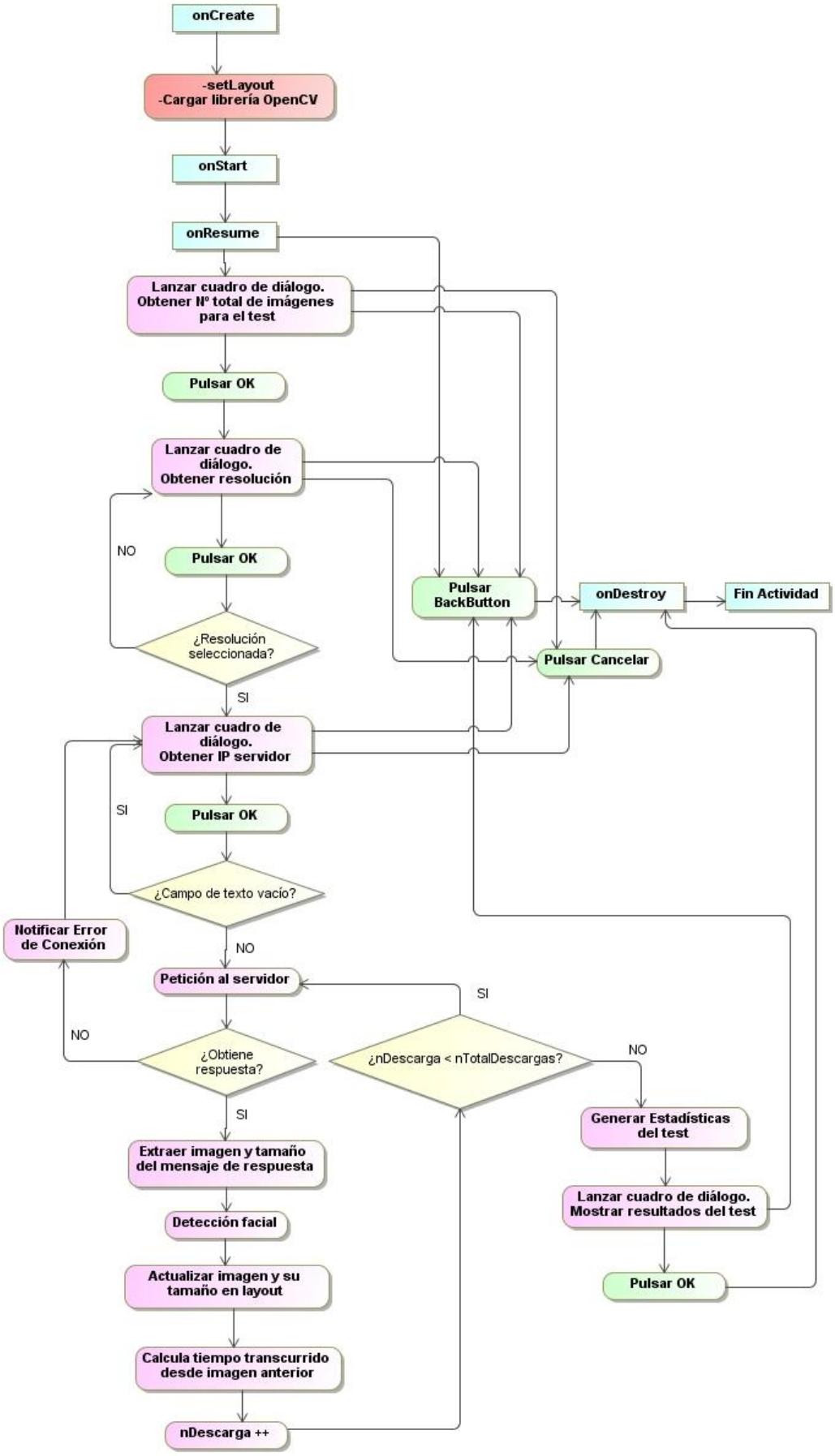

Figura 18. Diagrama de flujo de TestActivity

3.1.3. Peticiones al servidor y extracción de imágenes del mensaje HTTP de respuesta

Las peticiones al servidor se realizan en segundo plano para evitar el bloqueo del hilo principal de la aplicación. Este proceso se lleva a cabo dentro del método *doInBackGround* de la clase interna *procesadoPetición*, de tipo *AsyncTask*.

Para la conexión con el servidor, se crea un objeto *URL* para almacenar la url de la petición. Esta url tendrá el formato "http://IPSERVER/?format=jpeg?resize=RESOLUCION?quality=CALIDAD?". Donde IPSERVER, RESOLUCION y CALIDAD serán los parámetros que tendremos almacenados.

Después se crea un objeto de tipo *HttpURLConnection* para la conexión con el servidor, mediante el método *openConnection* de nuestro objeto *URL* creado anteriormente. Tras fijar algunos parámetros para la conexión (timeout, uso de caché y desactivar la propiedad KeepAlive), se procede a realizar la conexión utilizando el método *connect*.

Cuando se recibe la respuesta, ésta es almacenada en un objeto de tipo *InputStream*. Acto seguido, obtenemos la imagen que se encuentra en el mensaje de respuesta mediante la clase *BitmapFactory*. En esta clase disponemos de un método que decodifica un flujo de entrada a un objeto de tipo *Bitmap*. Los *Bitmap* son imágenes que soportan varios formatos. La propia clase *BitmapFactory* reconoce el formato de codificación de la imagen recibida y la decodifica satisfactoriamente para guardar el resultado en nuestro objeto *Bitmap*.

Una vez obtenida la imagen, procedemos a cerrar la conexión *HttpURLConnection* y el flujo *InputStream*. El *Bitmap* obtenido se pasará como parámetro al método *onPostExecute*, para que sustituir la anterior imagen por la que se acaba de recibir.

En caso de fallo en la conexión, se captura la excepción pertinente y se devuelve el valor NULL al método *onPostExecute*. Este método detectará si el fallo ha sido causado por un reinicio de la conexión por parte del servidor, en cuyo caso volverá a conectar con el mismo. Si la excepción fue lanzada por una pérdida de la conexión, se notificará mediante un *Toast* y se volverá a pedir la IP del servidor para reintentar la conexión.

3.1.4. Proceso de detección facial

La clase Conexión.java implementa el método *detectaCaras*. Este método (heredado tanto por ConexionActivity como por TestActivity) recibe como parámetro un *Bitmap* con la imagen donde se quiere realizar la detección facial, y devuelve un *Bitmap* con una copia de dicha imagen pero con unos recuadros de color verde alrededor de las caras detectadas (si las hubiera).

Excepto los *Bitmap*, todos los objetos mencionados en esta sección pertenecen a la librería de OpenCV.

En primer lugar se crea un *Bitmap* vacío con las mismas dimensiones que la imagen pasada como parámetro. Después se crea un objeto de tipo *Mat* donde se almacenará la imagen pasada por parámetro, mediante el método *bitmapToMat*. El tipo *Mat* pertenece al módulo *Core*, y es un array n-dimensional que puede utilizarse para almacenar vectores y matrices de datos. En nuestro caso será una matriz donde se almacenará la imagen de tal forma que el detector la entienda.

También crearemos un objeto de tipo *MatOfRect*. Este tipo de objeto no es más que una matriz de objetos de tipo *Rect*. Los objetos de tipo *Rect* representan rectángulos, que normalmente son utilizados para señalar algún objeto detectado con OpenCV, dibujándolo alrededor de dicho objeto. En nuestro caso, cada rectángulo representará una cara detectada en la imagen pasada como parámetro. El propio objeto *Rect* dispone de atributos como el alto, ancho y la posición, para poder ser dibujado después. Ambos objetos pertenecen al módulo *Core*.

Ahora es el momento de utilizar el detector. OpenCV dispone de la opción de utilizar un detector adaptado para Java, evitando usar el detector implementado en lenguaje nativo. En este proyecto se utilizará el detector java por su mayor velocidad de detección.

El detector es instanciado en el momento de creación de la actividad. Cuando se carga la librería de OpenCV

correctamente, se accede al fichero "lbpcascade_frontalface.xml". En este fichero encontramos una lista de rectángulos, que representan una base de datos de caras frontales y que será utilizado para reconocer los patrones de caras frontales en la imagen deseada.

El método que utiliza el detector es *detectMultiScale*. Este método recibe como parámetros la imagen convertida al tipo *Mat*, la matriz de rectángulos creada para almacenar los rectángulos que corresponderán a cada cara detectada, un factor de escalado para la imagen, el número de "vecinos" que debe tener un rectángulo para almacenarse (un número elevado evita falsos positivos, pero puede hacer que no se detecten algunas caras), y los tamaños mínimo y máximo de los objetos a detectar.

Una vez terminado el proceso de detección, convertimos la matriz de rectángulos a un vector de rectángulos, para recorrerlo dibujando cada rectángulo en la imagen que está almacenada en el objeto *Mat*. El método utilizado para dibujar los rectángulos es *rectangle*, del módulo *Core*. A este método se le pasa como parámetros la matriz donde se encuentra la imagen, el vértice noroeste y el vértice sureste del rectángulo a dibujar, el color del rectángulo y el grosor del borde.

Por último, se realiza una conversión de *Mat* a *Bitmap* de la imagen donde se ha realizado el proceso de detección, y se devuelve el *Bitmap* como parámetro de retorno del método.

3.1.5. **Obtención del tiempo medio entre imágenes y tamaño medio de imagen**

El tiempo medio entre imágenes se obtiene al dividir el tiempo total correspondiente a la secuencia de descarga de imágenes entre el número total de imágenes descargadas. Así mismo, el tamaño medio de imagen se obtiene al dividir el volumen total descargado durante la secuencia del test entre el número total de imágenes descargadas.

Para medir el tiempo que transcurre entre dos imágenes consecutivas, antes de cada descarga se almacena el tiempo actual y tras la actualización del layout se almacena el tiempo transcurrido restando al nuevo tiempo actual el valor anteriormente almacenado. Después se suma este valor al total del test. El tiempo actual (en milisegundos) se obtiene con el método *currentTimeMillis* del sistema.

Para obtener el tamaño de la imagen descargada, se hace uso del método *getContentLength* del objeto relativo a la conexión, una vez completada la descarga y antes de llamar al método *disconnect*. Después se suma este valor al total del test.

3.2. Aplicación práctica

Con el objetivo de presentar una aplicación para el proyecto, se ha planteado un servicio de videovigilancia de bajo coste. Esta solución está especialmente dirigida a propietarios de pequeños establecimientos (tiendas, bares, restaurantes, almacenes, etc) que podrían tener su negocio vigilado en cualquier momento y desde cualquier lugar. Sin embargo, este servicio también puede ser de interés para cualquier persona que desee tener controlado su hogar durante sus vacaciones, a sus bebés mientras está fuera de casa, una finca en el campo, etc.

Para poner en funcionamiento este servicio, se han utilizado los siguientes elementos:

- Raspberry Pi Modelo B+.
- Tarjeta MicroSD 8GB.
- Cargador de tipo microUSB.
- Webcam compatible con el controlador Linux-UVC.
- Servidor HTTP *MJPEG-streamer*.
- Conexión a internet.

Teclado y monitor (La primera vez obligatorio. Después opcional).

3.2.1. **Raspberry Pi**

Raspberry Pi es un ordenador de precio y tamaño reducidos, fabricado por la fundación Raspberry Pi. Sus dimensiones son 85.6 x 54 milímetros y su precio varía (según el modelo) entre 20 y 40 euros. El principal objetivo de la fundación Raspberry Pi era estimular la enseñanza de ciencias informáticas en las escuelas, acercando a los niños y jóvenes a la programación de manera sencilla y por poco dinero.

En febrero de 2012 se lanzó el primer modelo al mercado (modelo A). En esta primera versión, Raspberry Pi contaba con un procesador ARM (Acorn RISC Machine) a 700 MHz, una memoria RAM de 256 MB compartidos con la GPU (Graphics Processing Unit), un puerto USB, una ranura SD para almacenamiento integrado, salida de vídeo mediante un conector RCA, salida de audio mediante un conector de 3.5 mm y un puerto HDMI (High-Definition Multimedia Interface) para salida de video y audio. También contaba con 40 pines GPIO (General Purpose Input Output) para su utilización con dispositivos lógicos programables, además de una interfaz serie para conectar una cámara (fabricada por la propia fundación Raspberry Pi) y una interfaz serie para conectar un Display LCD (Liquid Crystal Display).

Más tarde se lanzaron los modelos B y B+ con 512 MB de memoria RAM, puerto Ethernet para la conexión de red por cable (antes sólo se podía mediante un módulo Wifi conectado vía USB), y un aumento en el número de puertos USB (2 en el modelo B y 4 en el modelo B+). En el modelo B+ se sustituye la ranura SD por una ranura microSD para ofrecer mayor comodidad.

En febrero de 2015 se lanzó la segunda generación de Raspberry Pi, más rápida y con más recursos. Este nuevo modelo cuenta con un procesador de cuatro núcleos a 900 MHz y una memoria RAM de 1 GB.

Raspberry Pi soporta múltiples sistemas operativos, de los cuales se ha seleccionado la denominada *Raspbian* (una versión de Linux Debian) para nuestro servicio. Merece especial mención el desarrollo de soporte de Windows 10 en la segunda generación de Raspberry Pi, para ser conscientes de la importancia que ha tomado en el mercado.

Para poder utilizar Raspberry Pi por primera vez es necesario disponer de, al menos, un teclado y un monitor para poder interactuar con ella. Además es necesario descargar una de las imágenes de Sistema Operativo disponibles en la web oficial de la fundación y grabarla en una tarjeta microSD de 4GB (como mínimo), que irá insertada en su correspondiente ranura. Una vez conectados los periféricos e introducida la microSD en su sitio, se debe conectar la fuente de alimentación (cualquier cargador microUSB de, al menos, 700mA). Al arrancar, la Raspberry Pi instalará el Sistema Operativo en la microSD guiando al usuario durante todo el proceso mediante un menú de navegación. Una vez concluida la instalación, se deberán introducir los credenciales por defecto (usuario: pi, contraseña: raspberry) para acceder a la terminal. Si se desea, se puede activar el servidor SSH (Secure SHell) de Raspberry Pi. De esta forma, no necesitaremos conectarla a ningún sitio más que a la fuente de alimentación y a un router mediante un cable RJ45 para utilizarla de forma remota (conociendo su dirección IP).

Aunque no es necesario para utilizar este servicio, Raspberry Pi cuenta con un entorno gráfico que puede iniciarse desde la terminal mediante el comando “startx”. Debido a que la memoria RAM está compartida con la GPU, iniciar el entorno gráfico disminuye sus capacidades. Como el servidor que se utilizará funciona mediante línea de comandos, es aconsejable no iniciar el entorno gráfico con el fin de preservar los máximos recursos posibles.

3.2.2. **MJPEG-Streamer**

El servidor de video utilizado en la primera parte del proyecto utiliza la tecnología JMF. Es una tecnología antigua y actualmente sin soporte. Fue abandonada antes de la creación de Raspberry Pi y, por ello, no es posible adaptarla a su arquitectura. Sin embargo, hay muchas aplicaciones de código libre que son

compatibles con Raspberry Pi. Una de ellas es *MJPEG-Streamer*.

Esta aplicación funciona con WebCams compatibles con el controlador Linux-UVC (USB Video Class). Está compuesta por un servidor HTTP accesible desde cualquier navegador web, que transmite las imágenes capturadas por la WebCam mediante un flujo M-JPEG. Además, al igual que el servidor JMF utilizado anteriormente, permite obtener una única imagen en vez de un flujo. De esta forma, las clases implementadas para la comunicación con el servidor JMF/JIMI pueden ser reutilizadas casi en su totalidad para este nuevo servidor.

A continuación se llevará a cabo un análisis de las ventajas y desventajas que presenta este servidor con respecto al servidor JMF/JIMI.

Ventajas:

- Puede utilizarse en Raspberry Pi.
- Es más rápido, puesto que está escrito en C y no tiene interfaz gráfica.
- Sólo es necesario un comando para ponerlo en funcionamiento. En este comando se introducen todas las opciones. El servidor JMF necesita ser ejecutado y después configurado antes de iniciar la transmisión.
- Mayor velocidad de respuesta. Debido a que las imágenes no se manipulan antes de ser enviadas.

Desventajas:

- Limitación en la manipulación de las imágenes. Gracias a la tecnología JIMI, el servidor utilizado en la primera parte podía manipular las imágenes antes de enviarlas como respuesta (cambiar resolución, calidad, aplicar escala de grises, etc). En este caso, la imagen es transmitida tal y como es capturada por la WebCam, con la resolución configurada en el momento de la ejecución del programa.
- Necesita instalación, puesto que C es un lenguaje compilado. En Java, se compila el código conforme lo necesita el intérprete.
- Al no tener interfaz gráfica, puede ser más difícil de utilizar para usuarios inexpertos.

No todas las WebCams son compatibles. Puede consultarse en la página web del controlador Linux-UVC.

3.2.3. **Puesta en marcha**

Para la puesta en marcha se ha utilizado un bar de mi localidad como ejemplo práctico.

Una vez instalado el Sistema Operativo, el servidor de vídeo y activado el servidor SSH (Secure SHell), se realiza la instalación de la Raspberry Pi en un sitio estratégico del establecimiento.

Es necesario abrir el puerto 8080 (por defecto) del router para poder acceder desde una red exterior al servidor de video, de tal forma que las peticiones lleguen a la dirección "IPpública:8080" y sean redirigidas a la IP privada de la Raspberry Pi.

Una vez puesto en funcionamiento, se inicia la conexión mediante SSH desde un PC portátil conectado a la misma red, para arrancar el servidor MJPEG-Streamer y comenzar la transmisión.

En las siguientes figuras se aprecia el montaje de la Raspberry Pi conectada al router mediante un cable RJ45 y a la WebCam por USB, además del escenario de pruebas.

Figura 19. Montaje Raspberry Pi

Figura 20. Montaje Raspberry Pi (II)

Figura 21. Montaje Raspberry Pi (III)

Ahora es el momento de comprobar el funcionamiento del cliente implementado para la recepción de las imágenes del establecimiento.

Se ha reutilizado la mayor parte del código para el cliente del servidor MJPEG-Streamer. Las únicas diferencias son:

- No se puede escoger entre varias resoluciones y calidad de imagen, debido a que este servidor no manipula las imágenes obtenidas de la WebCam.
- La URL a la que accede el cliente para obtener la imagen capturada por la WebCam, que tomará el formato "http://IPpublica:8080/?action=snapshot".

En la siguiente figura se observa la conexión con el servidor a través de la red HSDPA.

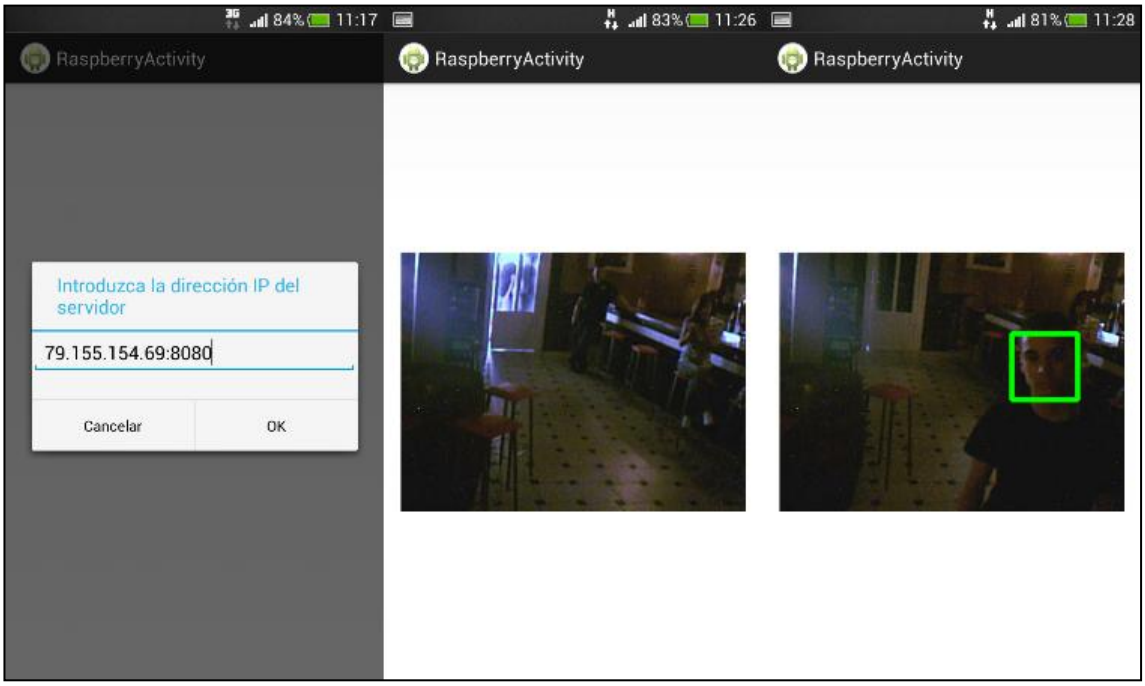

Figura 22. Recepción de imágenes desde Raspberry Pi

3.2.4. **Coste del servicio**

Para analizar el coste real de este servicio, debemos tener en cuenta que el acceso a internet en el establecimiento es el factor de mayor coste. Por ello, este servicio será más interesante para aquellos empresarios que ya tuvieran conexión a internet en su establecimiento. De esta forma, el coste de tener contratado acceso a internet no es derivado del servicio, pues seguiría estando aunque no se utilizase la videovigilancia.

A continuación se detallan los gastos que traería consigo este servicio:

Elemento	Gasto
Raspberry Pi modelo B+	30.95 € (PCComponentes)
WebCam Logitech. QuickCam for Notebooks	10 € (eBay)
Tarjeta MicroSD 8 GB Clase 10	4 € (Amazon)
Cargador microUSB 960 mAh	1.20 € (Amazon)
Consumo energetico (Uso ininterrumpido)	0.003KW x 0.13€/KWh x 24h/dia x 30dias/mes = 0.281 €/mes

Tabla 3. Costes del servicio de videovigilancia.

En la tabla anterior se da por supuesto que no será necesario comprar un teclado usb ni un monitor puesto que

sólo es necesario la primera vez que utilicemos la Raspberry Pi y, en caso de no disponer de un teclado y un monitor o TV con conector HDMI se supone tarea fácil conseguirlos pidiéndolos prestados a cualquier amigo o familiar.

El servidor MJPEG-Streamer es de código libre, por lo que su utilización es gratuita. Así mismo, el cliente Android también es gratuito. Por ello, no se han incluido en la tabla de gastos.

En conclusión, con una inversión inicial de 46.15€ más 0.28€ (de media) al mes, el propietario de un stablecimiento puede tener vigilado su negocio en cualquier momento y desde cualquier lugar.

Si bien es cierto que este servicio no es tan completo como otros muchos del mercado, ofrece un servicio básico de videovigilancia que puede interesar por su bajo coste a muchos propietarios de pequeñas empresas.

4. RESULTADOS OBTENIDOS

Este capítulo tiene como objetivo mostrar los resultados obtenidos en las pruebas realizadas sobre la aplicación. Se analizarán los tiempos medios entre imágenes consecutivas, así como el tamaño medio de las imágenes y la calidad seleccionada. Después se compararán estos resultados con los ofrecidos por J. M. Lora Moreno en su proyecto fin de carrera [1], para comprobar la diferencia entre el uso de las redes de datos 2G y 3G.

MJPG-Streamer no permite configurar cualquier resolución para las imágenes transmitidas, ya que no las manipula. Por lo tanto, sólo podrá ofrecer las resoluciones admitidas por la webcam utilizada. En este caso, la resolución máxima de la webcam utilizada es de 640x480. La siguiente resolución admitida es de 320x240. Se han realizado pruebas con dichas resoluciones, tanto con un servidor como con el otro, para poder comparar los resultados. La calidad de imagen seleccionada para el experimento es de un 85%. Los terminales utilizados para las pruebas han sido un HTC ONE X, con Android 4.2.2 y un Samsung Galaxy Note 4, con Android 4.4.4.

En la tabla y las gráficas siguientes se pueden observar los tiempos medios entre imágenes consecutivas y el tamaño medio de las imágenes descargadas.

Servidor (Resolución)	Tiempo medio entre imágenes	Tamaño medio de imagen
JMF (320x240)	750 milisegundos	9 KB
MJPG-Streamer (320x240)	480 milisegundos	11 KB
JMF (640X480)	1.1 segundos	30 KB
MJPG-Streamer (640x480)	570 milisegundos	40 KB

Tabla 4. Resultados de las pruebas con servidor JMF y MJPG-Streamer

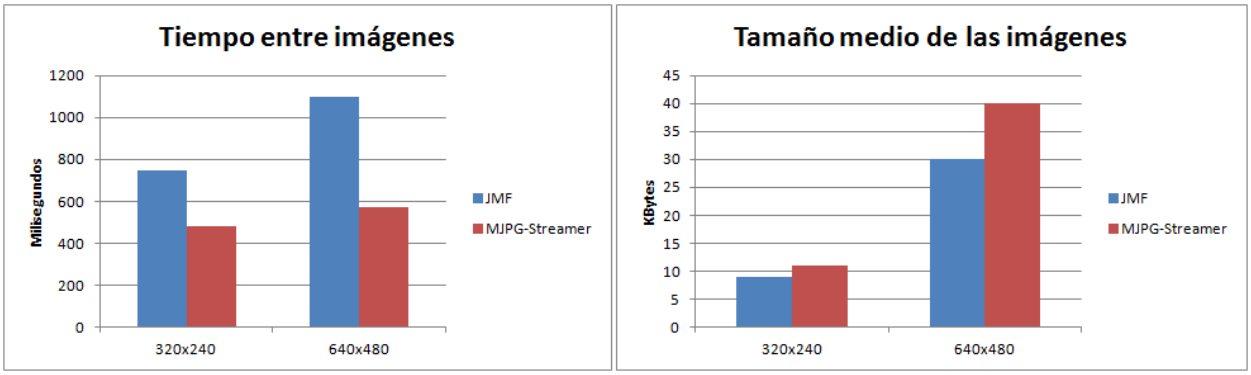

Figura 23. Resultados de las pruebas con servidor JMF y MJPEG-Streamer

Como puede apreciarse, el tiempo entre imágenes utilizando el servidor MJPG-Streamer es bastante menor

que utilizando el servidor JMF (Java Media Framework), aún teniendo un tamaño medio de imagen mayor. Esto es debido, tal y como se adelantó en el anterior capítulo, a que el servidor MJPG-Streamer no manipula las imágenes, sino que las envía tal y como las captura de la webcam.

De esta forma, el servidor MJPG-Streamer ofrece al usuario una mayor fluidez en la secuencia de imágenes recibidas, mientras que el servidor JMF ofrece la posibilidad de elegir la resolución y la calidad de las imágenes a recibir.

Comparando ahora los resultados de este proyecto con los resultados finales del proyecto antes citado, se puede apreciar una gran diferencia en el tiempo medio entre imágenes.

Resolución	Calidad	Tiempo medio entre imágenes	Tamaño medio de imagen
80x80	75%	1ª imagen: 7-10 seg	1.6 KB
80x80	100%	Resto de imágenes: 4-6 seg	6 KB
160x160	75%	1ª imagen: 7-10 seg	4.8 KB
160x160	100%	Resto de imágenes: 7-9 seg	22.5 KB

Tabla 5. Resultados extraídos del proyecto J2ME/GPRS

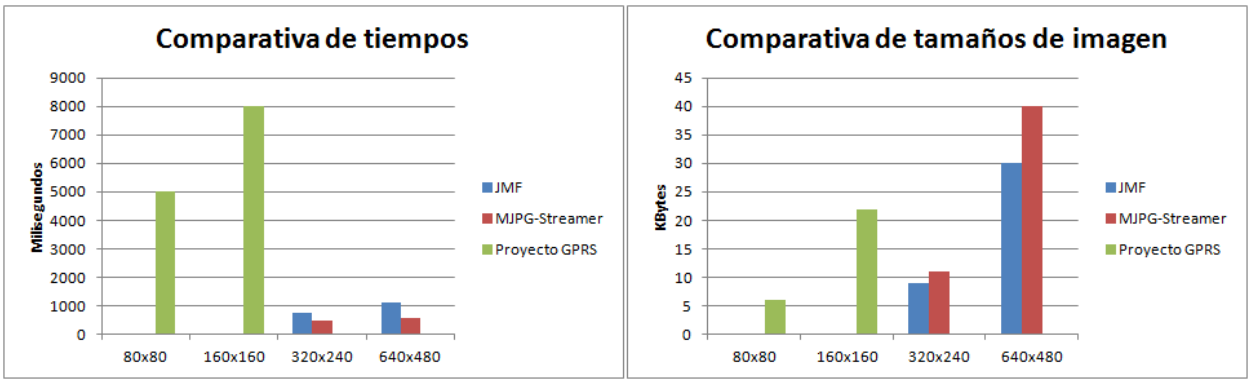

Figura 24. Comparativa de resultados con proyecto GPRS

Como puede observarse, a pesar de la baja resolución y el pequeño tamaño de las imágenes, los tiempos medios entre imágenes utilizando un cliente J2ME (Java 2 Micro Edition) con conexión GPRS (General Packet Radio Service) son mucho mayores que los obtenidos al utilizar un cliente Android con conexión HSDPA (*High Speed Downlink Packet Access*).

5. CONCLUSIONES

Este proyecto ha servido para comprobar el avance en las redes de datos móviles, propiciado por la gran influencia de Internet en nuestras vidas actualmente. En la actualidad, todo el mundo se encuentra conectado a la red desde cualquier sitio y, por ello, es necesario ofrecer cada vez mejores tasas de descarga y menores retardos en el acceso a Internet usando redes de datos móviles.

Para la realización del proyecto, ha sido necesario poner en práctica algunos conocimientos adquiridos a lo largo del Grado. Además, se ha realizado el estudio de otros proyectos y documentación de librerías para el correcto desarrollo y funcionamiento de la aplicación.

Finalmente, la propuesta de una aplicación práctica para el proyecto ha complementado el aprendizaje, llevando el estudio a un escenario real.

5.1. Objetivos logrados

- Implementación de un cliente Android para la recepción de imágenes desde un servidor de vídeo.
- Estudio y utilización de la librería de visión artificial OpenCV (Open source Computer Vision) para añadir la funcionalidad de detección facial al cliente.
- Descubrimiento, estudio y utilización de Raspberry Pi y MJPG-Streamer para el diseño de una propuesta de aplicación práctica para el proyecto. Servicio de videovigilancia de bajo costo. Estudio de ventajas y desventajas, así como del presupuesto necesario para la realización del mismo.
- Comprobación de la mejora en los resultados obtenidos al utilizar HSDPA (*High Speed Downlink Packet Access*) como red de datos móviles para la comunicación con el servidor de vídeo con respecto a los resultados obtenidos en proyectos anteriores utilizando GPRS (General Packet Radio Service).

5.2. Vías de mejora

- Aplicación sobre las redes de datos móviles 4G emergentes en la actualidad.
- Utilización de un servidor de vídeo distinto, capaz de transmitir flujos RTSP (*Real Time Streaming Protocol*) para que el cliente Android pueda recibir el flujo de vídeo directamente en vez de realizar una secuencia de peticiones de imágenes estáticas. Esto conllevaría una implementación distinta del cliente Android.
- Adaptación del cliente Android para su utilización en dispositivos Android 5.0 y superiores. A partir de esta versión de Android, se incluyen nuevas reglas de seguridad en el código que pueden obligar a realizar cambios en el mismo para el correcto funcionamiento en dispositivos con esta versión y superiores.
- Implementación de un servidor para la aplicación práctica, capaz de ofrecer nuevas funcionalidades al servicio (detección de movimientos, alarmas por correo electrónico, etc).

BIBLIOGRAFÍA

[1] J. M. Lora Moreno, Proyecto Fin de Carrera. Implementación de decodificación JPEG para cliente receptor de imágenes en J2ME, Sevilla: Universidad de Sevilla, 2006.

[2] B. Eckel, Thinking in Java, Prentice Hall, 2003.

[3] L. Arenas Hernández, Documentación de la asignatura. Programación Orientada a Objetos en Java., Universidad Nacional Autónoma de México.

[4] J. Tomás Gironés, El gran libro de Android, MARCOMBO S.A., 2012.

[5] The Eclipse Foundation, «Documentación,» [En línea]. Available: http://help.eclipse.org/luna/index.jsp.

[6] IETF, «RFC 1945 - HTTP/1.0,» [En línea]. Available: http://tools.ietf.org/html/rfc1945.

[7] OpenCV, «API y documentación,» [En línea]. Available: http://docs.opencv.org/.

[8] Google Inc, «Documentación y recursos para Android,» [En línea]. Available: http://developer.android.com/index.html.

[9] StackOverflow, «Recursos para programadores,» [En línea]. Available: http://stackoverflow.com/.

[10] J. Korhonen, Introduction to 3G Mobile Communications, Artech House, 2003.

[11] D. Ruíz Macías, Proyecto Fin de Carrera: Servidor para la captura de imágenes y vídeo con tecnología Java (JMF y JIMI), Sevilla: Universidad de Sevilla, 2004.

[12] «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.org/.

[13] «MJPEG-Streamer,» [En línea]. Available: http://sourceforge.net/projects/mjpg-streamer/.

[14] «Controlador Linux-UVC,» [En línea]. Available: http://www.ideasonboard.org/uvc/.

[15] A. Inc., «Distribución de las versiones del SDK,» [En línea]. Available: https://developer.android.com/about/dashboards/index.html.

[16] Oracle, «Java. Sitio oficial.,» [En línea]. Available: https://www.oracle.com/java/index.html.

ANEXO A. MANUALES DE USUARIO

Instalación del plug-in ADT en Eclipse

Este plug-in desarrollado por Google facilita la tarea de programar y depurar aplicaciones Android usando Eclipse. Para instalarlo, será necesario disponer de:

- Eclipse 3.7.2 (Indigo) o superior
- Tener instalado el plug-in JDT (Java Development Tools)
- JDK 6 (Java Development Kit)
- Android SDK (Software Development Kit), disponible en http://dl.google.com/android/android-sdk_r24.1.2-windows.zip

Una vez cumplidos los requisitos, podemos continuar instalando el plug-in:

1. En Eclipse, seleccionar **Help > Install New Software.**
2. Hacer click en **Add**, en la esquina superior derecha.
3. En el cuadro de diálogo que aparece, insertar "ADT Plugin" en el apartado *Name* y la siguiente URL en *Location*: https://dl-ssl.google.com/android/eclipse/
4. Hacer click en **OK.**
5. En el diálogo de Software disponible, seleccionar la casilla de *Developer Tools* y *NDK* Plugins y hacer click en **Next**.
6. En la siguiente ventana, se puede observar una lista de las herramientas que van a descargarse. Sólo debemos hacer click en **Next**.
7. Leer y aceptar los acuerdos de la licencia y hacer click en **Finish**.
8. Cuando se complete la instalación, reiniciar Eclipse.

Tras el reinicio de Eclipse, es necesario especificar la localización del directorio que contiene el Android SDK:

1. En la ventana "Welcome to Android Development" que aparece, seleccionar **Use existing SDKs**.
2. Seleccionar la localización del Android SDK que ha sido descargado y descomprimido con anterioridad.
3. Hacer click en **Next**.

Por último, será necesario actualizar algunos paquetes del SDK. Para ello:

1. En Eclipse, seleccionar **Window > Android SDK Manager**.
2. Aparecerá una ventana con una lista de paquetes. Seleccionamos (al menos):
    a. **Android SDK Tools**, **Android SDK Platform-tools** y **Android SDK Build-tools** del directorio *Tools*.
    b. **SDK Platform** y un emulador (si no queremos utilizar un terminal real) como **ARM EABI v7a System Image** del directorio *Android X.X* (la última versión disponible).
3. Una vez seleccionados todos los paquetes deseados:

a. Hacer click en **Install X packages**.

b. En la siguiente ventana, aceptar el acuerdo de licencia para cada paquete que vaya a descargarse.

c. Hacer click en **Install**.

Ya sólo queda esperar a que se complete la descarga e instalación y Eclipse estará listo para empezar a programar aplicaciones Android. Es importante no cerrar el SDK Manager hasta que se complete la descarga e instalación, o se cancelará dicho proceso.

Añadir la librería OpenCV a un proyecto Android

Para poder utilizar la librería de OpenCV, será necesario descargar el Android NDK previamente. Para ello, sólo hay que acceder al siguiente enlace: http://developer.android.com/tools/sdk/ndk/index.html. Aquí se puede seleccionar el paquete que más convenga en función del sistema operativo donde vaya a instalarse. Una vez descargado (para Windows), al ejecutar el archivo descargado se generará un directorio con el Android NDK que podrá moverse a la ubicación que más convenga.

Ahora es necesario descargar la última versión de OpenCV4Android SDK desde su página oficial: http://sourceforge.net/projects/opencvlibrary/files/opencv-android/ y descomprimirla en un directorio que será utilizado como *Workspace* en Eclipse para el desarrollo de aplicaciones Android que incorporen esta librería.

En este momento, es necesario iniciar Eclipse con el directorio donde hemos descomprimido el SDK de OpenCV como *Workspace*. Cuando se haya iniciado Eclipse, para incluir la librería en el *Workspace*:

1. Click derecho sobre la ventana *Package Explorer* y seleccionar **Import…**
2. Aparecerá un panel con una lista de posibles tipos de proyectos a importar. Seleccionar **General > Existing Projects into Workspace**.
3. En la siguiente ventana, en el campo *Select root directory*, insertar la ubicación del SDK de OpenCV. Esta ubicación será de la forma: \Ubicación donde se descomprimió OpenCV\OpenCV-X.X-android-sdk. Donde X.X es la versión que se haya descargado. Eclipse mostrará la librería y los ejemplos que ésta incluye.
4. Seleccionar la casilla de la librería y pulsar **Finish**.

A partir de este momento, los proyectos Android que se encuentren en este *Workspace* podrán importar la librería OpenCV.

Instalación MJPG-Streamer

Para la instalación del servidor MJPG-Streamer en una Raspberry Pi, con el Sistema Operativo Raspbian instalado, se deben seguir los siguientes pasos:

1. Una vez iniciada la sesión en la Raspberry Pi, ejecutar los comandos “sudo apt-get update” y “sudo apt-get upgrade”.
2. Descargar las librerías necesarias para MJPG-Streamer. “sudo apt-get install libjpeg8-dev imagemagick libv4l-dev”.
3. Descargar MJPG-Streamer. ”svn co https://mjpg-streamer.svn.sourceforge.net/svnroot/mjpg-streamer

mjpg-streamer".

4. Cambiar al directorio para realizar la instalación. "cd mjpg-streamer/mjpg-streamer".
5. Ejecutar "make" para la instalación. Este proceso tardará unos minutos.
6. En el mismo directorio ejecutamos "export LD_LIBRARY_PATH=.". Esta variable será utilizada por el servidor.
7. El servidor está listo para ser ejecutado. "./mjpg_streamer -i "./input_uvc.so –d /dev/video0" -o "./output_http.so -w ./www"" para una ejecución por defecto. Si se desea cambiar alguno de los parámetros, como la resolución o los FPS, hay que añadirlos tras la opción *–i*. Por ejemplo, para 5 FPS el comando sería "./mjpg_streamer -i "./input_uvc.so –d /dev/video0 –f 5" -o "./output_http.so -w ./www""

Desde cualquier navegador se puede acceder al servidor mediante la URL "http://IP:8080", donde IP representa la dirección privada del host donde se está ejecutando el servidor.

Instalación aplicación Android

Para instalar la aplicación Android del cliente implementado, sólo es necesario copiar el fichero "PedcanvazTFG" y pegarlo en el dispositivo Android. Con un gestor de ficheros, localizar el fichero y ejecutarlo.

Es posible que sea necesario cambiar los ajustes del dispositivo para permitir la instalación de aplicaciones provenientes de origen desconocido. Tras realizar la instalación, sólo tendremos que abrir la aplicación y comenzar a utilizarla.

ANEXO B. JAVADOC DE LA APP ANDROID

com.pedcanvaztfg

Class MainActivity

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.MainActivity

- ---

```
public class MainActivity
extends Activity
```

Clase de inicio de la aplicación. Muestra la pantalla principal, con cuatro botones. Dos para conectar con los servidores (JMF y MJPEG-Streamer) y dos para realizar tests de rendimiento de descarga sobre los mismos servidores.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Constructor Summary**

Constructors

Constructor and Description

`MainActivity``()`

- **Method Summary**

Methods

Modifier and Type	Method and Description
static Intent	**createExplicitFromImplicitIntent**(Context context, Intent implicitIntent) Crea Intent explícito para lanzar actividades en Android 5.0 o superior.
void	**onBackPressed**() Finaliza la actividad al pulsar BackButton.
void	**onClickInicio**(View v) Lanza la actividad de conexión con servidor JMF.
void	**onClickInicioTest**(View v) Lanza la actividad de test del servidor JMF
void	**onClickRaspberry**(View v) Lanza la actividad de conexión con servidor MJPEG-Streamer, en Raspberry Pi.
void	**onClickRaspberryTest**(View v) Lanza la actividad de conexión con servidor MJPEG-Streamer, en Raspberry Pi.
protected void	**onCreate**(Bundle savedInstanceState)
boolean	**onCreateOptionsMenu**(Menu menu)
boolean	**onOptionsItemSelected**(MenuItem item)

- **Methods inherited from class java.lang.Object**

  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

- **Constructor Detail**
  - **MainActivity**

    public MainActivity()

- **Method Detail**

- **onCreate**

  ```
  protected void onCreate(Bundle savedInstanceState)
  ```

- **onCreateOptionsMenu**

  ```
  public boolean onCreateOptionsMenu(Menu menu)
  ```

- **onOptionsItemSelected**

  ```
  public boolean onOptionsItemSelected(MenuItem item)
  ```

- **onClickInicioTest**

  ```
  public void onClickInicioTest(View v)
  ```

  Lanza la actividad de test del servidor JMF

  Parameters:

  `v` - boton pulsado

- **onClickRaspberryTest**

  ```
  public void onClickRaspberryTest(View v)
  ```

  Lanza la actividad de conexión con servidor MJPEG-Streamer, en Raspberry Pi.

  Parameters:

  `v` - boton pulsado

- **onClickInicio**

  ```
  public void onClickInicio(View v)
  ```

  Lanza la actividad de conexión con servidor JMF.

  Parameters:

  `v` - boton pulsado

- **onClickRaspberry**

  ```
  public void onClickRaspberry(View v)
  ```

  Lanza la actividad de conexión con servidor MJPEG-Streamer, en Raspberry Pi.

  Parameters:

  `v` - boton pulsado

- **onBackPressed**

  ```
  public void onBackPressed()
  ```

Finaliza la actividad al pulsar BackButton.

- **createExplicitFromImplicitIntent**
- `public static Intent createExplicitFromImplicitIntent(Context context,`

  `Intent implicitIntent)`

  Crea Intent explícito para lanzar actividades en Android 5.0 o superior.

  Parameters:

  `context` - actividad que realiza la llamada

  `implicitIntent` - intent implícito que se desea transformar

  Returns:

  intent explícito

com.pedcanvaztfg

Class Conexion

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.Conexion

- Direct Known Subclasses:

  ConexionActivity, RaspberryActivity, TestActivity, TestRaspberryActivity

---

```
public abstract class Conexion
extends Activity
```

Interfaz de conexion con el servidor. Esta interfaz contendrá las variables y métodos comunes de las clases TestActivity y ConexionActivity de forma que no sea necesario reescribir los métodos idénticos en ambas clases.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Field Summary**

Fields

Modifier and Type	Field and Description
`protected java.lang.String`	**`calidad`**
`protected static org.opencv.core.Scalar`	**`FACE_RECT_COLOR`**
`protected int`	**`finApp`**

protected java.lang.String	**ip**
protected java.io.File	**mCascadeFile**
protected org.opencv.objdetect.CascadeClassifier	**mJavaDetector**
protected org.opencv.android.BaseLoaderCallback	**mLoaderCallback**
protected java.lang.String	**resolucion**
protected static java.lang.String	**TAG**

- **Constructor Summary**

Constructors

Constructor and Description

Conexion()

- **Method Summary**

Methods

Modifier and Type	Method and Description
protected Bitmap	**detectaCaras**(Bitmap img) Método para la detección facial.
void	**finalizarActividad**() Finaliza la actividad.
java.lang.String	**getCalidad**() Obtiene la calidad de imagen fijada
int	**getFinApp**() Comprueba si hay que finalizar la aplicación
java.lang.String	**getIP**() Obtiene la IP fijada
java.lang.String	**getResolucion**() Obtiene la resolución de imagen fijada

void	**obtieneIP**() Función que obtiene la dirección IP del servidor.
void	**obtieneResolucion**() Función que obtiene la resolución que se desea para las imágenes Obtiene la resolución de imagen mediante un cuadro de diálogo AlertDialog.
void	**onBackPressed**() Finaliza la actividad si se pulsa BackButton.
void	**onConfigurationChanged**(Configuration newConfig)
void	**peticion**() Método para ejecutar la Asynctask "procesadoPeticion".
void	**setCalidad**(java.lang.String cal) Fija la calidad de imagen elegida.
void	**setFinApp**(int i) Determina el final de la aplicación.
void	**setIP**(java.lang.String dir) Fija la IP del servidor.
void	**setResolucion**(java.lang.String res) Fija la resolución de las imágenes a descargar.

- **Methods inherited from class java.lang.Object**

  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

- **Field Detail**
  - **TAG**

    protected static final java.lang.String TAG

    See Also:

Constant Field Values

  - **FACE_RECT_COLOR**

    protected static final org.opencv.core.Scalar FACE_RECT_COLOR

- **mCascadeFile**

  ```
  protected java.io.File mCascadeFile
  ```

- **mJavaDetector**

  ```
  protected org.opencv.objdetect.CascadeClassifier
  mJavaDetector
  ```

- **ip**

  ```
  protected java.lang.String ip
  ```

- **finApp**

  ```
  protected int finApp
  ```

- **resolucion**

  ```
  protected java.lang.String resolucion
  ```

- **calidad**

  ```
  protected java.lang.String calidad
  ```

- **mLoaderCallback**

  ```
  protected org.opencv.android.BaseLoaderCallback
  mLoaderCallback
  ```

- **Constructor Detail**

  - **Conexion**

    ```
    public Conexion()
    ```

- **Method Detail**

  - **onConfigurationChanged**

    ```
    public void onConfigurationChanged(Configuration newConfig)
    ```

  - **obtieneResolucion**

    ```
    public void obtieneResolucion()
    ```

    Función que obtiene la resolución que se desea para las imágenes Obtiene la resolución de imagen mediante un cuadro de diálogo AlertDialog. Ofrece varias opciones de resolución.

  - **obtieneIP**

    ```
    public void obtieneIP()
    ```

Función que obtiene la dirección IP del servidor. Obtiene la IP mediante un cuadro de diálogo AlertDialog.

- **peticion**

```
public void peticion()
```

Método para ejecutar la Asynctask "procesadoPeticion". Supone el inicio de la conexión con el servidor, cuando ya se dispone de todos los parámetros de conexión. procesadoPeticion debe ser diferente dependiendo de la clase que la invoque. Cada clase que herede de Conexion sobreescribirá este método para un correcto funcionamiento de las peticiones.

- **finalizarActividad**

```
public void finalizarActividad()
```

Finaliza la actividad. Cada clase que herede de Conexion sobreescribirá este método para finalizar la actividad correspondiente.

- **onBackPressed**

```
public void onBackPressed()
```

Finaliza la actividad si se pulsa BackButton.

- **detectaCaras**

```
protected Bitmap detectaCaras(Bitmap img)
```

Método para la detección facial. Detecta las caras en la imagen pasada como parámetro y la muestra por pantalla con un rectángulo verde alrededor de las caras detectadas.

Parameters:

`img` - de tipo Bitmap, imagen a la que se desea aplicar la detección facial

Returns:

devuelve la imagen con la detección facial aplicada

- **setIP**

```
public void setIP(java.lang.String dir)
```

Fija la IP del servidor.

Parameters:

`dir` - direccion IP del servidor ("IP:Puerto").

- **setResolucion**

```
public void setResolucion(java.lang.String res)
```

Fija la resolución de las imágenes a descargar.

Parameters:

`res` - resolución escogida

- **setFinApp**

```
public void setFinApp(int i)
```

Determina el final de la aplicación. Si el entero que se pasa por parámetro es distinto de 0, se procede a finalizar las actividades de la app.

Parameters:

`i` - entero para determinar si hay que destruir el proceso de la app

- **setCalidad**

```
public void setCalidad(java.lang.String cal)
```

Fija la calidad de imagen elegida.

Parameters:

`cal` - calidad de imagen escogida

- **getIP**

```
public java.lang.String getIP()
```

Obtiene la IP fijada

Returns:

IP del servidor

- **getResolucion**

```
public java.lang.String getResolucion()
```

Obtiene la resolución de imagen fijada

Returns:

resolución de imagen

- **getFinApp**

```
public int getFinApp()
```

Comprueba si hay que finalizar la aplicación

Returns:

entero distinto de cero para finalizar la aplicación

- **getCalidad**

  ```
  public java.lang.String getCalidad()
  ```

  Obtiene la calidad de imagen fijada

  Returns:
  calidad de imagen

---

com.pedcanvaztfg

Class ConexionActivity

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.Conexion
      -
      - com.pedcanvaztfg.ConexionActivity

- ---

```
public class ConexionActivity
extends Conexion
```

Clase para la conexión al servidor JMF.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

  -
  - **Nested Class Summary**

Nested Classes

Modifier and Type	Class and Description
private class	**ConexionActivity.procesadoPeticion** Clase para la conexión con el servidor JMF.

  - **Field Summary**

- **Fields inherited from class com.pedcanvaztfg.Conexion**

```
calidad, FACE_RECT_COLOR, finApp, ip, mCascadeFile,
mJavaDetector, mLoaderCallback, resolucion, TAG
```

- **Constructor Summary**

Constructors

Constructor and Description

`ConexionActivity``()`

- **Method Summary**

Methods

Modifier and Type	Method and Description
void	**`finalizarActividad`**`()` Finaliza la actividad.
void	**`obtieneCalidad`**`()` Obtiene la calidad de imagen.
protected void	**`onCreate`**`(Bundle savedInstanceState)`
void	**`peticion`**`()` Método para ejecutar la Asynctask "procesadoPeticion".

  - **Methods inherited from class com.pedcanvaztfg.Conexion**

    `detectaCaras, getCalidad, getFinApp, getIP, getResolucion, obtieneIP, obtieneResolucion, onBackPressed, onConfigurationChanged, setCalidad, setFinApp, setIP, setResolucion`

  - **Methods inherited from class java.lang.Object**

    `clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

- **Constructor Detail**

  - **ConexionActivity**

    `public ConexionActivity()`

- **Method Detail**

  - **onCreate**

    ```
    protected void onCreate(Bundle savedInstanceState)
    ```

  - **peticion**

    ```
    public void peticion()
    ```

    Método para ejecutar la Asynctask "procesadoPeticion". Supone el inicio de la conexión con el servidor, cuando ya se dispone de todos los parámetros de conexión.

    Overrides:

    `peticion` in class `Conexion`

  - **finalizarActividad**

    ```
    public void finalizarActividad()
    ```

    Finaliza la actividad.

    Overrides:

    `finalizarActividad` in class `Conexion`

  - **obtieneCalidad**

    ```
    public void obtieneCalidad()
    ```

    Obtiene la calidad de imagen. Utiliza un cuadro de diálogo con varias opciones para seleccionar la calidad de imagen.

com.pedcanvaztfg

Class ConexionActivity.procesadoPeticion

- java.lang.Object
  -
  -
    -
      - com.pedcanvaztfg.ConexionActivity.procesadoPeticion
- Enclosing class:

  ConexionActivity

```
private class ConexionActivity.procesadoPeticion
extends
```

Clase para la conexión con el servidor JMF. Hereda de AsyncTask porque es necesario realizar la descarga de imágenes en segundo plano, para no bloquear el hilo principal de la aplicación.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Field Summary**

Fields

Modifier and Type	Field and Description
`private boolean`	**`econrst`**
`private java.lang.String`	**`ipServer`**
`private ConexionActivity`	**`padre`**

  - **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**ConexionActivity.procesadoPeticion**(ConexionActivity parent, java.lang.String ip)

- **Method Summary**

Methods

Modifier and Type	Method and Description
protected Bitmap	**doInBackground**(java.lang.Void... params) Método para descargar imagen.
protected void	**onPostExecute**(Bitmap result) Método para detección facial y actualización de la imagen en pantalla.

  - **Methods inherited from class java.lang.Object**

    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

- **Field Detail**

  - **econrst**

    private boolean econrst

  - **padre**

    private ConexionActivity padre

  - **ipServer**

    private java.lang.String ipServer

- **Constructor Detail**

  - **ConexionActivity.procesadoPeticion**
  - protected ConexionActivity.procesadoPeticion(ConexionActivity parent,
    java.lang.String ip)

- **Method Detail**

- **doInBackground**

  ```
  protected Bitmap doInBackground(java.lang.Void... params)
  ```

  Método para descargar imagen. Toma los parámetros guardados de la conexión y realiza una petición al servidor para recibir la imagen que está transmitiendo en este momento.

  Throws:

  `java.net.SocketException` - si hay un error en la conexión

  `java.io.FileNotFoundException` - sólo para depuración

  `java.lang.Exception` - sólo para depuración

- **onPostExecute**

  ```
  protected void onPostExecute(Bitmap result)
  ```

  Método para detección facial y actualización de la imagen en pantalla. Toma la imagen devuelta por el método procesadoPeticion#doInBackground(), le aplica la detección facial y sustituye la imagen en pantalla por esta última. Si hubo error de conexión, notificará mediante un Toast y volverá a pedir la IP del servidor.

com.pedcanvaztfg

Class RaspberryActivity

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.Conexion
      -
      - com.pedcanvaztfg.RaspberryActivity

```
public class RaspberryActivity
extends Conexion
```

Clase para conectar con servidor MJPEG-Streamer instalado en Raspberry Pi.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Nested Class Summary**

Nested Classes

Modifier and Type	Class and Description
private class	**RaspberryActivity.procesadoPeticion** Clase para la conexión con el servidor MJPEG-Streamer.

  - **Field Summary**
    - **Fields inherited from class com.pedcanvaztfg.Conexion**

```
calidad, FACE_RECT_COLOR, finApp, ip, mCascadeFile,
mJavaDetector, mLoaderCallback, resolucion, TAG
```

- **Constructor Summary**

Constructors

Constructor and Description

`RaspberryActivity`()

- **Method Summary**

Methods

Modifier and Type	Method and Description
void	**`finalizarActividad`**() Finaliza la actividad.
protected void	**`onCreate`**(Bundle savedInstanceState)
void	**`peticion`**() Método para ejecutar la Asynctask "procesadoPeticion".

  - **Methods inherited from class com.pedcanvaztfg.Conexion**

    detectaCaras, getCalidad, getFinApp, getIP, getResolucion, obtieneIP, obtieneResolucion, onBackPressed, onConfigurationChanged, setCalidad, setFinApp, setIP, setResolucion

  - **Methods inherited from class java.lang.Object**

    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

  - **Constructor Detail**

    - **RaspberryActivity**

      public RaspberryActivity()

  - **Method Detail**

- **onCreate**

  ```
  protected void onCreate(Bundle savedInstanceState)
  ```

- **peticion**

  ```
  public void peticion()
  ```

  Método para ejecutar la Asynctask "procesadoPeticion". Supone el inicio de la conexión con el servidor, cuando ya se dispone de todos los parámetros de conexión.

  Overrides:

  `peticion` in class `Conexion`

- **finalizarActividad**

  ```
  public void finalizarActividad()
  ```

  Finaliza la actividad.

  Overrides:

  `finalizarActividad` in class `Conexion`

---

com.pedcanvaztfg

Class RaspberryActivity.procesadoPeticion

- java.lang.Object
  -
  -
    -
      - com.pedcanvaztfg.RaspberryActivity.procesadoPeticion
- Enclosing class:

  RaspberryActivity

```
private class RaspberryActivity.procesadoPeticion
extends
```

Clase para la conexión con el servidor MJPEG-Streamer. Hereda de AsyncTask porque es necesario realizar la descarga de imágenes en segundo plano, para no bloquear el hilo principal de la aplicación.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Field Summary**

Fields

Modifier and Type	Field and Description
`private boolean`	**`econrst`**
`private java.lang.String`	**`ipServer`**
`private RaspberryActivity`	**`padre`**

  - **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**RaspberryActivity.procesadoPeticion**(RaspberryActivity parent, java.lang.String ip)

- **Method Summary**

Methods

Modifier and Type	Method and Description
protected Bitmap	**doInBackground**(java.lang.Void... params) Método para descargar imagen.
protected void	**onPostExecute**(Bitmap result) Método para detección facial y actualización de la imagen en pantalla.

  - **Methods inherited from class java.lang.Object**

    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

- **Field Detail**

  - **econrst**

    private boolean econrst

  - **padre**

    private RaspberryActivity padre

  - **ipServer**

    private java.lang.String ipServer

- **Constructor Detail**

  - **RaspberryActivity.procesadoPeticion**
  - protected RaspberryActivity.procesadoPeticion(RaspberryActivity parent,
    java.lang.String ip)

- **Method Detail**

  - **doInBackground**

    ```
    protected Bitmap doInBackground(java.lang.Void... params)
    ```

    Método para descargar imagen. Toma los parámetros guardados de la conexión y realiza una petición al servidor para recibir la imagen que está transmitiendo en este momento.

    Throws:

`java.net.SocketException` - si hay un error en la conexión

`java.io.FileNotFoundException` - sólo para depuración

`java.lang.Exception` - sólo para depuración

  - **onPostExecute**

    ```
    protected void onPostExecute(Bitmap result)
    ```

    Método para detección facial y actualización de la imagen en pantalla. Toma la imagen devuelta por el método procesadoPeticion#doInBackground(), le aplica la detección facial y sustituye la imagen en pantalla por esta última. Si hubo error de conexión, notificará mediante un Toast y volverá a pedir la IP del servidor.

com.pedcanvaztfg

Class TestActivity

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.Conexion
      -
      - com.pedcanvaztfg.TestActivity

- ---

```
public class TestActivity
extends Conexion
```

Clase para probar el rendimiento de descarga con el servidor JMF.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-

  - **Nested Class Summary**

Nested Classes

Modifier and Type	Class and Description
private class	**TestActivity.procesadoPeticion** Clase para la conexión con el servidor JMF.

  - **Field Summary**

Fields

Modifier and Type	Field and Description
private boolean	**descartarDescarga**
private int	**numDescarga**
private long	**tamTotalDescarga**
private long	**tiempoTotalDescarga**
private int	**totalDescargas**

- **Fields inherited from class com.pedcanvaztfg.Conexion**

  calidad, FACE_RECT_COLOR, finApp, ip, mCascadeFile, mJavaDetector, mLoaderCallback, resolucion, TAG

- **Constructor Summary**

Constructors

Constructor and Description
TestActivity()

- **Method Summary**

Methods

Modifier and Type	Method and Description
void	**actualizaTiempoTotal**(long tDescarga) Método para actualizar el tiempo total de descarga de imágenes, cuando una nueva imagen ha sido descargada
void	**actualizaVolumenTotalDescargado**(long tam) Método para actualizar el volumen de datos descargado en total, cuando una nueva imagen ha sido descargada
void	**finalizarActividad**()

	Finaliza la actividad.
	generaEstadistica()
void	Función para obtener el tiempo medio entre imágenes consecutivas y el tamaño medio de cada imagen recibida
	getDescartarDescarga()
boolean	Determina si hay que descartar la última descarga.
	getTotalDescargas()
int	Obtiene el total de imágenes a descargar en el test
	inicio()
void	Obtiene el número de imágenes a descargar para el test.
protected void	**onCreate**(Bundle savedInstanceState)
	peticion()
void	Método para ejecutar la Asynctask "procesadoPeticion".
	setDescartarDescarga(boolean c)
void	Determina si hubo error en la última descarga.
	setTotalDescargas(int i)
void	Fija el número de imágenes a descargar

- **Methods inherited from class com.pedcanvaztfg.Conexion**

  detectaCaras, getCalidad, getFinApp, getIP, getResolucion, obtieneIP, obtieneResolucion, onBackPressed, onConfigurationChanged, setCalidad, setFinApp, setIP, setResolucion

- **Methods inherited from class java.lang.Object**

  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

  - **Field Detail**

    - **totalDescargas**

      private int totalDescargas

- **tiempoTotalDescarga**

  ```
  private long tiempoTotalDescarga
  ```

- **tamTotalDescarga**

  ```
  private long tamTotalDescarga
  ```

- **numDescarga**

  ```
  private int numDescarga
  ```

- **descartarDescarga**

  ```
  private boolean descartarDescarga
  ```

- **Constructor Detail**

  - **TestActivity**

    ```
    public TestActivity()
    ```

- **Method Detail**

  - **onCreate**

    ```
    protected void onCreate(Bundle savedInstanceState)
    ```

  - **inicio**

    ```
    public void inicio()
    ```

    Obtiene el número de imágenes a descargar para el test. Tras la selección, llama a `Conexion.obtieneResolucion()`

  - **actualizaTiempoTotal**

    ```
    public void actualizaTiempoTotal(long tDescarga)
    ```

    Método para actualizar el tiempo total de descarga de imágenes, cuando una nueva imagen ha sido descargada

    Parameters:

    `tDescarga` - tiempo de la ultima descarga

  - **actualizaVolumenTotalDescargado**

    ```
    public void actualizaVolumenTotalDescargado(long tam)
    ```

    Método para actualizar el volumen de datos descargado en total, cuando una nueva imagen ha sido descargada

Parameters:

`tam` - tamaño de la última imagen descargada

- **generaEstadistica**

```
public void generaEstadistica()
```

Función para obtener el tiempo medio entre imágenes consecutivas y el tamaño medio de cada imagen recibida

- **finalizarActividad**

```
public void finalizarActividad()
```

Finaliza la actividad.

Overrides:

`finalizarActividad` in class `Conexion`

- **peticion**

```
public void peticion()
```

Método para ejecutar la Asynctask "procesadoPeticion". Supone el inicio de la conexión con el servidor, cuando ya se dispone de todos los parámetros de conexión. Aumenta la variable `numDescarga` para controlar el número de imágenes descargadas.

Overrides:

`peticion` in class `Conexion`

- **setTotalDescargas**

```
public void setTotalDescargas(int i)
```

Fija el número de imágenes a descargar

Parameters:

`i` - cantidad de imágenes a descargar

- **setDescartarDescarga**

```
public void setDescartarDescarga(boolean c)
```

Determina si hubo error en la última descarga. Si hay error, la descarga no se toma en cuenta para la secuencia del test.

Parameters:

`c` - true si hay que descartar la descarga

- **getTotalDescargas**

  ```
  public int getTotalDescargas()
  ```

  Obtiene el total de imágenes a descargar en el test

  Returns:

  total de imágenes para el test

- **getDescartarDescarga**

  ```
  public boolean getDescartarDescarga()
  ```

  Determina si hay que descartar la última descarga.

  Returns:

  true si hay que descartar la descarga

---

com.pedcanvaztfg

Class TestActivity.procesadoPeticion

- java.lang.Object
  -
  -
    -
      - com.pedcanvaztfg.TestActivity.procesadoPeticion

- Enclosing class:

  TestActivity

```
private class TestActivity.procesadoPeticion
extends
```

Clase para la conexión con el servidor JMF. Hereda de AsyncTask porque es necesario realizar la descarga de imágenes en segundo plano, para no bloquear el hilo principal de la aplicación. Esta clase se utiliza para conectar al servidor JMF y tomar los datos de descarga para el test.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Field Summary**

Fields

Modifier and Type	Field and Description
`private boolean`	**`econrst`**
`private java.lang.String`	**`ipServer`**

private long	**momentoInicio**
private int	**nDescarga**
private TestActivity	**padre**
private long	**tamDescarga**
private long	**tiempoDescarga**

- **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**TestActivity.procesadoPeticion**(TestActivity parent, java.lang.String ip, int numDesc)

- **Method Summary**

Methods

Modifier and Type	Method and Description
protected Bitmap	**doInBackground**(java.lang.Void... params) Método para descargar imagen.
protected void	**onPostExecute**(Bitmap result) Método para detección facial y actualización de la imagen en pantalla.

  - **Methods inherited from class java.lang.Object**

    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

- **Field Detail**

  - **padre**

    private TestActivity padre

  - **ipServer**

```
private java.lang.String ipServer
```

- **nDescarga**

```
private int nDescarga
```

- **tamDescarga**

```
private long tamDescarga
```

- **momentoInicio**

```
private long momentoInicio
```

- **tiempoDescarga**

```
private long tiempoDescarga
```

- **econrst**

```
private boolean econrst
```

- **Constructor Detail**

  - **TestActivity.procesadoPeticion**
  - `protected TestActivity.procesadoPeticion(TestActivity parent,`
  - `java.lang.String ip, int numDesc)`

- **Method Detail**

  - **doInBackground**

```
protected Bitmap doInBackground(java.lang.Void... params)
```

Método para descargar imagen. Toma los parámetros guardados de la conexión y realiza una petición al servidor para recibir la imagen que está transmitiendo en este momento. Obtiene el tamaño de la imagen a descargar, para el test.

Throws:

`java.net.SocketException` - si hay un error en la conexión

`java.lang.Exception` - sólo para depuración

  - **onPostExecute**

```
protected void onPostExecute(Bitmap result)
```

Método para detección facial y actualización de la imagen en pantalla. Toma la imagen devuelta por el método procesadoPeticion#doInBackground(), le aplica la detección facial y sustituye la imagen en pantalla por esta última. Si hubo error de conexión, notificará mediante un Toast y volverá a pedir la IP del servidor. Obtiene

el tiempo que ha llevado cambiar entre la imagen anterior y la actual, para el test. Comprueba si se han descargado todas las imágenes para el test. Si es así, llamará al método `TestActivity.generaEstadistica()`

---

com.pedcanvaztfg

Class TestRaspberryActivity

- java.lang.Object
  -
  - Activity
    -
    - com.pedcanvaztfg.Conexion
      -
      - com.pedcanvaztfg.TestRaspberryActivity

- ---

```
public class TestRaspberryActivity
extends Conexion
```

Clase para probar el rendimiento de descarga con el servidor MJPEG-Streamer, instalado en Raspberry Pi.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-

  - **Nested Class Summary**

Nested Classes

Modifier and Type	Class and Description
private class	**TestRaspberryActivity.procesadoPeticion** Clase para la conexión con el servidor MJPEG-Streamer.

  - **Field Summary**

Fields

Modifier and Type	Field and Description
private boolean	**descartarDescarga**
private int	**numDescarga**
private long	**tamTotalDescarga**
private long	**tiempoTotalDescarga**
private int	**totalDescargas**

- **Fields inherited from class com.pedcanvaztfg.Conexion**

  calidad, FACE_RECT_COLOR, finApp, ip, mCascadeFile, mJavaDetector, mLoaderCallback, resolucion, TAG

- **Constructor Summary**

Constructors

Constructor and Description
TestRaspberryActivity()

- **Method Summary**

Methods

Modifier and Type	Method and Description
void	**actualizaTiempoTotal**(long tDescarga) Método para actualizar el tiempo total de descarga de imágenes, cuando una nueva imagen ha sido descargada
void	**actualizaVolumenTotalDescargado**(long tam) Método para actualizar el volumen de datos descargado en total, cuando una nueva imagen ha sido descargada
void	**finalizarActividad**()

	Finaliza la actividad.
	generaEstadistica()
void	Función para obtener el tiempo medio entre imágenes consecutivas y el tamaño medio de cada imagen recibida
	getDescartarDescarga()
boolean	Determina si hay que descartar la última descarga.
	getTotalDescargas()
int	Obtiene el total de imágenes a descargar en el test
	inicio()
void	Obtiene el número de imágenes a descargar para el test.
protected void	**onCreate**(Bundle savedInstanceState)
	peticion()
void	Método para ejecutar la Asynctask "procesadoPeticion".
	setDescartarDescarga(boolean c)
void	Determina si hubo error en la última descarga.
	setTotalDescargas(int i)
void	Fija el número de imágenes a descargar

- **Methods inherited from class com.pedcanvaztfg.Conexion**

  detectaCaras, getCalidad, getFinApp, getIP, getResolucion, obtieneIP, obtieneResolucion, onBackPressed, onConfigurationChanged, setCalidad, setFinApp, setIP, setResolucion

- **Methods inherited from class java.lang.Object**

  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

  - **Field Detail**

    - **totalDescargas**

```
private int totalDescargas
```

- **tiempoTotalDescarga**

```
private long tiempoTotalDescarga
```

- **tamTotalDescarga**

```
private long tamTotalDescarga
```

- **numDescarga**

```
private int numDescarga
```

- **descartarDescarga**

```
private boolean descartarDescarga
```

- **Constructor Detail**

  - **TestRaspberryActivity**

```
public TestRaspberryActivity()
```

- **Method Detail**

  - **onCreate**

```
protected void onCreate(Bundle savedInstanceState)
```

  - **inicio**

```
public void inicio()
```

Obtiene el número de imágenes a descargar para el test. Tras la selección, llama a `Conexion.obtieneIP()`

  - **actualizaTiempoTotal**

```
public void actualizaTiempoTotal(long tDescarga)
```

Método para actualizar el tiempo total de descarga de imágenes, cuando una nueva imagen ha sido descargada

Parameters:

`tDescarga` - tiempo de la ultima descarga

  - **actualizaVolumenTotalDescargado**

```
public void actualizaVolumenTotalDescargado(long tam)
```

Método para actualizar el volumen de datos descargado en total, cuando una nueva

imagen ha sido descargada

Parameters:

`tam` - tamaño de la última imagen descargada

- **generaEstadistica**

```
public void generaEstadistica()
```

Función para obtener el tiempo medio entre imágenes consecutivas y el tamaño medio de cada imagen recibida

- **finalizarActividad**

```
public void finalizarActividad()
```

Finaliza la actividad.

Overrides:

`finalizarActividad` in class `Conexion`

- **peticion**

```
public void peticion()
```

Método para ejecutar la Asynctask "procesadoPeticion". Supone el inicio de la conexión con el servidor, cuando ya se dispone de todos los parámetros de conexión. Aumenta la variable `numDescarga` para controlar el número de imágenes descargadas.

Overrides:

`peticion` in class `Conexion`

- **setTotalDescargas**

```
public void setTotalDescargas(int i)
```

Fija el número de imágenes a descargar

Parameters:

`i` - cantidad de imágenes a descargar

- **setDescartarDescarga**

```
public void setDescartarDescarga(boolean c)
```

Determina si hubo error en la última descarga. Si hay error, la descarga no se toma en cuenta para la secuencia del test.

Parameters:

c - true si hay que descartar la descarga

- **getTotalDescargas**

  ```
  public int getTotalDescargas()
  ```

  Obtiene el total de imágenes a descargar en el test

  Returns:

  total de imágenes para el test

- **getDescartarDescarga**

  ```
  public boolean getDescartarDescarga()
  ```

  Determina si hay que descartar la última descarga.

  Returns:

  true si hay que descartar la descarga

---

com.pedcanvaztfg

Class TestRaspberryActivity.procesadoPeticion

- java.lang.Object
  -
  -
    -
      - com.pedcanvaztfg.TestRaspberryActivity.procesadoPeticion

- Enclosing class:

  TestRaspberryActivity

```
private class TestRaspberryActivity.procesadoPeticion
extends
```

Clase para la conexión con el servidor MJPEG-Streamer. Hereda de AsyncTask porque es necesario realizar la descarga de imágenes en segundo plano, para no bloquear el hilo principal de la aplicación. Esta clase se utiliza para conectar al servidor MJPEG-Streamer y tomar los datos de descarga para el test.

Since:

1.0

Version:

1.0

Author:

Pedro Cano Vázquez

-
  - **Field Summary**

Fields

Modifier and Type	Field and Description
`private boolean`	**`econrst`**
`private java.lang.String`	**`ipServer`**

private long	**momentoInicio**
private int	**nDescarga**
private TestRaspberryActivity	**padre**
private long	**tamDescarga**
private long	**tiempoDescarga**

- **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**TestRaspberryActivity.procesadoPeticion**(TestRaspberryActivity parent, java.lang.String ip, int numDesc)

- **Method Summary**

Methods

Modifier and Type	Method and Description
protected Bitmap	**doInBackground**(java.lang.Void... params) Método para descargar imagen.
protected void	**onPostExecute**(Bitmap result) Método para detección facial y actualización de la imagen en pantalla.

  - **Methods inherited from class java.lang.Object**

    clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

- **Field Detail**

  - **padre**

    private TestRaspberryActivity padre

- **ipServer**

  ```
  private java.lang.String ipServer
  ```

- **nDescarga**

  ```
  private int nDescarga
  ```

- **tamDescarga**

  ```
  private long tamDescarga
  ```

- **momentoInicio**

  ```
  private long momentoInicio
  ```

- **tiempoDescarga**

  ```
  private long tiempoDescarga
  ```

- **econrst**

  ```
  private boolean econrst
  ```

o **Constructor Detail**

- **TestRaspberryActivity.procesadoPeticion**
- `protected TestRaspberryActivity.procesadoPeticion(TestRaspberryActivity parent,`
- `java.lang.String ip, int numDesc)`

o **Method Detail**

- **doInBackground**

  ```
  protected Bitmap doInBackground(java.lang.Void... params)
  ```

  Método para descargar imagen. Toma los parámetros guardados de la conexión y realiza una petición al servidor para recibir la imagen que está transmitiendo en este momento. Obtiene el tamaño de la imagen a descargar, para el test.

  Throws:

  `java.net.SocketException` - si hay un error en la conexión

  `java.lang.Exception` - sólo para depuración

- **onPostExecute**

  ```
  protected void onPostExecute(Bitmap result)
  ```

  Método para detección facial y actualización de la imagen en pantalla. Toma la imagen devuelta por el método procesadoPeticion#doInBackground(), le aplica la

detección facial y sustituye la imagen en pantalla por esta última. Si hubo error de conexión, notificará mediante un Toast y volverá a pedir la IP del servidor. Obtiene el tiempo que ha llevado cambiar entre la imagen anterior y la actual, para el test. Comprueba si se han descargado todas las imágenes para el test. Si es así, llamará al método `TestRaspberryActivity.generaEstadistica()`