ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD CARLOS III DE MADRID

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

TÍTULO: DISEÑO Y DESARROLLO DE UN SISTEMA DE VÍDEO-VIGILANCIA BASADO EN TECNOLOGÍA ANDROID

Autor: **Iván Alejandro Fernández Pacheco**
Tutora: **Iria Estévez Ayres**

Junio de 2010

A mis padres, a mi hermana y a Ángela, sin cuyo apoyo no habría podido sacarlo adelante.

Resumen

En la actualidad, la seguridad se está convirtiendo en uno de los ejes fundamentales de la sociedad. El aumento de la violencia y la inseguridad hace que cada vez se invierta más en este sector, investigando y desarrollando nuevos productos, cuyo fin último es prevenir y atajar dichas situaciones.

Por otra parte, en este momento los smartphones, en concreto aquellos que soportan el sistema operativo de Google, Android, proporcionan funcionalidades antes impensables en los teléfonos móviles, soportando, por ejemplo, la transmisión y recepción de vídeo.

Así, el objetivo del proyecto que aquí se presenta es incluir de forma provechosa y sencilla dichos dispositivos en un sistema de videovigilancia, de tal manera que dichos smartphones puedan ser usados para visionar las cámaras del sistema o vídeos de vigilancia previamente guardados en el mismo, sin necesidad de estar en el centro de control de vigilancia, tradicionalmente el punto donde se muestra toda la información del sistema, para tal fin.

En este proyecto, en primer lugar, se ha diseñado la estructura que debe tener un sistema de estas características, que contará con cámaras, servidores de transmisión de vídeo y unidades de almacenamiento de datos, además de los dispositivos Android, así como el esquema de interacción de los mismos. Además, se ha implementado un prototipo de sistema para probar la viabilidad de la idea.

Abstract

Today, security is becoming one of the cornerstones of society. Increased violence and insecurity makes increasingly more investment in this sector, researching and developing new products, whose ultimate goal is prevent and avoid such situations.

Moreover, at this moment smartphones, specifically those that support Google's operating system, Android, provide features previously unthinkable in mobile phones, supporting, for example, video transmission and reception.

Thus, the aim of the project presented here is so helpful and include such simple devices in a video surveillance system, so that these smartphones can be used for viewing the camera's video surveillance system or previously stored therein, without being in the surveillance control center, traditionally the point where it shows all system information, for this purpose.

In this project, firstly, we designed the structure must have a system of this kind, which will cameras, video streaming servers and data storage units, as well as Android devices as well as the scheme of interaction between them. In addition, we have implemented a prototype system to test the feasibility of the idea.

Contenidos

Índice de figuras

Índice de cuadros

Lista de acrónimos

AAC (Advanced Audio Coding)

ADB (Android Debug Bridge)

ALSA (Advanced Linux Sound Architecture)

API (Application Programmmig Interface)

ARM (Advanced Risc Machine)

AWT (Abstract Window Toolkit)

CPU (Central Processing Unit)

CVS (Concurrent Version System)

DDMS (Dalvik Debug Monitor Service)

DSA (Digital Signature Algorithm)

DSS (Darwin Stream Server)

EDGE (Enhanced Data rates for GSM of Evolution)

FIFO (First In First Out)

GEF (Graphic Editing Framework)

GIF (Graphics Interchange Format)

GPL (General Public License)

GPS (Global Positioning System)

GSM (Global System for Mobile Communications)

GUI (Graphical User Interface)

HTML (HyperText Markup Language)

HTTP (HyperText Transfer Protocol)

IDE (Integrated Development Environment)

IP (Internet Protocol)

JDK (Java Development Kit)

JDT (Java Development Tools)

$JPEG$ (Joint Photographic Experts Group)

JVM (Java Virtual Machine)

LED (Light Emitting Diode)

LRU (Least Recently Used)

MMU (Memory Management Unit)

$MPEG$ (Moving Picture Experts Group)

$NPTL$ (Native POSIX Thread Library)

PC (Personal Computer)

PDA (Personal Digital Assistant)

PNG (Portable Network Graphics)

QoS (Quality of Service)

RAM (Random Access Memory)

RDP (Radio Data Packet)

RIM (Research In Motion)

RPC (Remote Procedure Call)

RSA (Rivest, Shamir and Adleman)

$RTCP$ (Real Time Transport Control Protocol)

RTP (Real-Time Transport Protocol)

$RTSP$ (Real Time Streaming Protocol)

SD (Secure Digital)

SDK (Software Development Kit)

$SMPTE$ (Society of Motion Picture and Television Engineers)

SMS (Short Message Service)

SO (Sistema Operativo)

SQL (Structured Query Language)

SSL (Secure Sockets Layer)

SWT (Software Widget Toolkit)

TCP (Transmission Control Protocol)

TLS (Transport Layer Security)

UDP (User Datagram Protocol)

UI (User Interface)

UML (Unified Modeling Language)

URI (Uniform Resource Identifier)

URL (Uniform Resource Locator)

$WYSIWYG$ (What You See Is What You Get)

XML (Extensible Markup Language)

$XMPP$ (Extensible Messaging and Presence Protocol)

Capítulo 1

Introducción

A pesar de que en francés la palabra vigilancia significa literalmente "mirar por encima", el término es usualmente utilizado para toda forma de observación o monitorización, no sólo la observación visual. Es la monitorización del comportamiento, de personas, objetos o procesos dentro de sistemas para la conformidad de normas esperadas o deseadas en sistemas confiables para control de seguridad o social. Esta observación desde una posición de autoridad puede ser encubierta (sin su conocimiento) o manifiesta (tal vez con el frecuente recordatorio del estilo "te estamos vigilando"). La vigilancia ha sido una parte de la historia humana al que la tecnología moderna le ha dado un nuevo campo de operaciones. Para la vigilancia en muchas ciudades modernas y edificios se suelen emplear circuitos cerrados de televisión. Si bien la vigilancia puede ser una herramienta útil para las fuerzas y empresas de seguridad, mucha gente se muestra preocupada por el tema de la pérdida de privacidad.

De un tiempo a esta parte, se ha podido presenciar un aumento de las medidas de seguridad en muchos ámbitos de la vida. El aumento de la violencia, atentados atroces y la búsqueda del bienestar, han sido las razones principales esgrimidas para desarrollar nuevas tecnologías de vigilancia que nos permitan vivir en un mundo más seguro.

Aprovechando estas circunstancias, se ha decidido realizar este proyecto en el que se implementa una pequeña aplicación que permite tener un sistema de videovigilancia para entornos móviles, sin la necesidad de disponer para ello de una central de pantallas en las que el vigilante tenga que estar observando las distintas cámaras de seguridad. De esta forma, se consigue que el vigilante de seguridad que está haciendo la ronda o no se encuentra en el puesto de control por cualquier motivo, pueda seguir viendo lo que está ocurriendo, independientemente de su emplazamiento.

Para ello, se ha empleado el S.O. libre de Android, presentando el 5 de Noviembre de 2007 por Google junto con la fundación Open Handset Alliance, un consorcio de 48 compañías de hardware, software y telecomunicaciones comprometidas con la promoción de estándares abiertos para dispositivos móviles [4]. Al ser un sistema operativo libre y estar basado en un kernel de Linux, cualquiera puede realizar sus

aplicaciones e instalarla en dispositivos móviles.

En el proyecto que aquí se presenta, el mayor problema para implementar este sistema consiste en lograr la monitorización de las cámaras a través del dispositivo móvil. Para ello es necesario tener acceso a una red que nos proporcione unas capacidades que nos permitan ver las imágenes de forma fluida y que las cámaras IP emitan en Streaming. Además, la imagen no debe aparecer en applets ya que el dispositivo es limitado y debido a sus condiciones de origen de diseño, al menos en la versión actual, no los soporta.

El funcionamiento, de modo muy resumido, sería el siguiente: el/los vigilante/s, número variable dependiendo de la extensión a vigilar, llevan consigo un dispositivo móvil con el S.O. Android y la aplicación de videovigilancia instalada. Estos, en un momento dado, pueden consultar el estado del recinto, sin más que visionar las cámaras a través de su terminal móvil, pudiéndolas manejar mediante el joystick del aparato (esta opción dependería de la versión de la aplicación y de si las cámaras proporcionan un API para lograr su manejo), hacer zoom, etc. Otra opción sería que, llevando el aparato guardado en el bolsillo, las cámaras detectasen movimiento (esta opción de momento tampoco está soportada ya que las cámaras utilizadas no proporcionan dicha capacidad). En ese instante, el terminal avisaría al vigilante (como si un SMS llegase a nuestro teléfono móvil), de forma que alerte al personal de seguridad de una intrusión en el recinto. El vigilante recibiría una foto de la cámara que ha detectado movimiento, así como su localización para poder dirigirse a esa zona. De esta forma, el vigilante puede comprobar que no ha sido una falsa alarma, sin mas que verificar dicha imagen.

En resumen, los objetivos principales del proyecto son:

- Configurar un sistema de vigilancia.
- Dar de alta tanto cámaras IP como servidores de streaming.
- Monitorización de las cámaras en un dispositivo con Android.
- Acceso al historial de grabación de las cámaras de seguridad.

Para alcanzarlos, el proyecto se estructura de la siguiente forma: en el capítulo de "Estado del arte" se realiza un análisis somero de Android, el nuevo sistema operativo que ha desarrollado Google para teléfonos móviles. Se estudia su arquitectura, sus características principales, las herramientas que proporciona, como diseñar su interfaz gráfica, el ciclo de vida que siguen sus actividades, los mecanismos de seguridad que proporciona, etc.

Posteriormente se analiza el servidor de vídeo "Darwin Stream Server", estudiando sus características principales así como las funcionalidades que ofrece. Para finalizar este capítulo se estudia el programa "Motion", el cual permite analizar el vídeo que

nos proporciona las cámaras IP de seguridad.

Una vez finalizada la exposición de estas tres herramientas, en el capítulo "Desarrollo del sistema" se presenta el diseño de la aplicación, analizando las pruebas hechas para obtener las conclusiones pertinentes en el capítulo de "Pruebas y resultados", y para finalizar se presentan las posibles líneas futuras para este proyecto en el capítulo "Conclusiones y líneas futuras".

Capítulo 2

Estado del arte

En este capítulo se realiza el análisis Android, estudiando sus distintos componentes y posibilidades. En segundo lugar se estudia "Motion", un software libre para analizar el vídeo proporcionado por las cámaras. Y por último el "Darwin Stream Server", para realizar el *streaming* del vídeo grabado al dispositivo móvil.

2.1. Android O.S.

Google Inc. es la empresa propietaria de la marca Google, cuyo principal producto es el motor de búsqueda del mismo nombre. Aunque su principal producto es el buscador, la empresa ofrece también entre otros servicios: un comparador de precios llamado Google Product Search (antes conocido como "Froogle"), un motor de búsqueda para material almacenado en discos locales (Google Desktop Search), un servicio de correo electrónico llamado Gmail, su mapamundi en 3D Google Earth o un servicio de mensajería instantánea basado en Jabber/XMPP llamado Google Talk. También ha entrado de lleno en el terreno de telefonía móvil, el cual viene de la mano del sistema operativo Android, el cual es la base fundamental de este documento [13].

Una de las ideas con las que se ha desarrollado Android, es la de competir con el iPhone de Apple y el Blackberry de RIM ("Research In Motion"). La presentación de la plataforma Android, que se realizó el 5 de noviembre de 2007, se hizo sobre el G1 de T-Mobile, junto con la fundación Open Handset Alliance, un consorcio de 48 compañías de hardware, software y telecomunicaciones comprometidas con la promoción de estándares abiertos para dispositivos móviles. Se concibió como un sistema operativo orientado a dispositivos móviles basado en una versión modificada del núcleo Linux. Inicialmente fue desarrollado por Android Inc., compañía que fue comprada después por Google, y en la actualidad lo desarrollan los miembros de la Open Handset Alliance (liderada por Google) [14].

Android es una plataforma de programación de software para dispositivos móviles que incluye un sistema operativo, *middleware* y distintas aplicaciones. Google

proporciona el SDK de Android, que provee las herramientas y APIs necesarios para comenzar a desarrollar aplicaciones en esta plataforma, utilizando Java como lenguaje de programación, ejecutándose sobre Dalvik, una máquina virtual propia diseñada para uso embarcado que se ejecuta sobre un kernel de Linux. Para empezar a programar en Android es importante tener conocimientos de Java, así como de programación orientada a objetos. Se puede programar en cualquier plataforma: Windows, MAC o Linux. Para ello hay que descargar el Android SDK, que consta de todas las bibliotecas Java que utiliza Android, el emulador para poder probar las aplicaciones que realizas y multitud de aplicaciones que vienen ya desarrolladas por Google y que seguramente implementen los teléfonos móviles de fábrica que salgan con Android.

Desde el punto de vista del desarrollo de aplicaciones, Google proporciona APIs fáciles de utilizar gracias a la documentación, y los códigos de ejemplo. La idea de Google es que los desarrolladores puedan ayudar a mejorar el sistema, compartiendo ideas y sus desarrollos. Además de Java se utiliza XML para la parte visual, otro lenguaje descriptivo muy fácil de utilizar y que aporta sencillez a la hora de programar en Android.

Google trabaja en la comunicación entre las distintas aplicaciones, compartiendo capacidades de búsqueda. Por ejemplo, si estamos en el reproductor de música, desde él, poder acceder al navegador o a YouTube, para buscar información del artista, más música, o incluso videos musicales. También ofrece varias formas de acceder a las búsquedas en el nuevo sistema operativo. Cada aplicación susceptible de búsquedas incluye un menú para las mismas. Uno de los más problemas más llamativos es la imposibilidad de ejecutar las aplicaciones desde la tarjeta SD de ampliación de memoria, algo que otros sistemas operativos móviles, desde Palm OS hasta Windows Mobile pasando por Symbian, sí permiten (el iPhone no se sabe, puesto que no admite tarjetas de memoria) [6].

Por otra parte, Google implementa un deshabilitador de aplicaciones remoto en Android. Al igual que hizo Apple al reconocer que podría deshabilitar aplicaciones del iPhone de forma remota, Google ha admitido lo mismo. La razón que esgrime es la defensa del usuario ante programas malintencionados o poco seguros. Esto se puede ver como una defensa para el usuario, pero a la vez un punto flaco al limitar el uso del terminal y dar demasiados privilegios a Google [6].

2.1.1. Característiscas generales

Android no diferencia entre las aplicaciones básicas del teléfono y las aplicaciones desarrolladas por terceros. Proporciona un amplio espectro de aplicaciones y servicios. Con los dispositivos construidos en la plataforma Android, los usuarios pueden personalizar completamente el teléfono. Pueden intercambiar pantalla de inicio del teléfono, el estilo del marcador, o cualquiera de las aplicaciones.

Además, proporciona acceso a una amplia gama de bibliotecas y herramientas que pueden usarse para desarrollar aplicaciones. Por ejemplo, Android permite a los desarrolladores obtener la ubicación del dispositivo y que los dispositivos se comuniquen entre sí permitiendo aplicaciones sociales. También incluye un conjunto completo de herramientas que se han construido desde el inicio junto a la plataforma.

Algunas de sus características principales son [3]:

- *Framework* de aplicaciones: permite la reutilización y reemplazo de los componentes. Proporciona un alto grado de personalización por parte del usuario.

- Máquina virtual Dalvik: optimizada para dispositivos móviles. Posee una funcionamiento de llamadas de instancias muy similar a al de Java.

- Navegador integrado: basado en el motor open source *WebKit*. Se le pueden añadir características adicionales en función de las necesidades del desarrollador.

- Gráficos optimizados con una librería de gráficos 2D; los gráficos 3D están basados en la especificación *OpenGL* ES 1.0 (aceleración de hardware opcional).

- *SQLite* Base de datos para almacenamiento estructurado que se integra directamente con las aplicaciones.

- Soporte para medios con varios formatos comunes de audio, video e imágenes planas (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).

- Telefonía GSM (dependiente del *hardware*).

- Dependiendo del *hardware* puede soportar distintos protocolos de comunicaciones entre dispositivos y con acceso a la red como *Bluetooth*, *EDGE*, 3G, y *WiFi*.

- Dependiendo del *hardware* puede incluir distintas características para el ocio como pueden ser la cámara, GPS, brújula, y acelerómetro.

- Amplio ambiente de desarrollo incluyendo un emulador de dispositivo, herramientas para depuración, perfiles de memoria y de ejecución, y un plugin para el IDE Eclipse.

2.1.2. Arquitectura de Android

La arquitectura del sistema operativo de Android, la cual se muestra en la figura 2.1, consta de aplicaciones, marco de aplicaciones, bibliotecas, runtime de Android y su núcleo. Está construida desde el kernel hacia las capas superiores [3].

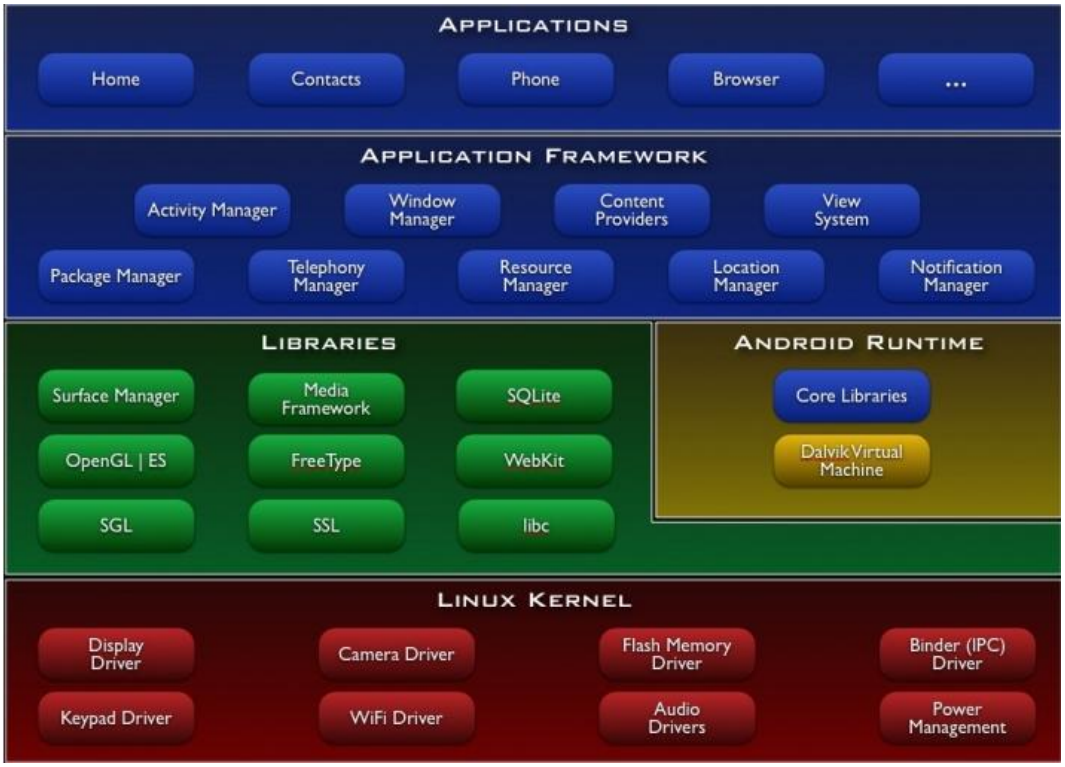


Figura 2.1: Arquitectura de Android [3]

Aplicaciones: Las aplicaciones base incluyen un cliente de email, programa de SMS, calendario, mapas, navegador, contactos... entre otros. Todas las aplicaciones están escritas en el lenguaje de programación Java.

Marco de aplicaciones: Los desarrolladores de aplicaciones tienen acceso completo a las mismas APIs del *framework* usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede hacer públicas sus capacidades y cualquier otra aplicación puede luego hacer uso de ellas (sujeto a reglas de seguridad del *framework*). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

Bibliotecas: Android incluye un conjunto de bibliotecas C/C++ usadas por varios componentes del sistema Android. Estas capacidades se exponen a los desarrolladores a través del *framework* de aplicaciones de Android. Algunas son: *System C library* (implementación biblioteca C standard), bibliotecas de medios,

bibliotecas de gráficos, 3d, *SQLite*, entre otras.

***Runtime* de Android**: se incluye un conjunto de bibliotecas base que proporcionan la mayor parte de las funcionalidades disponibles en las bibliotecas de Java. Cada aplicación Android tiene su propio proceso, con su propia instancia de la máquina virtual Dalvik. Ha sido implementado de forma que un dispositivo puede ejecutar múltiples máquinas virtuales de forma eficiente. Está optimizado para hacer un uso de memoria mínimo. La Máquina Virtual está basada en registros, y ejecuta clases compiladas por el compilador de Java que han sido transformadas al formato `.dex` por la herramienta incluida "dx".

Núcleo - Linux: depende de un Linux versión 2.6 para los servicios base del sistema, como seguridad, gestión de memoria, gestión de procesos, pila de red, y modelo de *drivers*. El núcleo también actúa como una capa de abstracción entre el *hardware* y el resto de la pila de *software*.

Notas sobre Linux para sistemas embarcados

Uno de los cambios fundamentales en Linux para la versión 2.6 viene de la aceptación e inclusión de gran parte del proyecto uClinux (Linux para Microcontroladores) en el *kernel* principal. Esta variante de Linux ha sido un pilar fundamental para su aceptación en el mercado embarcado (*embedded*), y su inclusión en la versión oficial debería aumentar aún más el desarrollo en este campo.

En los sistemas embarcados no se tienen todas las capacidades del *kernel*, debido a limitaciones de *hardware*. La principal diferencia en estas variantes es la ausencia de MMU (*"Memory Management Unit"* o "Unidad de Gestión de Memoria", lo que hace que un sistema operativo pueda trabajar en modo protegido) embebido en el procesador. Aunque suelen ser sistemas Linux multitarea, no tienen protección de memoria ni otras características asociadas. Sin protección de memoria, es posible que un proceso lea los datos de otros procesos, o incluso que los haga colgarse. Esto reduce su utilidad en un sistema multi-usuario, pero los hace ideales para una agenda electrónica (PDA: *Personal Digital Assistant* o "Asistente Digital Personal") de bajo coste o un dispositivo dedicado [7].

Además de soporte para *hardware*, hay también mejoras importantes resultantes de la inclusión de sistemas integrados en el *kernel* principal. Mientras que la mayoría de ellas no son visibles, la robustez general del sistema operativo se ve mejorada por cambios como la capacidad de construir un sistema completo sin soporte de *SWAP* (proceso de intercambio de un proceso de memoria física a disco duro) [7].

Otra prioridad de la nueva versión ha sido hacer que el sistema tenga una respuesta más ágil: no sólo es útil para el usuario final, también para aplicaciones críticas donde se requiere precisión absoluta. A pesar de estos cambios, Linux 2.6 no puede considerarse estrictamente como Sistema Operativo en Tiempo Real, ya que no cumple los rígidos criterios que aseguran que todas las acciones ocurren de forma

predecible; pero los cambios en velocidad de respuesta deberían ser atractivos para todo tipo de usuarios de Linux [7].

Una de las mejoras definitivas en Linux 2.6 es que el *kernel* puede ser interrumpido (*preemptive*). El *kernel* puede ser detenido a la mitad de alguna operación para que otras aplicaciones sigan ejecutándose, aunque se esté en medio de un proceso intensivo de bajo nivel. Claro que habrá situaciones en que el *kernel* no pueda ser interrumpido. En condiciones normales, la mayoría de los usuarios nunca han visto retrasos anormales, ya que no suelen pasar de una fracción de segundo. Aun así, muchos usuarios pueden notar que el sistema se comporta más ágilmente en modo interactivo al activar esta opción; la entrada de usuario parecerá más rápida, incluso en un sistema colapsado [7].

Otro cambio importante en la versión 2.6 es la reescritura de la infraestructura del *kernel* para manejo de hilos, de forma que se pueda ejecutar la Biblioteca Nativa de Hilos POSIX, NPTL [*Native POSIX Thread Library*] sobre ella. Esto conlleva mejores rendimientos en procesadores Pentium Pro y superiores, cuando se hace uso de muchos hilos, y el sector empresarial lleva tiempo esperándolo. Este cambio incluye nuevos conceptos en el espacio de hilos de Linux: grupos de hilos, memoria local para hilos individuales, señales tipo POSIX, y demás. Esto puede perjudicar a las aplicaciones que, en lugar de seguir las especificaciones, dependen de "linuxismos" ya desfasados; así ocurre con ciertas versiones de Java desarrolladas por Sun. Dado que los beneficios son mayores que el perjuicio, y vista la importancia de los promotores, está claro que las aplicaciones importantes serán adecuadas a los nuevos mecanismos poco después de la versión definitiva [7].

Una de las características nuevas de Linux 2.6 más esperadas por los usuarios es la inclusión de ALSA ("Arquitectura Avanzada de Sonido Linux" ("*Advanced Linux Sound Architecture*")) en lugar del sistema de sonido antiguo. La primera mejora en el nuevo sistema es que ha sido diseñado desde el principio para poder ser usado desde distintos hilos, procesos y procesadores, arreglando los problemas de los *drivers* antiguos que no funcionaban fuera del paradigma "un-ordenador-un-procesador" [7].

2.1.3. Anatomía de una aplicación

En las aplicaciones hay 5 bloques: *Activity, Intent, BroadcastReceiver, Service, Content Providers.*

Activity:a partir de ahora "actividad", son las más comunes de los 4 bloques. Habitualmente es una pantalla de la aplicación. Cada actividad es implementada como una clase que extiende a la clase base. La clase mostrará una interfaz de usuario compuesta por *Views* que responderá a los distintos eventos que se produzcan. La mayoría de las aplicaciones están compuestas por varias pantallas, por lo que la mayoría de ella estarán compuestas por varias actividades. El movimiento entre

pantallas se logra empezando una nueva actividad. Cuando una nueva ventana se abre, la anterior queda en pausa o en una pila. El usuario puede navegar hacia atrás a través de las ventanas previamente abiertas [3]. Es decir, básicamente es la parte visual de una aplicación.

Como ejemplo considérese una aplicación SMS. Tiene que tener una pantalla que muestre los contactos para enviar un SMS, una segunda pantalla para escribir el mensaje y otras pantallas para leer mensajes anteriores y cambiar las configuraciones. Cada una de esas pantalla será implementada por una actividad. Cada vez que se inicia otra pantalla se comienza una nueva actividad.

Intent e IntentFilters: son clases especiales para moverse de una pantalla a otra. Describe lo que una aplicación quiere hacer. Lo más importante de esta estructura son la acción y los datos para llevarla a cabo. La navegación entre pantallas se logra mediante la resolución de *Intents*. Para ello usamos *IntentFilter* que buscan entre todas las aplicaciones la que mejor se ajuste al *Intent* [3]. Es decir, es un evento de la aplicación que dice "quiero hacer esto". Un *Intent* puede ser, por ejemplo, "quiero enviar un email". Cuando la aplicación genera esa Intención (muchas están pre-definidas), Android busca la más adecuada para manejar esa intención. Si no está registrada, el usuario debe definirla en el `AndroidManifest.xml` para que se lance la actividad correspondiente.

Por ejemplo, para ver la información de un contacto, se debería crear un *Intent* que requiera una interfaz de usuario compuesta por *Views* y establecer los datos a una URI para representar a ese contacto.

Broadcast Intent Receiver: se puede usar cuando se requiere que se ejecute algo como reacción a un evento externo. No usan la interfaz de usuario. Usan *NotificationManager* para alertar de que algo ha pasado [3].

Por ejemplo, cuando suena el teléfono, o cuando los datos de la red están disponibles, o cuando sea una hora determinada, se ejecutará algo en reacción a estos eventos.

Service: a partir de ahora "servicio", es un código que se ejecuta sin ninguna interfaz de usuario, código que se ejecuta en segundo plano para permitir así la ejecución de otras actividades. Es importante notar que es posible conectarse a un servicio mediante la interfaz que la actividad pone a disposición del usuario [3].

Por ejemplo, el *Media Player*, está reproduciendo música en una lista de reproducción. En él habrá una o mas actividades que permitan al usuario elegir canción y empezar su reproducción. No será manejado por una actividad porque el usuario espera que la música se mantenga sonando mientras navega por otras pantallas. En este caso el *Media Player* podría comenzar un servicio para ejecutar en *background* y mantener la música sonando.

Content Provider: aplicaciones que establecen sus datos en ficheros, bases de datos, etc. Tienen sentido si se quiere compartir datos con otras aplicaciones. Es un servicio que da capacidad a las aplicaciones de comunicarse con otras de manera interna. Es similar a *inter-process communication* [3].

2.1.4. Herramientas de desarrollo

Google proporciona las herramientas necesarias para desarrollar las distintas aplicaciones sobre la plataforma Android. El "Android SDK" incluye una gran variedad de herramientas especialmente diseñadas para desarrollar aplicaciones sobre la plataforma Android. La herramienta más importante es el "Android Emulator" y las "Android Development Tools" plugin para Eclipse, pero el SDK también incluye un surtido de otras herramientas para depuración, empaquetado e instalación de aplicaciones en el dispositivo o emulador [3].

Android emulator: un dispositivo móvil virtual que se ejecuta sobre el ordenador. Se usa para diseñar, depurar y probar las aplicaciones desarrolladas. A continuación se muestran dos imágenes del emulador, figura 2.2, ejecutando el HelloAndroid y el menú [3].

(a) Hello Android (b) Menú

Figura 2.2: Algunas ejecuciones

Hierarchy viewer: permite depurar y optimizar la interfaz de usuario. Proporciona una representación visual de los distintos layouts. Estos dos árboles han sido obtenidos con el Hierarchy viewer. La primera, figura 2.3, se corresponde con la ejecución del HelloAndroid, y la segunda, figura 2.4, con el menú del terminal [3].

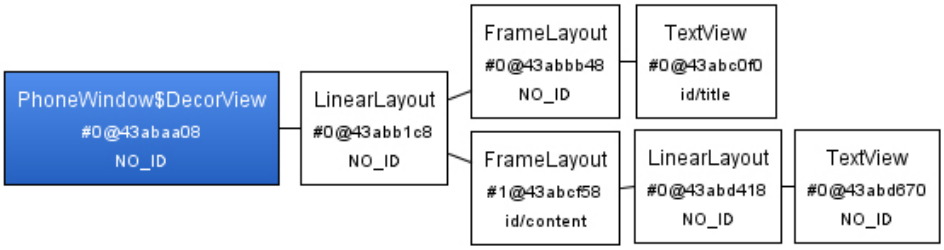


Figura 2.3: Hierarchy Viewer de Hello android [3]

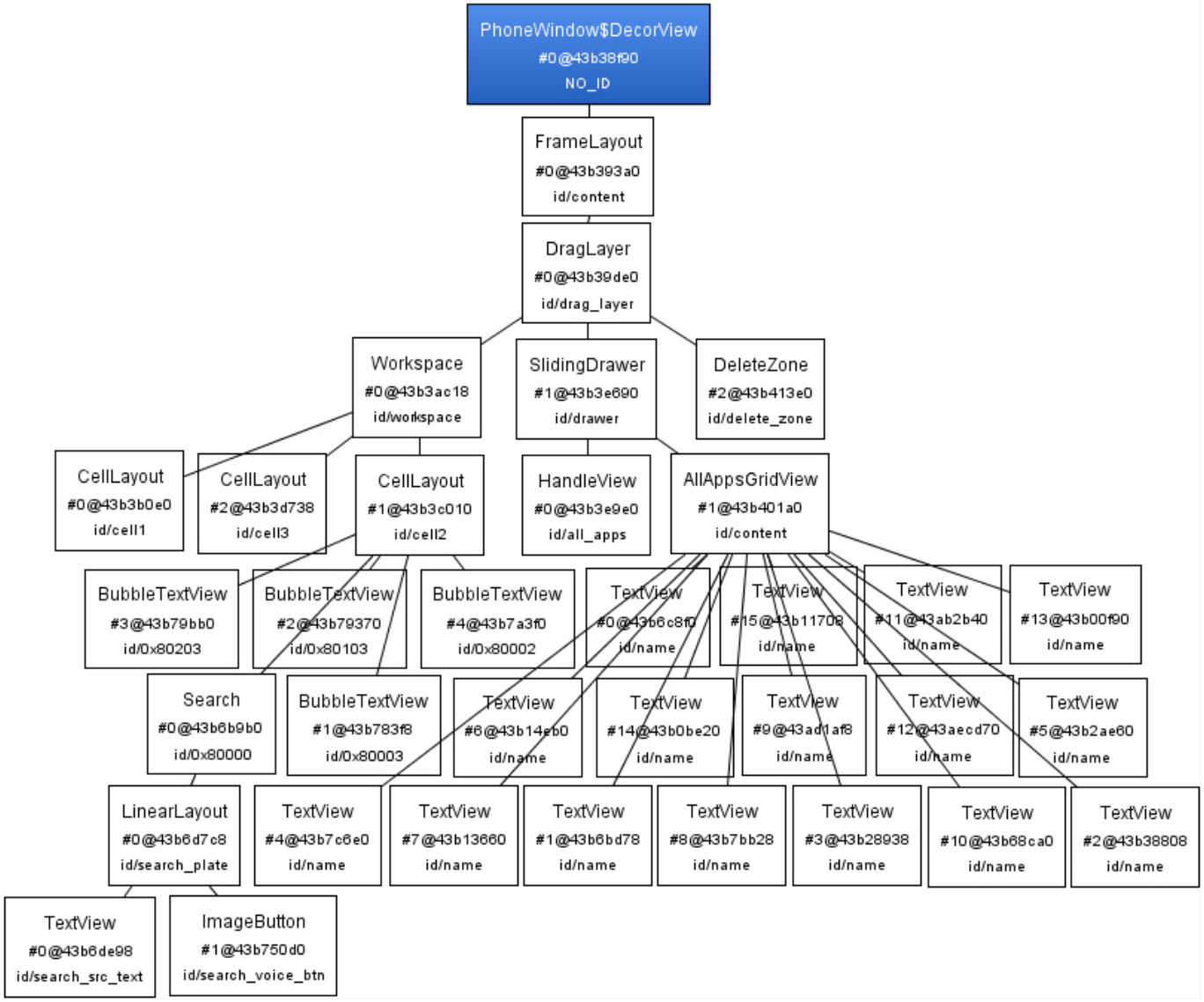


Figura 2.4: Hierarchy Viewer de Menú [3]

Activitycreator: script para generar ficheros Ant que se puedan usar para compilar las aplicaciones. Si se usa el entorno de Eclipse, esto no es necesario [3].

Android Development Tools Plugin for the Eclipse IDE: añade poderosas funcionalidades para el entorno de desarrollo Eclipse, creando y depurando las futuras aplicaciones par Android más fácil y rápidamente [3].

Dalvik Debug Monitor Service: máquina virtual que permite manejar los procesos en el emulador o dispositivo y ayudar en la depuración [3].

Android Debug Bridge: permite instalar tus aplicaciones en el emulador o dispositivo y acceder a ellas mediante línea de comando [3].

Android Interface Description Language: permite crear código para una interfaz entre procesos [3].

sqlite3: permite acceder a los ficheros SQlite usados y creados por las aplicaciones de Android [3].

Traceview: proporciona un análisis gráfico de la traza de datos generada por la aplicación [3].

mksdcard: permite crear una imagen que se puede usar con el emulador, de forma que simule la presencia de un dispositivo de almacenamiento [3].

dx : permite reescribir bytecode en Android bytecode [3].

UI/Application Exerciser Monkey: programa que se ejecuta sobre el emulador o dispositivo que simula secuencias pseudo-aleatoria de eventos de usuario. Se puede usar para probar las aplicaciones que se están desarrollando [3].

Figura 2.5: Draw nine patch[3]

Draw 9-patch: permite crear fácilmente un ninePatch gráfica usando el editor WYSIWYG. Muestra una imagen previa pequeña y remarca el área en donde el contenido está permitido. En la figura anterior, 2.5, se muestra un pequeño ejemplo [3].

2.1.5. Modelo de aplicaciones de Android: aplicaciones, tareas, procesos e hilos

En la mayoría de los sistemas operativos hay una fuerte relación uno a uno entre las imágenes ejecutables en las que está la aplicación, los procesos que se ejecutan y las interacciones del usuario con ello. En Android estas asociaciones son mucho más fluidas [3].

Es importante distinguir entre tarea y proceso. Una tarea es lo que el usuario percibe como una aplicación que ha sido lanzada (normalmente hay un icono en la pantalla que puede ser accedido). Por otro lado, un proceso, es un proceso de bajo nivel en el kernel en donde la aplicación se está ejecutando [3].

Tareas: cuando el usuario ve una aplicación lo que realmente está viendo es una tarea. Es decir, desde el punto de vista del usuario está viendo una aplicación, pero desde la perspectiva del desarrollador está viendo una o más actividades [3].

En algunos casos, Android necesita saber a qué tarea pertenece una actividad cuando esta no ha sido lanzada por una tarea en concreto. Esto se logra mediante la afinidad de tareas, que proporcionan un nombre para la tarea en la que las actividades pretenden ejecutarse. La afinidad de tarea por defecto es el nombre del paquete `.apk` en la que la actividad está implementada.

La forma principal en la que se controla como las actividades interactuan con las tareas es mediante el atributo launchMode y el flag asociado con el Intent.

Procesos: son un detalle de implementación de las aplicaciones y no algo de lo que el usuario sea consciente. Sus usos principales son: mejorar la estabilidad o seguridad, reducir la sobrecarga, ayudar al manejador de recursos del sistema poniendo código pesado en procesos separados que pueden ser eliminados independientemente [3].

Hilos: cada proceso tiene uno más hilos ejecutándose en él. En la mayoría de las ocasiones, se evita crear hilos adicionales en un proceso, manteniendo un único hilo a no ser que este cree los suyos propios. Es importante decir que los nuevos hilos no son creados por instancias de *Activity, BroadcastReceiver, Service o ContentProvider*. Son instanciados en el proceso deseado, en el hilo principal de ese proceso [3].

Aún así hay algunas excepciones a esta regla: llamada a *IBinder* o a una interfaz implementada por *IBinder*, llamada al método main por *ContentProvider* y llamadas

en View y su subclases [3].

2.1.6. Ciclo de vida de una aplicación

En la mayoría de los casos, las aplicaciones de Android se ejecutan en un proceso de Linux. El proceso lo crea la aplicación que necesita alguna parte de su código y se mantiene hasta que no se necesita más y el sistema reclama memoria para otras aplicaciones.

Es raro aunque fundamental saber que el tiempo de vida de una aplicación no está controlado por ella misma, sino que está determinada por el sistema mediante una combinación de partes de la aplicación que se están ejecutando, cuán de importantes son para el usuario y cuánta memoria están consumiendo. Es muy importante tener clara las diferencias entre los distintos componentes de aplicación (*Activity, Service y BroadcastReceiver*). No usarlos correctamente puede implicar que *el sistema los elimine* cuando están realizando alguna actividad importante.

Para determinar los procesos que eliminará el sistema cuando esté bajo de memoria, Android establece una jerarquía, figura 2.6, de importancia basado en los componentes que están ejecutando y en qué estado se encuentran. Estos tipos son[3]:

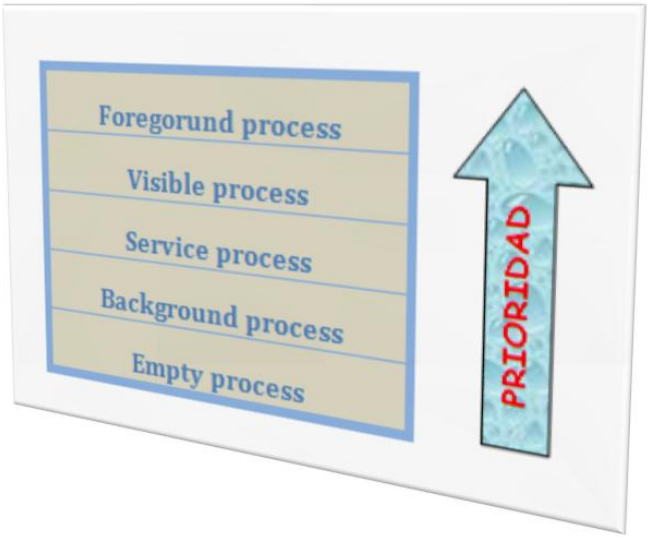


Figura 2.6: Nivel de prioridad

Foreground process: es el que es requerido por lo que está haciendo actualmente el usuario. Varias aplicaciones pueden estar proporcionando distintos elementos para que se consideren '*foreground*'. Habrá solo unos pocos procesos de este tipo en el sistema y solo serán eliminados cuando la memoria sea tan baja que incluso estos

procesos no se puedan seguir ejecutando.

Visible process: el que mantiene una actividad que es visible en la pantalla para el usuario (pero está en pausa). Esto puede ocurrir si el *Foreground process* se muestra en pantalla permitiendo que la actividad anterior sea mostrada detrás de ella. Como un proceso es considerado extremadamente importante no se eliminará a no ser que sea necesario tener todos los *Foregorund Processes* ejecutando.

Service process: procesos que no son vistos directamente por el usuario pero que hacen cosas requeridas por el usuario, por lo que el sistema los mantiene ejecutando a no ser que los 2 tipos de procesos anteriores requieran más memoria.

Background process: proceso que mantiene una actividad pero que no es visible al usuario. No tienen incidencia directa en la actividades que realiza el usuario. El sistema puede eliminar procesos en cualquier momento para liberar memoria para cualquiera de los 3 tipos de procesos anteriores. Se mantiene una lista LRU (*Least Recently Used*, "No usada recientemente") para asegurar que los procesos usados mas recientemente por el usuario son los últimos en ser eliminados.

Empty process: que no tiene ningún componente de aplicación activo, la única razón de mantenerlos funcionando es como caché para mejorar el tiempo de arranque la siguiente vez que una aplicación empiece. El sistema eliminará estos procesos para equilibrar los recursos del sistema.

Una vez que se ha decidido como clasificar el proceso, el sistema tomará su decisión en base al nivel más importante de entre todos los procesos activos. La prioridad de un proceso puede incrementarse a otras dependencias que tenga.

Un ejemplo de todo esto son los *BroadcastReceiver*. Estos lanzan un hilo cuando reciben un *Intent*. Después, devolverá el control a la función que corresponda. En ese momento, el sistema puede considerar que el *BroadcastReceiver* ya no está activo porque ya ha cumplido su función, y por lo tanto no se necesita más, a no ser que otros componentes de aplicación sigan activos en él. Por tanto el sistema eliminará el proceso en cualquier momento para liberar memoria, por lo que termina el hilo lanzado que estaba ejecutando en el proceso. Para evitar esto se debe comenzar un nuevo servicio a partir del *BroadcastReceiver* para que el sistema vea que sigue activo.

2.1.7. Implementar una interfaz de usuario

A la hora de diseñar la interfaz gráfica de la aplicación, se debe tener en cuenta la estructura jerárquica de los elementos en pantalla así como los elementos comunes entre los distintos objetos.

JERARQUÍA DE LOS ELEMENTOS DE PANTALLA

Views: es una estructura de datos cuyas propiedades permiten establecer el layout y el contenido de un area rectangular. Permite manejar medidas, focalizar elementos, scrolling, etc. Sirve de base para widgets (conjunto de subclases enteramente implementadas para mostrar elementos de pantalla interactivos).

Viewgroups: objeto cuya función es almacenar y manejar conjuntos subordinados de objetos Views y Viewgroups.

Tree-Structured UI: en la figura 2.7 se muestra el árbol que representa las relaciones entre los 'views'y 'Viewgroups'.

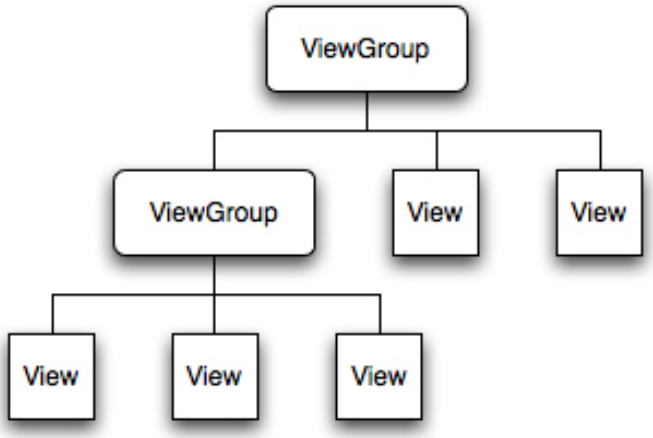


Figura 2.7: Árbol de view y viewgroup nodos [3]

LayoutParams: esta subclase contiene propiedades que definen el tamaño y la posición de los hijos. En la figura 2.8 se muestra la jerarquía de los elementos de la clase.

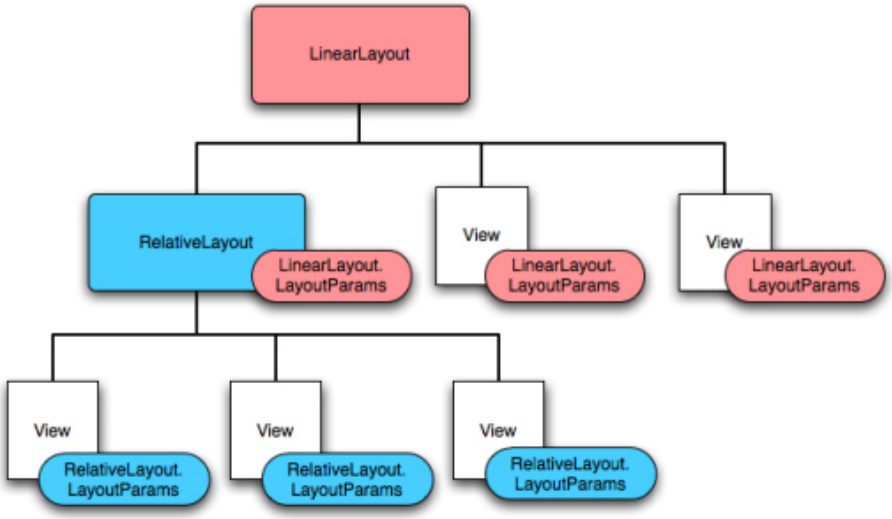


Figura 2.8: Posiciones relativas de los nodos hijo [3]

COMMON LAYOUT OBJECTS

Frame Layout: Es el objeto de layout más simple. Es un hueco en blanco reservado en la pantalla que se puede rellenar más tarde. Los elementos hijo se ponen en la esquina superior izquierda. No se puede especificar una posición concreta.

LinearLayout: alinea todos los hijos en una misma dirección, vertical u horizontal. Se apilan uno detrás de otro. Permite establecer la altura de los hijos.

TableLayout: posiciona a los hijos en filas y columnas. No muestra las líneas de las filas ni de las columnas.

AbsoluteLayout: permite especificar las coordenadas exactas de los hijos, siendo la coordenada (0,0) la esquina superior izquierda.

RelativeLayout: permite establecer las posiciones relativas entre los hijos, diferenciándolos por su identificador.

DISEÑAR LA PANTALLA EN XML

Diseñar una pantalla mediante código puede ser tedioso. Android soporta XML para diseñar pantallas. Android define un montón de elementos personalizados, cada uno representando a una subclase específica de View. Se pueden crear pantallas igual que se diseñan ficheros HTML. Cada fichero describe un único elemento, pero este puede ser una vista simple, un layout que contenga unos cuantos hijos. Cuando se compila la aplicación, Android compila cada fichero en recurso un `android.view.View`.

ESCUCHANDO NOTIFICACIONES DE LA INTERFAZ DE USUARIO

Algunas notificaciones son expuestas y llamadas por Android automáticamente. Algunas llamadas, como los clicks del botón, deben ser registradas manualmente.

USAR TEMAS EN UNA APLICACIÓN

Si no se especifica un tema, Android usará el tema por defecto. Se puede crear uno propio, para posteriormente establecerlo en el fichero XML o programando. Si se elige esta última opción, es necesario tener creado el tema antes de crear las distintas vistas usadas en la interfaz.

2.1.8. Bloques de Android

Para analizar la estructura de Android, podemos dividirlo en los siguientes bloques funcionales [3]:

1. **AndroidManifest.xml**: es el fichero de control que le dice al sistema que hacer con los elementos del nivel más alto que se han creado.

   Este fichero es requerido por todas las aplicaciones. Contiene los valores globales del paquete que se está usando, incluyendo los componentes de aplicación., la implementación de las clases para cada componente, que tipo de datos puede manejar y cuando pueden ser lanzados.

   Hay que destacar que están incluidos los *Intent Filters*. Estos describen dónde y cuándo puede comenzar una actividad. Cuando una actividad quiere realizar algo, como abrir una página Web, crea un *Intent*. Éste puede tener descriptores de lo que se quiere hacer, qué datos quieres para hacerlo, que tipo de datos... Después de esto, Android compara está información del *Intent* con el *Intent Filter* de cada aplicación, los compara y elige el más adecuado para realizar la operación especificada por el llamante.

2. **Actividades**: fundamentalmente son objetos que tienen un ciclo de vida (ver figura 2.9). Es un trozo de código que hace algo. Si es necesario puede mostrar una interfaz de usuario. Interactuan con el usuario, por lo que ponen una ventana para que el usuario pueda diseñar su interfaz de usuario.

   Aunque normalmente se presentan al usuario en ventanas a pantalla completa, se las puede usar de otras maneras: como ventanas flotantes o embebidas

dentro de otra actividad.

Las actividades del sistema se manejan en una pila. Cuando se crea una nueva, se pone arriba del todo. Las previas permanecen debajo. A continuación se muestra el ciclo de vida de una actividad.

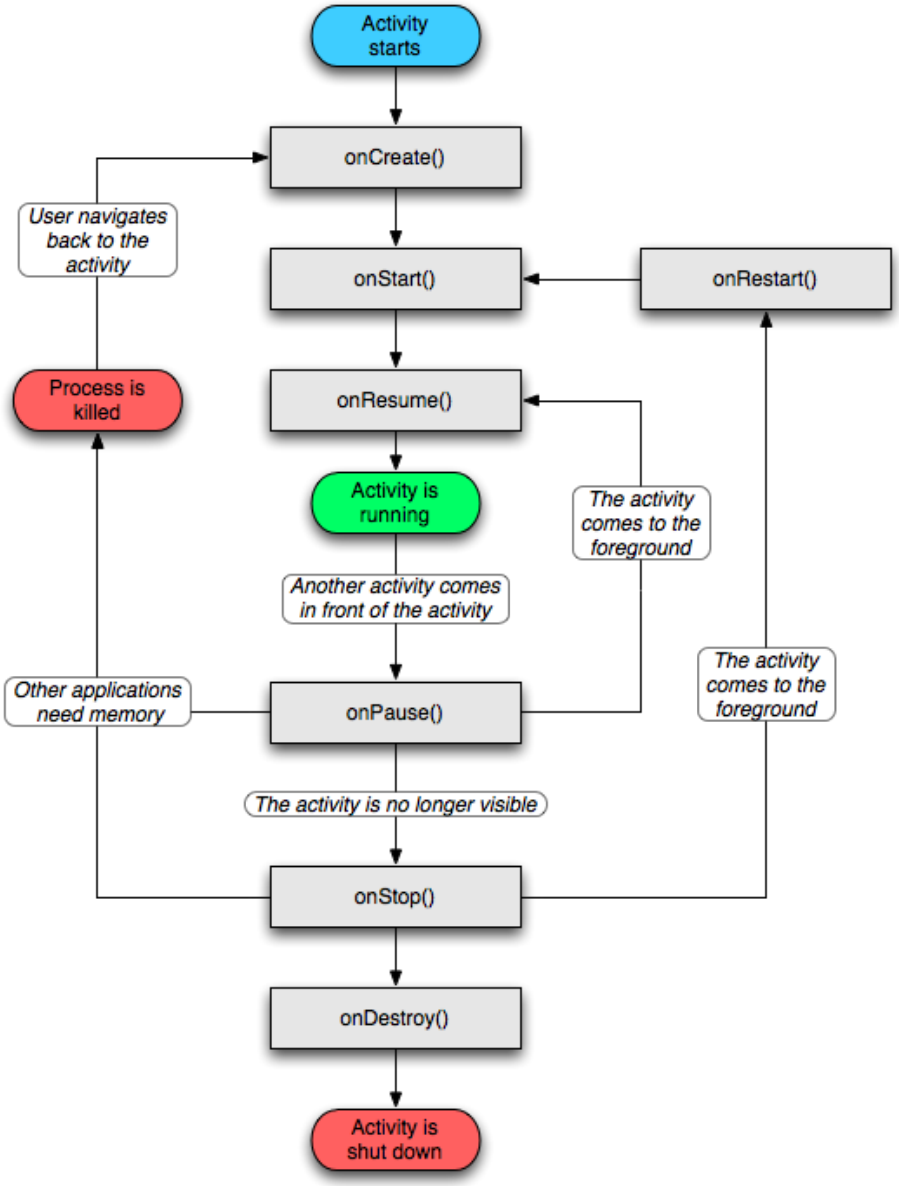


Figura 2.9: Ciclo de vida de Activity [3]

Las actividades tienen 4 posibles estados [3]:

a) Cuando una actividad está en *foreground* (encima de la pila) se dice que está activa o ejecutando.

b) Si ha perdido el foco, pero todavía es visible (porque una nueva actividad, no maximizada o transparente, tiene ahora el foco), se dice que está pausada. Aún así está viva, mantiene la información de estado, pero puede ser eliminada por el sistema en caso de que quede muy poca memoria libre.

c) Si ésta está oscurecida por otra y ha perdido completamente el foco, está parada. mantiene la información de estado, pero no es visible para el usuario, y puede ser eliminada por el sistema cuando se requiere más memoria.

d) Cuando una actividad está pausada o parada, el sistema puede pedirla que acabe o simplemente eliminarla de la memoria. Cuando se vuelve a mostrar al usuario debe haberse reiniciado y restaurado completamente a su estado previo.

Si cambia la configuración del dispositivo, algo que está mostrando la interfaz de usuario necesitará actualizarse para corresponderse con esa nueva configuración. Debido a que las actividades son el mecanismo primario para interactuar con el usuario, éstas incluyen soporte especial para manejar estos cambios de configuración.

A menos que se especifique otra cosa, un cambio de configuración provocará que la actividad se destruya, siguiendo el curso normal de su ciclo de vida [3].

3. ***Views***: objeto que sabe cómo dibujarse en la pantalla. Las interfaces de usuario están compuestas por árboles de *Views*. La geometría de un *View* es un rectángulo, que tiene unas coordenadas en la pantalla, empezando a contarse por la esquina superior izquierda. Se puede especificar su tamaño.

4. ***Intents***: es un mensaje de objeto que representa la intención de hacer algo. Por ejemplo, si la aplicación quiere mostrar una página web, expresa su intención de hacer eso creando un *Intent* y manejándolo. El sistema localiza el trozo de código que sabe manejar esa intención y la hace funcionar.

5. **Servicios**: es código que corre en *background*. Puede ejecutar su proceso en el contexto de otros procesos de aplicación. Por ejemplo cuando se usa el reproductor de música y mientras tanto se vuelve a pantallas anteriores para hacer otras cosas. Mientras se hace eso, la música seguirá sonando.

6. ***Notifications***: es un icono pequeño persistente que aparece en la barra de estado. Estos podría ser por ejemplos SMS, apareciendo por ejemplo un

sobrecejo. El usuario puede interactuar con él para recibir información. Estas son los mecanismos preferidos para alertar al usuario cuando pasa algo. También se puede avisar al usuario mediante LED´s en el dispositivo, con sonidos, vibraciones...

7. ***ContentProviders***: es un lugar para almacenar datos que proporciona acceso a datos del dispositivo. Encapsulan datos y se los proporcionan a las aplicaciones. Solo se necesita si necesitamos compartir datos entre multiples aplicaciones. Si no se necesita compartir los datos se puede usar una base de datos directamente.

2.1.9. Almacenaje, recuperación, y presentación de datos

Un sistema operativo proporciona un sistema de ficheros común que cualquier aplicación puede usar para almacenar y leer archivos que pueden ser leídos por otras aplicaciones (con algunos ajustes de control de acceso). Android usa un sistema diferente: en Android, todos los datos de aplicación (incluyendo archivos) son internos a esa aplicación.

Sin embargo, Android también proporciona un modo para presentar sus datos internos a otras aplicaciones [3]:

Preferences: mecanismo ligero para establecer y recuperar parejas de datos primitivos.

Files: poner los ficheros en el dispositivo o en otros medios extraíbles.

Databases: el API contiene soporte para SQLite. La aplicación puede crear y usar esta base de datos.

ContentProviders: componente opcional de la aplicación que lee/escribe cuando accede a datos privados de una aplicación.

Network: también se puede utilizar la red para poner y recuperar datos.

2.1.10. Modelo de seguridad

Android es un sistema multiproceso donde cada aplicación se ejecuta sobre su propio proceso. La mayor parte de la seguridad entre aplicaciones y el sistema se hace cumplir en el nivel de proceso de Linux, como los identificadores de grupo y usuario que son asignados a las aplicaciones. También se proporcionan más características de seguridad a través de mecanismos de permisos que ofrecen restricciones en operaciones específicas [3].

ARQUITECTURA DE SEGURIDAD

En el punto central de la arquitectura de seguridad de Android, por defecto, ninguna aplicación tiene permisos para realizar ninguna operación sobre él, lo que podría provocar graves daños en otras aplicaciones, el sistema operativo, etc. Un proceso de aplicación es una caja segura. No puede molestar a otras aplicaciones excepto si se han declarado los permisos explícitamente. Estos permisos pueden ser manejados de diferentes maneras, típicamente permitiendo o no mediante certificados, o promovido por el usuario. Los permisos requeridos por una aplicación son declarados estáticos.

FIRMAS DE APLICACIONES

Todas las aplicaciones de Android deben ser firmadas con un certificado cuya clave la tiene el desarrollador. Este certificado identifica al autor de la aplicación. No necesita ser firmado por una autoridad certificadora. Es típico que sean autofirmados. Simplemente es utilizado para establecer relaciones entre aplicaciones.

IDENTIFICADOR DE USUARIO Y ACCESO A FICHEROS

Cada paquete instalado se le da su propio identificador de usuario, creando una caja para prevenir que otras aplicaciones interfieran, o que él interfiera en otras.

Como la seguridad se hace cumplir a nivel de proceso, el código de dos paquetes, normalmente, no pueden correr sobre el mismo proceso, ya que necesitan correr como usuarios distintos.

USANDO PERMISOS

Una aplicación básica, no tiene permisos asociados a ella, es decir, no puede hacer cosas que repercutan negativamente en la experiencia de usuario o en los datos del dispositivo. En el momento de la instalación, los permisos requeridos por la aplicación, son garantizados por el instalador del paquete, basado en verificación contra las firmas de las aplicaciones declarando aquellos permisos y/o interacciones con el usuario. No se verifica nada mientras la aplicación está corriendo.

Un permiso en particular se debe hacer cumplir: cuando lo llama el sistema, cuando se empieza una actividad, cuando se envía y reciben broadcasts, cuando se opera y accede a un Content Provider

DECLARAR Y HACER CUMPLIR LOS PERMISOS

Para hacer cumplir los permisos se deben declarar en AndroidManifest.xml. Los permisos de Activity, Service, BroadcastReceiver restringen quien puede realizar esas actividades, y los permisos de Content Provider restringen quien puede acceder a los datos en el ContentProvider.

En el caso de los 'BroadcastReceiver'se puede exigir que las aplicaciones en el receptor también tengan los permisos necesarios.

PERMISOS URI

El sistema de permisos estándar descrito puede no ser suficiente en el caso de usar Content Providers. Estos objetos pueden querer protegerse con permisos de lectura y/o escritura, mientras sus clientes directos también tienen que dar URIS específico a otras aplicaciones para funcionar sobre ellas.

2.1.11. Creación y manipulación de hilos

Los hilos, al igual que en cualquier programa Java, tienen dos formas de crearse.

1. Heredando de Threads.

```
public class Clase1{
        .
    Hilo h = new Hilo();
    h.start();
}

public class Hilo extends Thread{
        .
    public void run(){
        .
        .
        .
    }
}
```

2. Implementando la interfaz Runnable

```
public class Clase1 implements Runnable{
        .
    public void run(){
        .
        .
        .
    }
}
```

```
public class Clase2{
        .
        Clase2 c = new Clase2();
        Thread h=new Thread(c);
        h.start();
}
```

Para su uso de forma correcta, se debe utilizar una serie de métodos para poder evitar las condiciones de carrera. Entre ellos se tienen los métodos de espera y notificación *(wait() y notify())* y los semáforos, que mediante sus operaciones *P(acquire()) y V(release())*, permiten el acceso de forma sincronizada a la sección crítica del código.

El SDK de Android proporciona un API con clases y métodos que permiten implementar acciones concurrentes. Entre ellas la clase "*Semaphore*", la cual proporciona una serie de métodos (P's y V's mencionadas anteriormente).

Hay que destacar que las operaciones de espera, tanto *wait()* como *acquire()* tienen que estar encerradas en una estructura "*try-catch*" para poder capturar las distintas excepciones que pudieran producirse.

Otra opción para trabajar con estructuras concurrentes es el uso del modificador "*Synchronized*". Usándolo de forma combinada con un objeto, utilizándolo como cerrojo, se puede proteger el acceso a zonas donde ya esté trabajando otro hilo. Se puede complementar su funcionalidad con el uso de *wait()* y *notify()*.

En este caso, para poder depurar el código y observar las condiciones de carrera, no se puede usar la consola habitual y utilizar "*System.out.println*", ya que aunque se programe en Java, no se ejecuta ese código como tal, sino que se interpreta mediante la máquina de *Dalvik* para poder mostrarla en el emulador.

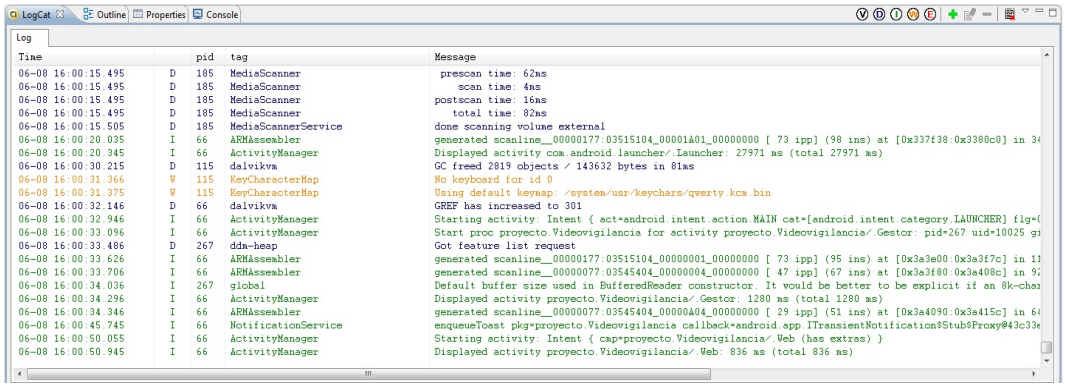


Figura 2.10: DDMS [3]

Para ello se usa la perspectiva "*DDMS*" con la herramienta "*LogCat*" en la que se monitorizan todos los estados por los que pasa Android. Usando la instrucción

Log.d(tag, String), se puede mostrar el estado de las variables que interesen y en el punto que interese. Como se puede ver en la figura 2.10 se muestran distintas variables ("tag") con su correspondiente valor ("Message"). De esta forma, se pueden definir en el código variables y obtener su valor sin más que utilizar esta perspectiva.

2.1.12. APIs opcionales

Android es válido para una gran cantidad de teléfonos, smartphones... El núcleo de las APIs de Android son válidas en cualquier teléfono, pero hay algunas que son opcionales. Son opcionales en el sentido de que el aparato no puede soportarlas completamente debido a que no tienen el *hardware* apropiado [3].

Wi-Fi APIs: proporcionan lo necesario para que las aplicaciones puedan comunicarse con el nivel inferior de la pila que proporciona el acceso a la red Wi-Fi. La mayor parte de la información del dispositivo está disponible, velocidad de conexión, dirección IP, estado, etc.

Servicios basados en localización: *software* que permite obtener al teléfono su posición. Esto incluye localización mediante satélites GPS, pero no se limita a ello. Otros sistemas de localización que aparezcan en el futuro, también se podrán utilizar.

Media APIs: se usa para reproducir ficheros multimedia. Esto incluye video y audio, *streaming* de datos sobre la red. Técnicamente no son opcionales, porque siempre están presentes, pero puede haber diferencia en los *codecs* dependiendo del dispositivo.

Gráficos 3D con OpenGL: el interfaz de usuario primario es una jerarquía de clases típica. Es el API usado para acceder a las capacidades 3D. No es estrictamente opcional ya que siempre estará presente. Pero puede ser que algunos dispositivos no tengan aceleración de *hardware* y usen *software rendering*, lo que afectará al funcionamiento de la aplicación.

2.2. RTSP

Se denomina transmisión en tiempo real a la comunicación en la que el receptor puede manipular los datos al mismo tiempo que el emisor los está enviando. Para ello se necesita un sistema que proporcione a las aplicaciones los servicios y el control necesario para la gestión de la calidad de servicio (QoS) [12].

Los protocolos de transmisión tienen sistemas de control de errores y de reenvío de paquetes que aseguran que la fiabilidad entre emisor y receptor sea transparente a los niveles superiores [12].

Para la transmisión en tiempo real, esta gestión de errores puede ser negativa, debido al retraso que produciría la retransmisión de un paquete de nuevo. Para evitar este problema se plantea que el tratamiento y gestión de los errores sea a niveles superiores. En una red en tiempo real, la pérdida de paquetes puede ocurrir debido a la saturación de memoria en los nodos o al superar el retraso máximo exigido [12].

Las prestaciones de una transmisión multimedia pueden ser medidas en dos dimensiones: latencia y fidelidad. La latencia puede ser vital para aplicaciones interactivas como conferencias mientras que para la transmisión de una película no lo es. La fidelidad de la transmisión es variable. Hay aplicaciones que no toleran ninguna variación en la fidelidad de la imagen como podría ser la transmisión de imágenes médicas y otras en que esta variación sólo produce una cierta distorsión tolerable como la transmisión de películas o música [12].

La provisión de servicios multimedia sobre redes en tiempo real se denomina "Streaming". La tecnología de *streaming* se utiliza para aligerar la descarga y ejecución de audio y vídeo en la web, ya que permite escuchar y visualizar los archivos mientras se están descargando [12].

La diferencia con los servicios de descarga es clara, ya que en este se produce primero el envío de datos y sólo después se accede a los contenidos, por el contrario, el streaming permite ver o escuchar el contenido durante la descarga; es decir, el transporte y el tratamiento de datos de producen en forma simultánea [12].

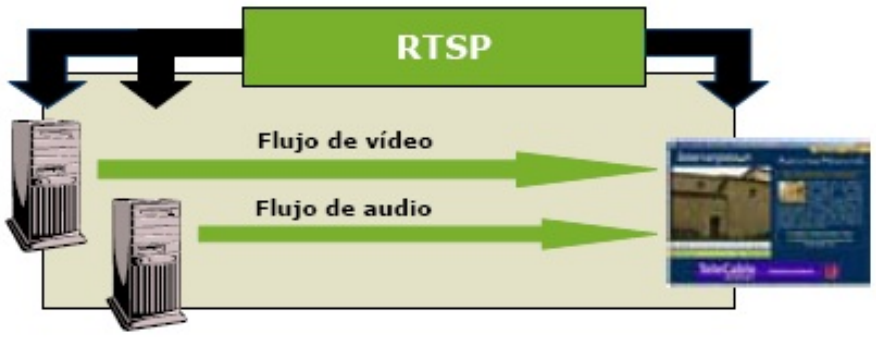


Figura 2.11: Ejemplo de streaming con RTSP [16]

RTSP (Real Time Streaming Protocol) es un protocolo no orientado a conexión. El servidor mantiene una sesión asociada a un identificador, en la mayoría de los casos RTSP usa TCP para datos de control del reproductor y UDP para los datos de audio y vídeo aunque también puede usar TCP en caso de que sea necesario (en la figura 2.12 se muestra el lugar de RTSP en la pila de protocolos). En el transcurso de una sesión RTSP, figura 2.13, un cliente puede abrir y cerrar varías conexiones de transporte hacia el servidor por tal de satisfacer las necesidades del protocolo [10].

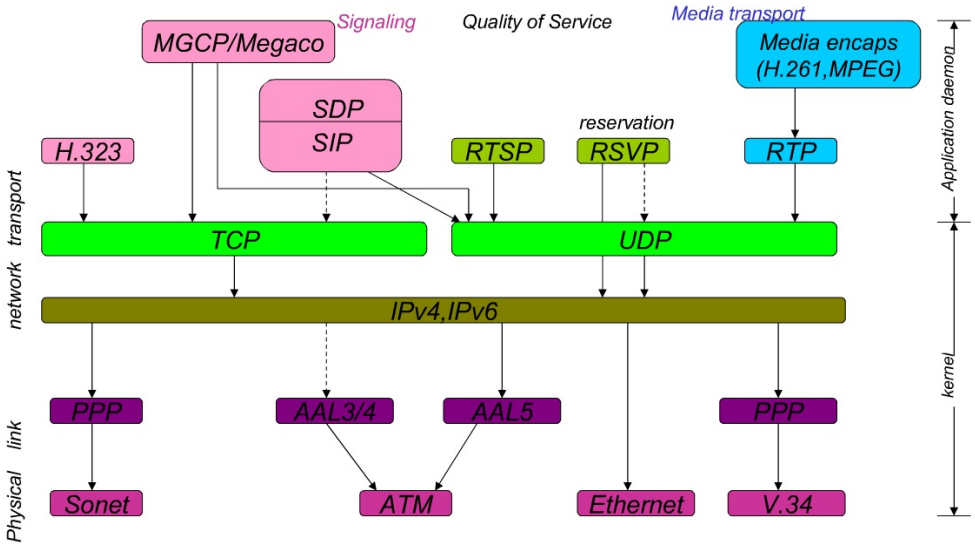


Figura 2.12: Pila de protocolos [17]

De forma intencionada, el protocolo es similar en sintaxis y operación a HTTP de forma que los mecanismos de expansión añadidos a HTTP pueden, en muchos casos, añadirse a RTSP. Sin embargo, RTSP difiere de HTTP en un número significativo de aspectos [11]:

- RTSP introduce nuevos métodos y tiene un identificador de protocolo diferente.
- Un servidor RTSP necesita mantener el estado de la conexión, al contrario de HTTP.
- Tanto el servidor como el cliente pueden lanzar peticiones.
- Los datos son transportados por un protocolo diferente.
- RTSP es definido utilizando la norma ISO 10646 (UTF-8) en lugar de la norma ISO 8859-1.

El protocolo soporta las siguientes operaciones [11]:

- Recuperar contenidos multimedia del servidor: el cliente puede solicitar la descripción de una presentación por HTTP o cualquier otro método. Si la presentación es multicast, la descripción contiene los puertos y las direcciones que serán usados. Si la presentación es unicast el cliente es el que proporciona el destino por motivos de seguridad.
- Invitación de un servidor multimedia a una conferencia: Un servidor puede ser invitado a unirse a una conferencia existente en lugar de reproducir la presentación o grabar todo o una parte del contenido. Este modo es útil para aplicaciones de enseñanza distribuida dónde diferentes partes de la conferencia van tomando parte en la discusión.

- Adición multimedia a una presentación existente: Particularmente para presentaciones en vivo, útil si el servidor puede avisar al cliente sobre los nuevos contenidos disponibles.

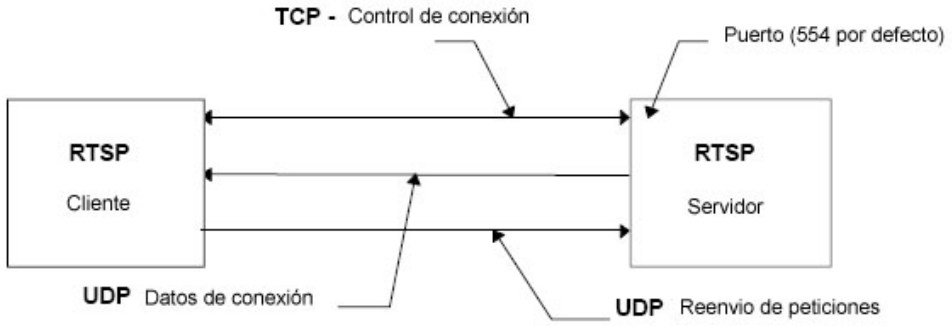


Figura 2.13: RTSP UDP [10]

2.2.1. Propiedades

RTSP tiene las siguientes propiedades [11]:

- Extensible: nuevos métodos y parámetros pueden ser fácilmente añadidos al RTSP.

- Seguro: RTSP reutiliza mecanismos de seguridad web ya sea a los protocolos de transporte (TLS) o dentro del mismo protocolo. Todas las formas de autentificación HTTP ya sea básica o basada en resumen son directamente aplicables.

- Independiente del protocolo de transporte: RTSP puede usar indistintamente protocolos de datagrama no fiables (UDP) o datagramas fiables RDP (Radio Data Packet") o un protocolo fiable orientado a conexión como el TCP.

- Capacidad multi-servidor: Cada flujo multimedia dentro de una presentación puede residir en servidores diferentes, el cliente automáticamente establece varías sesiones concurrentes de control con los diferentes servidores, la sincronización la lleva a término la capa de transporte.

- Control de dispositivos de grabación: El protocolo puede controlar dispositivos de grabación y reproducción (p.ej cámaras IP RTSP).

- Adecuado para aplicaciones profesionales: RTSP soporta resolución a nivel de frame mediante marcas temporales SMPTE para permitir edición digital.

- Separación del control de flujo y la iniciación de conferencias: el control de flujo está separado de invitar a un servidor de medios para una conferencia. El único requisito es que el protocolo de inicio de conferencia proporciona o se puede utilizar para crear un identificador único.

- Negociación de transporte: el cliente puede negociar el método de transporte antes de procesar un flujo de datos.

2.2.2. Peticiones RTSP

Las peticiones RTSP están basadas en peticiones HTTP y generalmente son enviadas del cliente al servidor. Su sintaxis y operación es similar a HTTP con lo que los mecanismos de expansión añadidos a HTTP pueden añadirse también en muchos casos a RTSP. A continuación se describe el intercambio de peticiones y respuestas más típicos [10] [11]:

- **PROPIEDADES**

  Este método obtiene una descripción del objeto multimedia apuntado por una URL RTSP (DESCRIBE) situada en un servidor. El servidor responde a esta petición con una descripción del recurso solicitado (Content-Type, Content-Length, etc), entre otros datos la descripción contiene una lista de los flujos multimedia que serán necesarios para la reproducción. Esta solicitud/respuesta constituye la fase de inicialización del RTSP.

  Ejemplo:

  -Cliente a Servidor:

  -DESCRIBE rtsp://unservidor.com/uncontingut RTSP/1.0
  -Accept: application/sdp, application/rtsl, application/ mheg

  -Servidor a Cliente:

  -RTSP/1.0 200 OK
  -Content-Type: application/sdp
  -Content-Length: 376
  -i=Descripción del contenido

-m=audio 3456 RTP/AVP 0
-m=video 2232 RTP/AVP 31

- **SETUP**

  Esta petición contiene la URL del flujo multimedia y especifica como se transportará el flujo de datos, así como un puerto para recibir los datos (audio o vídeo) y otro para los datos RTCP (meta-datos).

  El servidor responde a esta petición confirmando los parámetros escogidos y completa las partes restantes, como los puertos escogidos. Cada flujo de datos debe ser configurado con un SETUP antes de enviar una petición de PLAY.

  Ejemplo:

  -Cliente a Servidor:

  -SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
  -Transport: RTP/AVP;unicast;client_port=4588-4589

  -Servidor a Cliente:

  -RTSP/1.0 200 OK
  -Session: 47112344
  -Transport: RTP/AVP;unicast;
  -client_port=4588-4589;server_port=6256-6257

- **PLAY**

  Una petición de PLAY provocará que el servidor comience a enviar datos de los flujos especificados utilizando los puertos configurados con SETUP.

  Ejemplo:

  -Cliente a Servidor:

  -PLAY rtsp://unservidor.com/audio RTSP/1.0
  -Session: 12345678

- **PAUSE**

  Detiene temporalmente uno o todos los flujos, de manera que puedan ser recuperados con un PLAY posteriormente.

  Ejemplo:

  -Cliente a Servidor:

  -PAUSE rtsp://unservidor.com/video1 RTSP/1.0
  -Session: 12345678

  -Servidor a Cliente:

  -RTSP/1.0 200 OK

- **TEARDOWN**

  Detiene la entrega de datos para la URL indicada liberando los recursos asociados.

  Ejemplo:

  -Cliente a Servidor:

  -TEARDOWN rtsp://example.com/fizzle/foo RTSP/1.0
  -Session: 12345678

  -Servidor a Cliente:

  -RTSP/1.0 200 OK

En la siguiente gráfica, 2.14, se muestra el diagrama de estados del protocolo según la operación que se realiza.

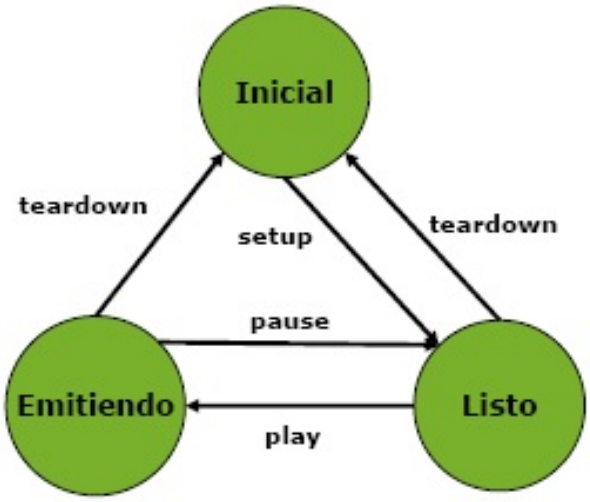


Figura 2.14: Diagrama de estados de RTSP [16]

2.2.3. Sesión RTSP:

Para establecer unas conexión RTSP, se realizan los siguientes pasos [10]:

1. El cliente accede a la URL RTSP para colocar el nombre del servidor y el puerto.
2. Si el nombre del servidor no está en formato IP, el cliente hace una consulta DNS para obtener la dirección correspondiente.
3. El cliente inicia una conexión TCP hacia el servidor.
4. Cuando la conexión está establecida correctamente, el cliente envía al servidor una petición OPTIONS. EL servidor devuelve información que puede incluir la versión de RTSP, la fecha, el número de sesión, el nombre del servidor y los métodos soportados.
5. El cliente envía una petición DESCRIBE para obtener una descripción de la presentación. El servidor responde con todos los valores de inicialización necesarios para la presentación.
6. El cliente envía SETUP para cada flujo de datos que se quiere reproducir. El SETUP especifica los protocolos aceptados para el transporte de los datos.
7. El cliente inicializa los programas adecuados requeridos para reproducir la presentación.
8. El cliente envía una petición PLAY que informa al servidor que ahora es el momento de comenzar a enviar datos.
9. Durante la sesión, el cliente periódicamente hace ping al servidor utilizando peticiones SET_PARAMETER. Aunque la respuesta sea errónea el cliente la ignora informando al cliente que el servidor todavía está activo.

10. Cuando la presentación termina o el usuario la para, el cliente envía un SET_PARAMETER que contiene las estadísticas de la sesión.

11. El cliente envía TEARDOWN para dar por terminada la conexión con el servidor.

2.3. Motion

Motion es un programa que monitoriza la señal de vídeo de una o más cámaras y es capaz de detectar si ha cambiado una parte significativa de la imagen, es decir, se puede detectar el movimiento. Toda la documentación se ha obtenido de [8].

El programa está escrito en C y funciona bajo el sistema operativo Linux. Motion es una herramienta basada en línea de comandos cuya salida puede ser jpeg, mpeg o secuencias de vídeo. No tiene absolutamente ninguna interfaz gráfica de usuario. Todo se configura a través de la línea de comandos o a través de un conjunto de archivos de configuración (simples archivos ASCII).

2.3.1. Características generales

Además de analizar las imágenes para detectar movimiento, Motion proporciona funcionalidades adicionales, siendo las siguientes las más representativas:

- Captar imágenes de movimiento
- Visualizar múltiples dispositivos de vídeo al mismo tiempo
- Visualizar múltiples entradas de una tarjeta de captura, al mismo tiempo
- Streaming de Webcam en vivo (usando multipart / x-mixed-replace)
- Creación en tiempo real de vídeos MPEG usando las bibliotecas de ffmpeg
- Toma instantánea de imágenes a intervalos regulares
- Captura instantánea de imágenes a intervalos irregulares utilizando cron
- Ejecutar comandos externos cuando se detecta movimiento (y por ejemplo, enviar SMS o correo electrónico)
- Seguimiento de movimiento (la cámara sigue el movimiento - hardware especial requerido)
- Fuente eventos en un MySQL o base de datos PostgreSQL.
- Video Comunicación a un bucle de tiempo real video4linux visualización

- Muchos usuarios han contribuido con proyectos relacionados con interfaces web, etc
- Usuario configurable y definido por el usuario en pantalla.
- Control a través del navegador (las versiones anteriores utilizan XML-RPC)
- Control automática de ruido y del umbral
- Motion es un demonio con un bajo consumo de CPU de memoria.

2.3.2. ¿Quién hace Motion?

Motion es un proyecto de código abierto. No cuesta nada. Motion se publica bajo licencia GNU General Public License (GPL) versión 2 o posterior. Esto significa que se puede conseguir el programa, instalarlo y usarlo libremente.

2.3.3. Notas sobre el funcionamiento

Si se tiene más de una cámara no se debe ejecutar el programa más de una vez. Motion trabaja con más de una cámara creando una serie de hilos, siendo cada uno de ellos un archivo de configuración. Si sólo tiene una cámara, sólo se necesitará el archivo motion.conf. En el momento en que haya dos o más cámaras, se debe tener un archivo de configuración por cada cámara, además del archivo `motion.conf`. Así, si por ejemplo se tienen dos cámaras, se necesitarán el archivo `motion.conf` hilo y dos archivos más de configuración. En total tres archivos.

Las opciones comunes a todas las cámaras se pueden definir en el archivo `motion.conf` y las particulares de cada una de ellas en el archivo correspondiente a su hilo. Otra opción menos elegante consistiría en definir en el archivo perteneciente a cada hilo todos los parámetros de configuración.

Así, el funcionamiento anteriormente descrito se podría resumir siguiendo los siguientes cuatro pasos.

1. Motion lee el archivo de configuración `motion.conf` desde el principio del archivo bajando línea por línea.

2. Si la opción de "hilo" se define en `motion.conf`, el archivo de configuración de hilo (s) es / (son) leídos.

3. Motion continúa leyendo el resto del archivo `motion.conf`. Cualquiera de las opciones definidas de aquí en adelante sobrescribirán las definidas previamente en los ficheros de configuración de cada hilo.

4. Motion vuelve a leer la línea de comandos, sobrescribiendo las opciones definidas previamente.

Siempre deben llamarse a los ficheros de configuración de hilo al final del archivo `motion.conf`. Si se definen opciones en `motion.conf` después de que se invoque a los distintos hilos, dichas opciones nunca serán utilizadas.

2.3.4. Cámaras de red

Motion puede conectarse a una cámara de red a través de un socket TCP normal. Todo lo que se necesita es la dirección. La dirección URL dada debe devolver una sola imagen jpeg o un stream mjpeg. Por el momento Motion no se puede conectar a una secuencia de vídeo tal MPEG, MPEG4, DivX. Puede conectarse a través de un servidor proxy. También hay que tener en cuenta que no utilice una dirección URL que cree una página HTML con un jpg incrustado. Lo que debe ser devuelto es la imagen jpg propia o en el flujo mjpeg.

2.3.5. Control remoto vía http

Motion puede ser controlado de forma remota mediante una interfaz http simple. El servidor web responde con algunas cabeceras http simple seguido de una página web codificada en HTML.

La mayoría de opciones de configuración de Motion se pueden cambiar mientras se está ejecutando, excepto las opciones relacionadas con el tamaño de las imágenes capturadas y los archivos de máscara que sólo se cargan cuando se inicia el programa.

Así que la herramienta más obvia para controlar de forma remota el programa es cualquier navegador web. Todos los comandos se envían utilizando el método HTTP GET. Puede utilizar cualquier navegador (Firefox, Mozilla, Internet Explorer, Konqueror, etc). También puede utilizar el navegador basado en texto "Lynx"para controlar el movimiento desde una consola.

2.4. Darwin Stream Server

Se denomina “Streaming” a la técnica utilizada para transmitir datos de tal modo que estos puedan ser procesados en nuestra computadora como un flujo continuo y constante, mostrando la información transmitida antes de que el archivo se haya descargado en su totalidad, aún cuando los archivos en teoría “no tengan fin” como una transmisión de televisión o de radio en vivo por ejemplo.

La tecnología de *streaming* se utiliza para aligerar la descarga y ejecución de audio y vídeo en la web, ya que permite escuchar y visualizar los archivos mientras se están descargando [19].

Si no utilizamos streaming, para mostrar un contenido multimedia en la Red, tenemos que descargar primero el archivo entero en nuestro ordenador y más tarde ejecutarlo, para finalmente ver y oír lo que el archivo contenía. Sin embargo, el streaming permite que esta tarea se realice de una manera más rápida y que podamos ver y escuchar su contenido durante la descarga [19].

El *streaming* funciona de la siguiente manera [19]:

- El cliente conecta con el servidor y éste comienza el envío del fichero.
- El cliente comienza la recepción del fichero, construyendo un buffer donde almacena la información.
- Cuando se ha alcanzado un pequeño porcentaje de llenado el buffer, el cliente comienza la reproducción del mismo, a la vez que continúa con la descarga.
- El sistema está sincronizado para que el archivo se pueda ver mientras se descarga, de modo que cuando el archivo acaba de descargarse el fichero también ha acabado de visualizarse.
- Si en algún momento la conexión sufre descensos de velocidad, se utiliza la información que hay en el buffer, de modo que se pueda soportar esa disminución.
- Si la comunicación se corta demasiado tiempo, el buffer se vacía y la reproducción del archivo se cortaría hasta que se restaurase la transmisión.

A la hora de seleccionar un servidor de streaming se han valorado distintas opciones. Las tres tecnologías de *streaming*, posiblemente más conocidas del momento, son [19]:

- Real Media es posiblemente la más popular. También es la empresa con más experiencia en el sector y desarrolla muchos productos orientados a la distribución de archivos multimedia.
- Windows Media es la apuesta de Microsoft. Ya posee una cuota de usuarios muy importante y seguramente aumentará con rapidez ya que Microsoft incluye el plug-in en la instalación típica de los sistemas operativos que está fabricando.
- Quick Time es la tercera opción. Con menor cuota de mercado. Su version para Linux es Darwin Stream Server (DSS).

Darwin Streaming Server (DSS) es un servidor que permite el envío de esta información, utilizando el formato QuickTime, a través de los protocolos RTP y RTSP, estándares en la industria para la transmisión de Streaming. Este formato

está basado en pistas, lo que permite combinar prácticamente cualquier contenido multimedia (audio, vídeo, imágenes fijas, texto, realidad virtual, capítulos e incluso idiomas alternativos) en una misma película. Dispone de más de 2.500 APIs multiplataforma totalmente documentadas que permiten a los desarrolladores de software aprovechar las posibilidades de QuickTime en aplicaciones multimedia, de creación y de entretenimiento [18].

Hemos elegido este servidor porque está basado en el mismo código que el Apple's QuickTime Streaming Server. Proporciona un alto nivel de personalización y se ejecuta en una variedad de plataformas que le permite manipular el código para que se ajuste a sus necesidades.

Si bien comparten la misma base de código como QuickTime Streaming Server, Darwin Streaming Server es un proyecto open source destinado a desarrolladores que necesitan flujo de QuickTime y MPEG-4 en los medios de comunicación alternativos, tales como las plataformas Windows, Linux y Solaris, o los desarrolladores que necesitan ampliar y / o modificar el código de servidor existente que se ajuste a sus necesidades.

El servidor de streaming DSS de vídeo y audio transmite el flujo de datos en respuesta a peticiones que usan el software cliente. Las solicitudes se manejan utilizando Real-Time Streaming Protocol (RTSP), un protocolo para el control flujo de contenidos multimedia en tiempo real. El streaming se realiza utilizando Real-time Transport Protocol (RTP), un protocolo de transporte utilizados para la transmisión en tiempo real de contenido multimedia a través de redes [9].

2.4.1. Streaming en directo vs Streaming bajo demanda

Las opciones servicio para los medios de streaming en tiempo real, se dividen en dos categorías: en directo y bajo demanda. Los eventos en vivo, como conciertos, discursos y conferencias, son comúnmente escuchados en Internet según se producen con la ayuda de software. Se codifica la fuente, como el vídeo de una cámara, en tiempo real, figura 2.16, y proporciona el flujo resultante en el servidor. El servidor sirve la transmisión en directo a los clientes [9].

Independientemente de cuán diferentes sean los clientes, cada uno ve el mismo punto al mismo tiempo. Esta experiencia en vivo se puede simular con el contenido grabado [9].

Para una experiencia de entrega bajo demanda, como una película, cada cliente empieza a recibir el flujo de datos desde el principio, por lo que nadie llega tarde al comienzo. El flujo comienza cuando el cliente lo ordena [9].

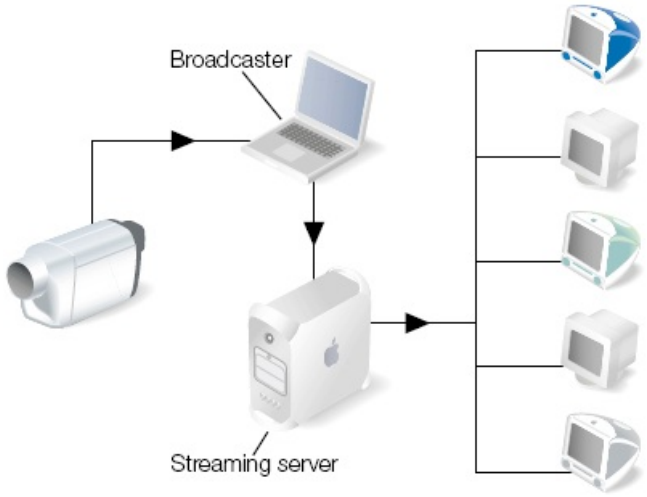


Figura 2.15: Live streaming [9]

2.4.2. ¿Cómo funciona el Streaming?

Al ver y escuchar las transmisiones en la televisión o los medios de comunicación ya sea por radio, cable... estas transmisiones son en su mayoría sin comprimir y consumen grandes cantidades de ancho de banda. Pero eso no es un problema, porque no tienen que competir con otras transmisiones en la frecuencia en la que están emitiendo [9].

Cuando se envía el mismo contenido a través de Internet, el ancho de banda utilizado ya no es exclusivo para eso. Los medios de comunicación ahora tiene que compartir el ancho de banda con miles, posiblemente millones de personas, de otras transmisiones a través de Internet [9].

Por lo tanto, los contenido multimedia enviados a través de Internet son codificados y comprimidos para transmisión. Los archivos resultantes se guardan en un lugar específico, y los softwares de streaming como Darwin Streaming Server se utilizan para enviar los distintos contenidos [9].

Los streamings los pueden ver tanto por usuarios de distintas plataformas usando QuickTime Player (disponible gratuitamente en el sitio web de Apple) o cualquier otra aplicación compatible con QuickTime o archivos estándar MPEG-4. También se puede configurar para que los usuarios puedan verlos desde un navegador web con el *plugin* QuickTime está instalado [9].

Cuando un usuario comienza a reproducir contenido multimedia a través de una página web, el *plugin* QuickTime envía una solicitud al servidor de streaming. El servidor responde enviando el contenido multimedia al equipo cliente [9].

El tipo de contenido multimedia que se envía al equipo cliente depende de lo que se haya especificado en la página web. Si alguien se conecta a una lista de reproducción creada por el servidor de streaming, eso es lo que se envía. Si se conecta a una película QuickTime en el directorio de los contenidos, eso es lo que se envía. Si se conecta a una transmisión en vivo, eso es lo que se envía [9].

2.4.3. Multicast vs Unicast

DSS soporta multicast y unicast para ofrecer media streaming.

En multicast, figura 2.17, un solo flujo de datos se comparte entre todos los clientes (ver figura). Cada cliente "sintoniza" un flujo como cuando se sintoniza una emisora de radio. Aunque esta técnica reduce la congestión de la red, se requiere una red en la que cualquiera tenga acceso a la red troncal de multicast, también llamada "Mbone", de los contenidos distribuidos a través de Internet, o de multicast habilitado para la distribución en una red privada [9].

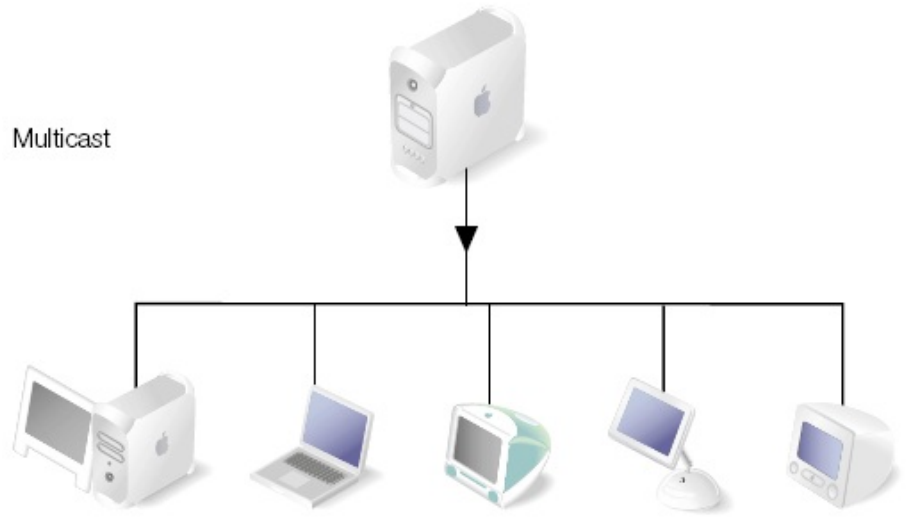


Figura 2.16: Multicast [9]

En un unicast, figura 2.18, cada cliente inicia su propio flujo, dando como resultado la generación de muchas conexiones entre el cliente y el servidor (ver figura). Muchos clientes conectados en unicast a un flujo en una red local puede dar lugar a tráfico de red. Pero esta técnica es más fiable para la entrega a través de Internet, porque no necesita ningún tipo de apoyo al transporte [9].

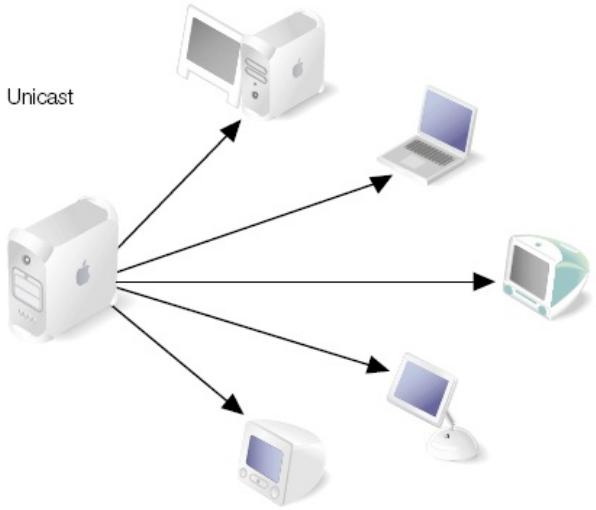


Figura 2.17: Unicast [9]

2.4.4. Streaming de contenidos multimedia

Darwin Streaming Server puede ser configurado para escuchar un flujo entrante y luego enviar ese flujo a uno o más destinos (ver figura 2.19). Puede reducir el consumo de ancho de banda de Internet. Puede ser útiles en casos especiales, sobre todo si los espectadores en diferentes lugares deseas optimizar sintonizar un mismo flujo [9].

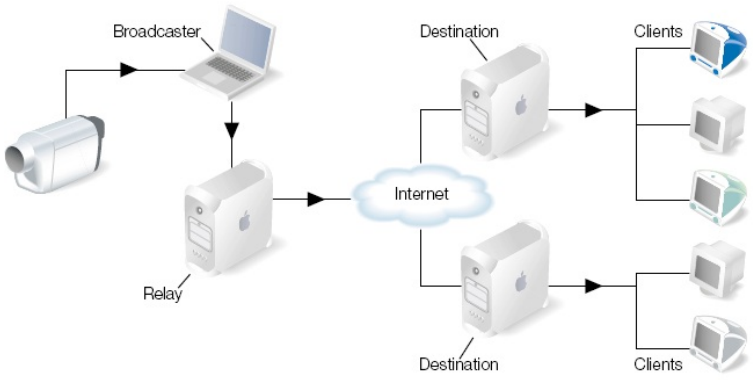


Figura 2.18: Retransmisión en vivo [9]

2.5. Conclusiones

A la vista del estado del arte, se puede concluir que el uso de Android como plataforma para el desarrollo de una aplicación de vídeo vigilancia, proporciona una gran cantidad de herramientas para que se pueda desarrollar el programa, de forma que se adapte lo más posible a las necesidades del proyecto.

Por otra parte, Motion, da muchísimas opciones de personalización en el tratamiento y gestión de las imágenes, siendo muy útil, además de proporcionar una amplia variedad de utilidades que harán que se pueda mejorar la aplicación en el futuro, dotándola de nuevas funcionalidades.

Por último, “Darwin Stream Server", realiza el streaming del vídeo grabado utilizando el protocolo RTSP, el cual es la única forma que soporta Android para realiza dicha operación.

Capítulo 3

Desarrollo del sistema

En este capítulo se va a presentar todo lo relativo al proceso de diseño y desarrollo del sistema, entendiendo como tal, el proceso de concepción del mismo, así como el establecimiento de los requisitos, los posibles casos de usos y los diagramas de flujo y de clases de la aplicación. También se discutirán las distintas decisiones de implementación, tales como el lenguaje de programación y el entorno de desarrollo.

3.1. Diseño del sistema

La televigilancia permite mediante un dispositivo (dispositivo móvil con Android en este caso), visualizar cámaras desde cualquier lugar, sólo con disponer de acceso a Internet, ya se vía WiFi o con tarifa de datos, para visualizar entornos diversos como empresas, comercios, hogares, etc... proporcionando además acceso para recuperar imágenes grabadas o en tiempo real.

El envío las imágenes captadas por las cámaras IP, través de la red, lo realizan ellas mismas o mediante servidores web de vídeo. Toda la información de vídeo que estén captando en ese momento, se analizan con "Motion"y se almacenan en un disco duro para poder se recuperadas para el historial. Gracias a ello podemos ver de forma remota, desde cualquier lugar, la zona a vigilar, grabar de imágenes, simplificar la supervisión de personal, supervisión de espacios, abiertos y cerrados...

Con este proyecto se pretende ofrecer un sistema sencillo y básico, compuesto por las cámaras IP especificadas anteriormente, un PC que hace las veces de servidor y dispositivos con el Sistema Operativo Android. A pesar de ser un sistema muy concreto, lo que puede hacer caer en el error de pensar que no se trata de una aplicación que pueda llegar a un gran número de usuarios, se trata de una aplicación totalmente abierta soportada por una tecnología emergente que pretende hacerse fuerte en el mercado de la tecnología portátil.

3.1.1. Requisitos

A continuación se muestran los requisitos de la aplicación, obtenidos teniendo en cuenta las necesidades del proyecto y de las limitaciones, hardware y software, que planteaban Android, Motion y DSS.

1. Es necesario tener acceso a una red de datos, preferiblemente Wifi.

2. Deberá haber suficientes números de acceso para que la cobertura de la zona a cubrir sea la adecuada.

3. El visionado de las cámaras no puede ser con *applets* ya que la JVM que lleva incorporada Android es parcial y no los soporta.

4. En la URL de la cámara sólo deben aparecer las imágenes del *streaming* para que "Motion" pueda realizar el tratamiento de las mismas.

5. Las direcciones de las cámaras y los servidores deben ser introducidas por los usuarios cuando se realice la configuración del sistema.

6. Los ficheros de vídeo del historial serán de un minuto para evitar que tarden tiempo en cargarse, ya que el tiempo en estas aplicaciones es un factor crítico.

7. Se guardarán ficheros de vídeo grabados como mucho una hora atrás. Aún así, esto dependerá de la capacidad de almacenamiento del sistema.

8. El servidor de vídeo debe hacer el streaming de los ficheros mediante el protocolo RTSP.

9. Versión mínima del sistema operativo Android: 1.5

3.1.2. Arquitectura servidor

En esta sección se muestra la arquitectura del sistema en la figura 3.1, haciendo énfasis en la parte del servidor. En él, se puede ver como la cámara realiza el *streaming* de las imágenes, las cuales, aparte de recibirlas el dispositivo móvil, también son captadas por Motion. Éste, las analiza y graba, en vídeos de sesenta segundos, y posteriormente, las deja accesibles para que el servidor de *streaming*

DSS pueda acceder a ellas, y enviarlas cuando llegue la petición correspondiente.

Figura 3.1: Arquitectura del sistema

En la otra parte del sistema se encuentra el dispositivo móvil, que recibe las imágenes de la cámara cuando solicita la visualización en directo, o las imágenes del servidor DSS cuando pide ver el historial almacenado.

3.1.3. Funciones del programa

A continuación se detallan las características que han de tener los distintos elementos de la aplicación:

- Cámaras IP

  Hemos de tener un fichero de configuración en el que se detallen las direcciones IP de cada una de las cámaras para poder acceder a ella. Así mismo, se tendrá la posibilidad de editar el fichero, añadiendo y borrando las direcciones que se estime conveniente. Aquí, tendremos la opción, mediante una lista, de escoger la cámara que queremos visualizar en directo o de la cual queremos acceder a su historial.

- Servidores

  También hemos de tener un fichero de configuración en el que se detallen las direcciones IP de cada una de los servidores para poder acceder a los videos de historial grabados por cada una de las cámaras. Asimismo, se tendrá la posibilidad de editar el fichero, añadiendo y borrando las direcciones que se estime conveniente. Aquí, tendremos la opción, mediante una lista, de escoger

el servidor que queremos utilizar.

- Web

  El programa podrá visualizar en una ventana cualquier cámara IP seleccionada. Podrá hacer zoom (tanto alejar como acercar), además de pausar y reanudar la visualización de las imágenes.

- Historial

  El programa, después de seleccionar una cámara, un minuto de los últimos 60 y un servidor, podrá visualizar mediante streaming un vídeo grabado y almacenado en un servidor. Para ello ser hace una petición RTSP, y mediante un reproductor podemos ver el vídeo seleccionado.

- Características principales

  La aplicación debe tener un alto grado de manejabilidad, es decir, debe que ser fácil de usar por cualquier tipo de usuario. Tiene que ser intuitiva de forma que pueda ser manejada con rapidez y agilidad. Además debe de ser fiable y de evitar los errores.

  El usuario será capaz de dar de alta y eliminar, tanto cámaras como servidores, además de la visualización en directo o de un historial grabado servido por streaming.

- Limitaciones

  El sistema que se trata de construir ha de funcionar en cualquier dispositivo que use como sistema operativo Android y con el software que se proporciona no se necesitará ni hardware ni software adicional.

  Además, el dispositivo tendrá que disponer de una tarjeta de red capaz de detectar redes Wi-Fi, o tener una tarifa de datos, de forma que pueda acceder a internet con el fin de visualizar las cámaras o los videos del historial.

  Para un funcionamiento adecuado será necesario hacer un estudio del tiempo que tarda en servirse el streaming, para así poder realizar una codificación del video en su grabación, que sea de una calidad suficiente y que no sea demasiado pesado para que se pueda cargarse en el menor tiempo posible.

3.1.4. Funcionamiento de la aplicación

El funcionamiento del sistema se podría representar mediante el siguiente diagrama, figura 3.2

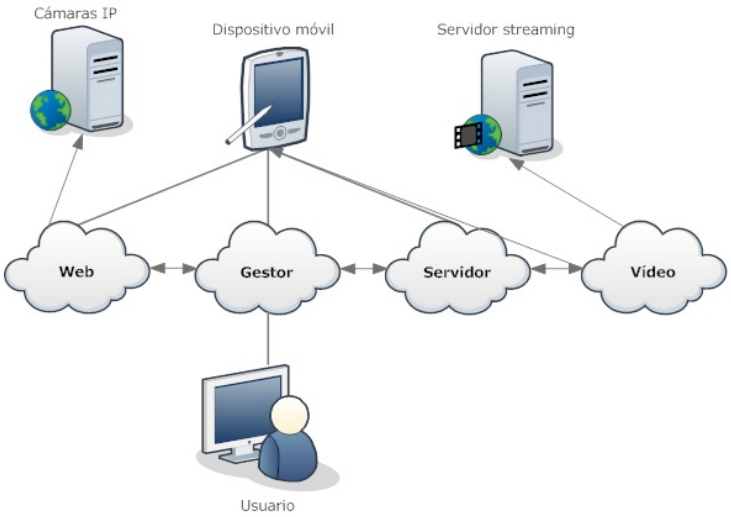


Figura 3.2: Diagrama de casos de uso

- Caso 1: el usuario inicia la aplicación y accede al menu "Gestor", desde donde puede realizar la gestión de las distintas cámaras, así como proceder al visionado de las mismas o a las grabaciones de historial.
- Caso 2: el usuario accede al menú de "Servidores"desde donde realiza su gestión y elige uno para su uso.
- Caso 3: el usuario accede a la parte "Web"desde donde podrá ver las imágenes emitidas por las distintas cámaras en directo.
- Caso 4: el usuario podrá ver las grabaciones de las cámaras mediante un servicio de `streaming`.

3.1.5. Seguridad

Actualmente la seguridad que ofrece la aplicación son los *firewall* de los routers y de la red del departamento de Telemática, WLIT y la seguridad de la red Wi-Fi UC3M. El uso de la aplicación no supone un compromiso para la seguridad de estas redes puesto que no se están enviando datos desde el dispositivo a la red, simplemente se está obteniendo información de la misma.

Como líneas futuras de trabajo, se puede llevar un registro de las personas que acceden al sistema mediante un sistema de autenticación que se instalaría en la aplicación.

3.1.6. Planteamiento del problema

El problema principal a resolver es el establecimiento de un circuito de vigilancia de un entorno determinado. Para ello necesitamos las cámaras de seguridad así como

dispositivos con Android, que serán con los que podamos visualizar dichas cámaras. Además, necesitaremos un servidor de vídeo, en el que se almacene el historial de grabación y nos lo sirva cuando lo requiramos.

Las ventajas de implementar el sistema de esta forma es el ahorro de costes de material, ya que toda la parte de pantallas y servidores nos la podemos ahorrar. Él único material hardware que necesitamos es: las cámaras de seguridad (indispensable en cualquier sistema), un PC que haga de servidor, así como los dispositivos móviles con Android.

3.2. Decisiones de implementación

A continuación se detallan decisiones relativas a la estructura de la aplicación, la interfaz gráfica y del tratamiento de los ficheros que contienen los datos de las direcciones de las cámaras y lo servidores.

1. Se ha decidido separar la funcionalidad de forma que tengamos los gestores tanto de las cámaras como de los servidores y asociados a cada uno de ellos, las opciones de visualizar las cámaras o el historial.

2. **Interfaz gráfica:** Aunque técnicamente es posible crear y adjuntar los *widgets* para nuestra actividad tan solo a través de Código Java, el enfoque más común es utilizar scripts basados en un diseño XML. La llamada dinámica de instancias de widgets se reserva para situaciones más complejas, donde los widgets no se conocen en tiempo de compilación.

   Como su nombre indica, un diseño basado en XML es una especificación de widgets, las relaciones de cada uno de ellos con el resto y con sus contenedores, todo esto codificado en formato XML.

   La mayoría de todo lo que se puede hacer en archivos de diseño de XML se puede lograr a través de Java. Por ejemplo, se podría utilizar setTypeface() para tener un botón que tuviese su texto en negrita, en lugar de utilizar una propiedad en un esquema XML.

   Quizás la razón más importante para usar una interfaz gráfica basada en XML es la ayuda que se obtiene gracias a la creación de herramientas para la definición de la vista de la aplicación. Una de estas herramientas es el constructor de GUI para el entorno de desarrollo de Eclipse. Otra herramienta que se puede encontrar en internet es DroidDraw. Si la información se encuentra estructurada y ordenada es más sencillo crear la interfaz gráfica en un fichero

XML, que en código Java.

Por otro lado, al estar separado la parte gráfica del código de Java facilita la labor a posibles desarrolladores secundarios, que trate de reinterpretar y modificar el código. Además, XML como un formato de definición de interfaz gráfica de usuario es cada vez más común. Como ejemplos tenemos XAML2 de Microsoft, Flex3 de Adobe, y XUL4 de Mozilla. Todos tratan de adoptar un enfoque similar al de Android: poner el diseño gráfico en un archivo XML.

3. **Ficheros:** En el caso de los ficheros de configuración de las direcciones IP tanto de las cámaras como de los servidores de vídeo, se ha optado por elegir un archivo de texto simple, sin formato. La elección de este archivo se debe a que de los editores de texto sus archivos son los de menor tamaño.

   Las direcciones se almacenarán de forma consecutiva, separadas por un retorno de carro.

   A la hora de guardar los archivos, Android ofrece la posibilidad de guardarlos en la propia aplicación, lo cual resulta muy útil para tener todo ordenado y almacenado en el mismo paquete del programa. Aún así se eligió por añadirlos en la tarjeta de memoria externa, en el directorio raíz. Con esto se consigue disponer de los archivos siempre que se necesiten y además, al estar en el directorio raíz, no será necesario crear nuevas carpetas.

   Se ha optado por un fichero de texto y no una base de datos por las siguientes razones. La primera y principal, es que un usuario que quisiese acceder a los datos tendría que tener conocimientos de programación para poder volcar el contenido de la base de datos en un fichero. La segunda razón es que la base de datos de Android, SQLite, está implementada en bibliotecas de C y habría que crear unas nuevas bibliotecas en Java. Esto supondría un aumento del tiempo de trabajo y la complejidad del proyecto sin ser un objetivo del proyecto la creación de bibliotecas para el manejo de la base de datos.

3.3. Implementación

En esta sección se analizan brevemente el lenguaje de programación usado para desarrollar en la plataforma Android, así como el entorno de desarrollo. Posteriormente, se muestran los diagramas de flujo de cada una de las partes de la aplicación y el diagram de clases de la misma.

3.3.1. Lenguaje de programación y entorno de desarrollo

El lenguaje de programación sobre el que se desarrolla en Android es Java. Aunque las aplicaciones se escriben en en este lenguaje, los archivos y librerías de bajo nivel están escritas en C/C++. Para programar en Android es necesario descargar el SDK proporcionado por Google, el cual contiene el conjunto de desarrollo y los paquetes necesarios para implementar aplicaciones para Android. Este paquete incluye las APIs y herramientas necesarias para desarrollar las aplicaciones utilizando Java como lenguaje de programación, el cual es el lenguaje utilizado en este proyecto fin de carrera.

3.3.2. El entorno de desarrollo

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse) [20].

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios [20].

Los widgets de Eclipse están implementados por una herramienta de widget para Java llamada SWT, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing. La interfaz de usuario de Eclipse también tiene una capa GUI intermedia llamada JFace, la cual simplifica la construcción de aplicaciones basada en SWT [20].

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos para proporcionar toda su funcionalidad al frente de la plataforma, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse [20].

En cuanto a las aplicaciones clientes, eclipse provee al programador con *fra-*

meworks para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros *plugins*, hacen uso de su interfaz gráfica personalizable y profesional [20].

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas de los archivos en tanto se refresque el espacio de trabajo correspondiente [20].

Eclipse permite extender su funcionalidad a través del *plugin* de Android, de forma que sólo hay que preocuparse por escribir el código, ya que él configura el proyecto de forma que se pueda compilar o ejecutar de forma similar a otros proyectos Java. Además del código propio de la aplicación que se realiza, se necesitan otras paquetes desarrollados por Google que se encuentran en el SDK de Android y permiten acceder a distintas funciones de la plataforma. Además del código propio de la aplicación, se necesitan otros tres archivos que están relacionados entre sí: un archivo `xml` para la interfaz gráfica, el `AndroidManifest.xml` y un archivo `R.java` que genera la aplicación automáticamente y sirve para referenciar cualquier recurso de los `xml` anteriores [20].

3.3.3. Diagramas de flujo

Este sistema de videovigilancia esta compuesto por dos partes bien diferenciadas: la parte hardware(cámaras IP, servidor vídeo y dispositivos Android), y la parte software (aplicación para la gestión de las cámaras y de la grabación).

A su vez, el software de la aplicación lo podemos dividir en 4 partes: la gestión de las cámaras (añadir y eliminar), gestión de los servidores (añadir y quitar), visualización de la cámara seleccionada y el visor de historial.

Gestor

Clase encargada de la gestión de las cámaras de seguridad (ver diagrama de flujo en la figura 3.3). En él se muestra una lista de cámaras que se obtiene de la lectura de un fichero de configuración.

El fichero de configuración puede estar cargado previamente en la tarjeta SD o se puede editar desde el mismo programa. Para ello existen en el botón menú las

opciones de añadir o borrar. Al añadir, aparecerá una ventana en la que habrá que añadir la dirección IP de la cámara. Esta se añadirá al final de la lista con un número correlativo al último registrado. Para borrar, bastará con indicar el número de la cámara que se desea eliminar.

A modo informativo, si se pulsa sobre una cámara, aparecerá un texto en el que se indica el número de cámara y la dirección IP de la misma: Cámara X: XXX.XXX.XXX.XXX

En el menú, aparte de las opciones de añadir y borrar mencionadas anteriormente, aparecer la opción "Video Server", la cual nos lleva al gestor de servidores.

Además de todo ello, se muestran dos botones en pantalla, cámara e historial, los cuales solo funcionan si previamente se ha seleccionado una cámara. Una vez seleccionada una, con "cámara" accedemos al visualizado de la misma, y si elegimos "historial", pasaremos a ver la grabación del minuto seleccionado.

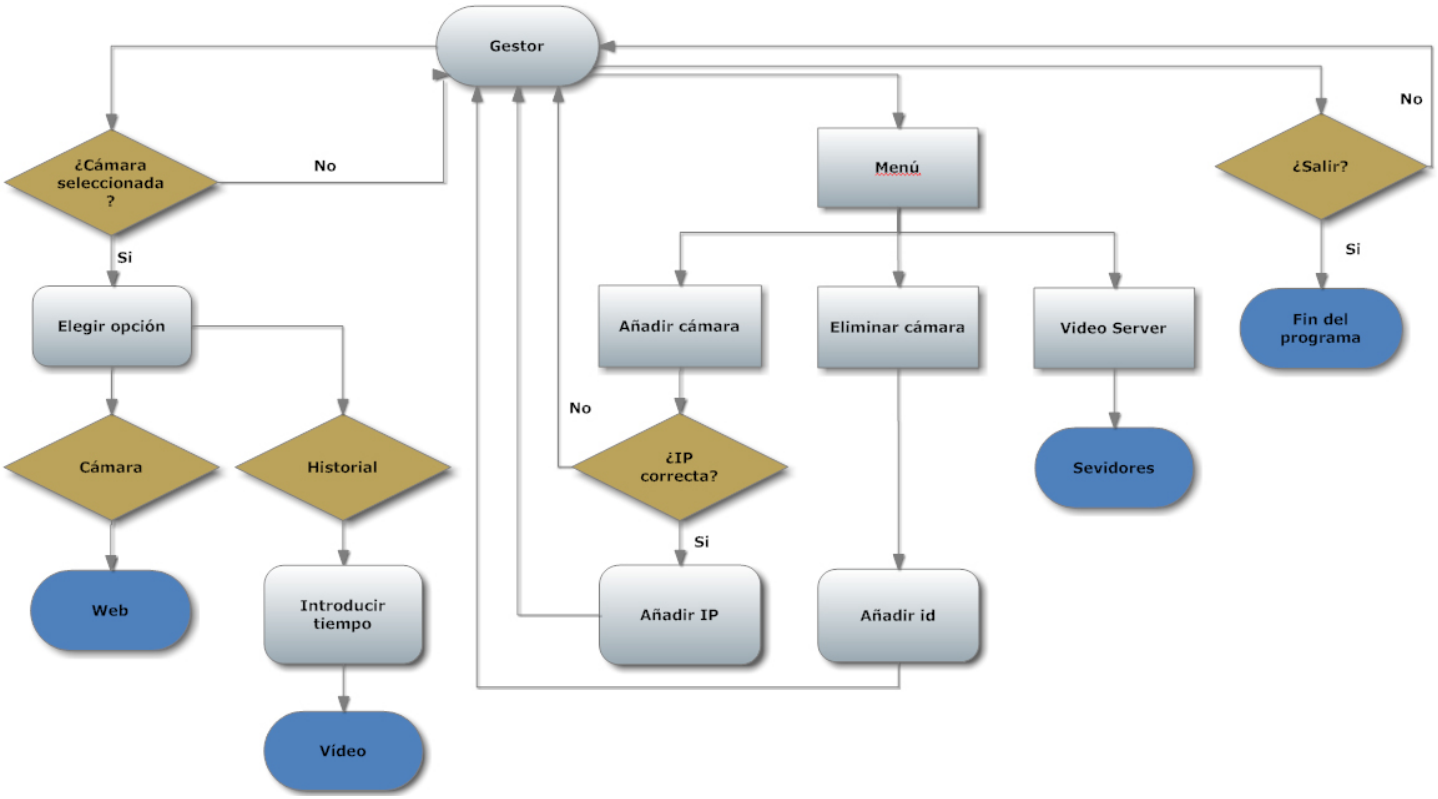


Figura 3.3: Diagrama de flujo de la clase Gestor

Servidor

Clase encargada de la gestión de los servidores de vídeo (ver diagrama de flujo en la figura 3.4). En él se muestra una lista de direcciones IP de cada uno de los mismo, la cual se obtiene de la lectura de un fichero de configuración.

El fichero de configuración puede estar cargado previamente en la tarjeta SD o se puede editar desde el mismo programa. Para ello existen en el botón menú las opciones de añadir o borrar. Al añadir, aparecerá una ventana en la que habrá que añadir la dirección IP del servidor. Esta se añadirá al final de la lista. Para borrar, bastará con indicar el número del servidor que se desea eliminar.

A modo informativo, si se pulsa sobre un servidor, aparecerá un texto en el que se indica la dirección IP del mismo.

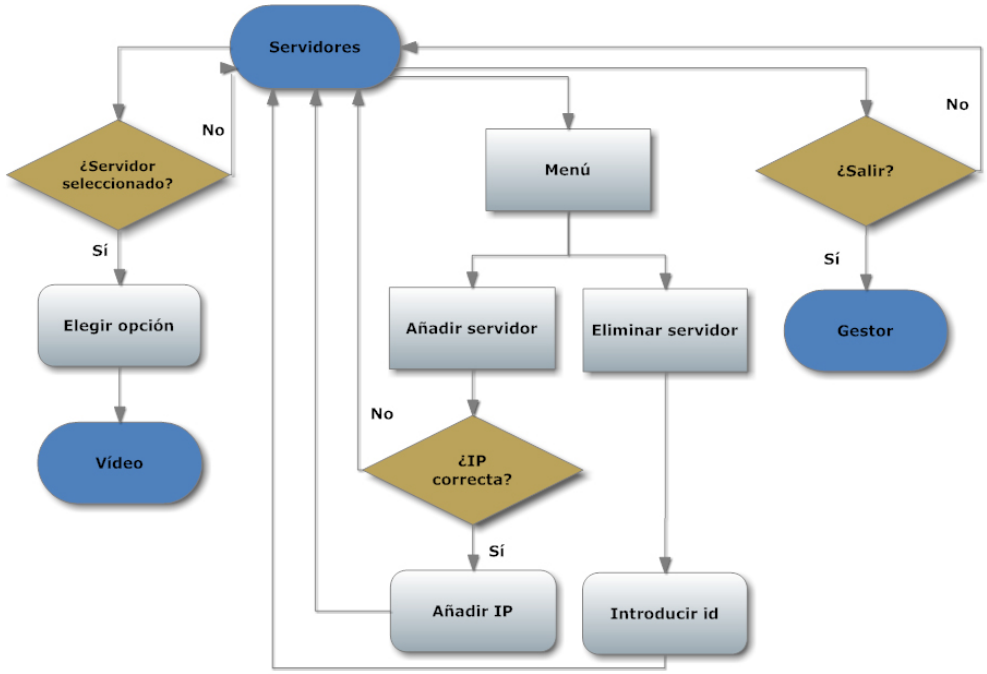


Figura 3.4: Diagrama de flujo de la clase Servidor

Web

Clase que nos permite el visualizado de las cámaras en directo (ver diagrama de flujo en la figura 3.5). Aquí se accede desde el menú Gestor, después de haber seleccionado una cámara. Existen las posibilidades de hacer zoom digital, moviendo el pad a izquierda o derecha además de pausar o reanudar el visionado sin más que presionar en el botón correspondiente.

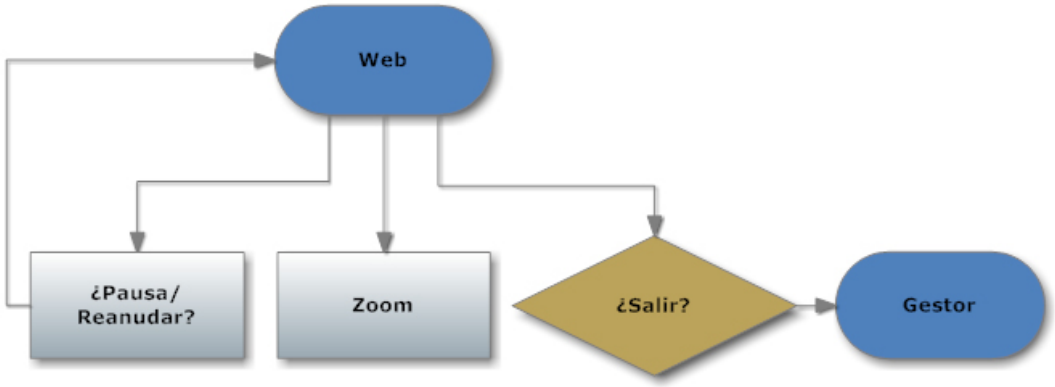


Figura 3.5: Diagrama de flujo de la clase Web

Vídeo

Clase que permite el visionado en streaming de un vídeo de un minutos sobre los últimos sesenta (ver diagrama de flujo en la figura 3.6). Para acceder a esta parte de la aplicación, primero hay que seleccionar una cámara, un minuto del historial y elegir un servidor en los que pueda estar alojado el vídeo.

Una vez comenzada la reproducción se podrá pausar o reanudar con los botones correspondientes, además de pasar al siguiente minuto o al anterior con los botones dispuestos a tal efecto.

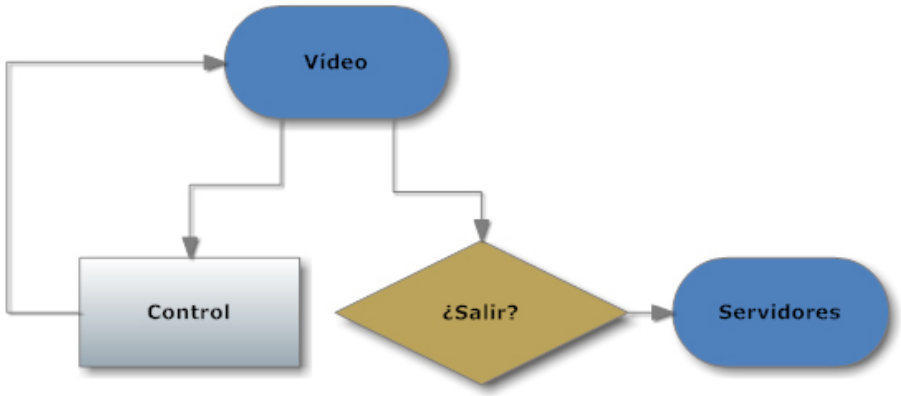


Figura 3.6: Diagrama de flujo de la clase Vídeo

Ficheros

Clase dispuesta para el tratamiento de los ficheros de configuración de las cámaras y los distintos servidores. Permite tanto añadir como borrar, así como buscar y modificar.

3.3.4. Diagramas de clases

Como se puede ver en el diagrama, figura 3.7, existen cuatro actividades en esta aplicación, por lo que tenemos cuatro pantallas distintas de funcionamiento.

La primera pantalla está desarrollada a través de la clase "Gestor" que extiende de la interfaz `Activity`. En esta actividad se muestra la lista de cámaras del sistema. Con ella podemos dar de alta y de baja las cámaras, y seleccionar la opción que deseamos, es decir, ver una cámara en directo o el historial de alguna de ellas. Para realizar estas tareas de altas y bajas de los dispositivos, utilizamos la clase "Ficheros" que es la que nos permite realizar la lectura del fichero de configuración para cargarlas así como realizar las distintas modificaciones. Además, contamos con la clase anidada "GestorAdapter", que nos permite gestionar la lista de elementos que se muestra en pantalla.

La segunda es la clase servidor que extiende la interfaz "Activity", mostrando la lista de servidores del sistema. Aquí también podemos darlos de alta y de baja. Para realizar estas tareas de altas y bajas de los dispositivos, al igual que en la clase "Gestor", utilizamos la clase "Ficheros", que es la que nos permite realizar la lectura del fichero de configuración para cargar los servidores así como realizar las distintas modificaciones. Además, aquí también contamos con la clase anidada GestorAdapter, que nos permite gestionar la lista de elementos que se muestra en pantalla.

La tercera es la clase Web, que también extiende `Activity`, mostrando las imágenes de la cámara IP seleccionada, así como realizar zoom, pausar el visionado,etc. Para ello cuenta con un elemento llamado "WebKit", que es un navegador "open source" al cual se le pueden ir añadiendo funcionalidades según las necesidades, y que nosotros personalizamos para poder visionar las distintas cámaras.

La cuarta es la clase "Rtsp", que también extiende de `Activity`, implementando el visor del vídeo del historial, implementada gracias al streaming mediante RTSP. En ella, podemos pausar y reanudar el streaming. Para ello usamos un reproductor multimedia que soporta dicho streaming gracias a este protocolo de transmisión de datos.

Estas cuatro clases son las vistas principales de la aplicación. La pantalla que se carga de forma inicial es la de la clase "Gestor". Está considerada como el menú principal, desde el cual se puede dirigir uno a visualizar las cámaras, acceder al menú de servidores para realizar las gestiones oportuna o visualizar el historial, además de las operaciones propias de alta y baja de las cámaras. Además, desde esta clase, es desde la única desde la que se puede salir de la aplicación.

A la clase "Web", que es donde se ven las cámaras, sólo se puede acceder desde la clase "Gestor", por lo que, al salir de ella, también se va a parar en ella. Es decir, "Web" solo se relaciona con Gestor.

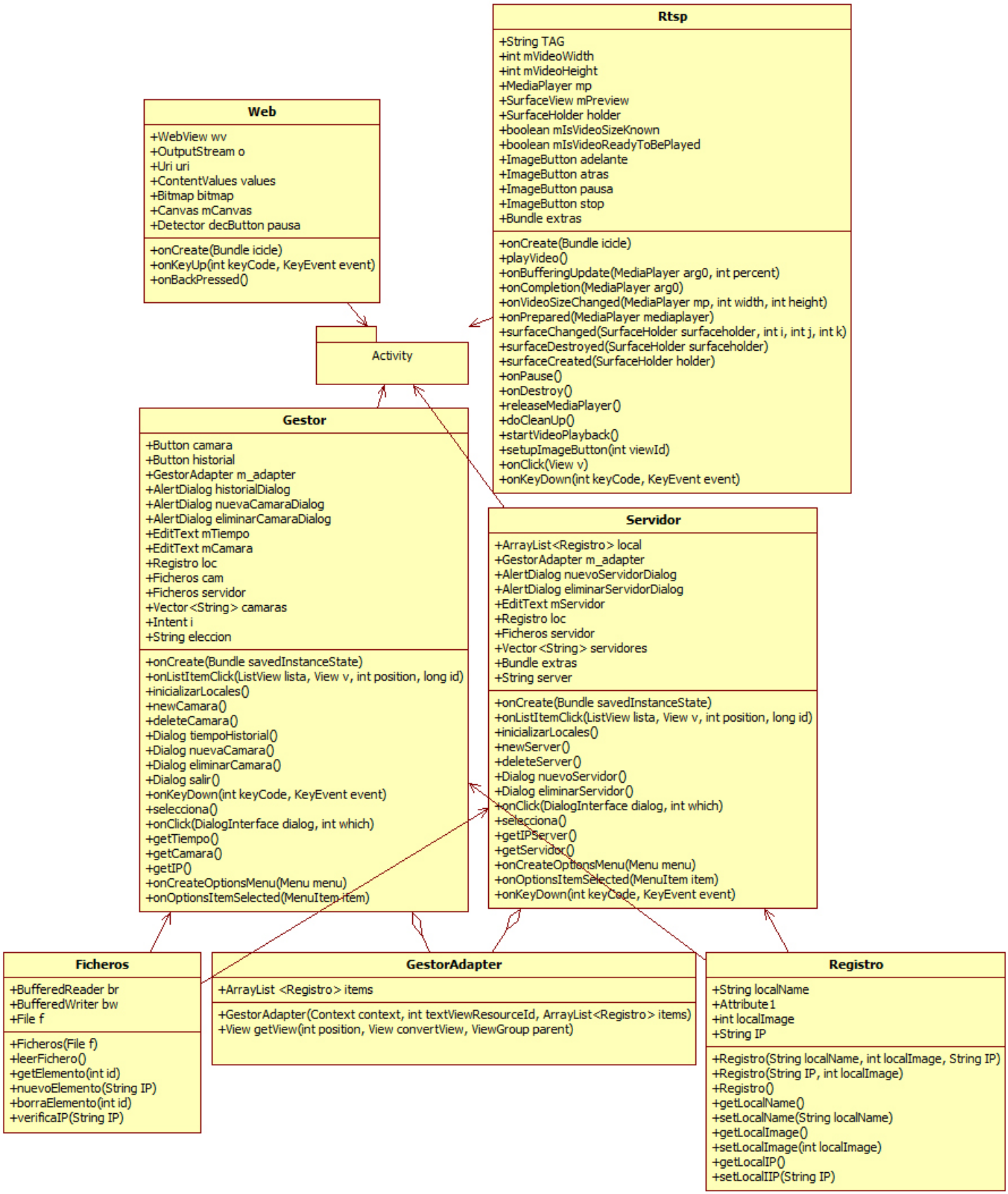


Figura 3.7: Diagrama de clases de la aplicación

Con la clase "Rtsp" es un caso similar a "Web". Para llegar a ella, se puede acceder desde "Gestor" (siempre que haya seleccionado un servidor previamente), o desde "Servidores" (siempre que primero se haya seleccionado una cámara y un tiempo de historial). Pero al salir de ella, se llega únicamente a la clase de los servidores.

A la clase "Servidores" se puede acceder desde el menú gestor, para realizar las tareas de gestión de los servidores que se consideren oportunas o desde el visor del historial (clase "Rtsp") al salir de ella. Y desde "Servidores" se puede volver a "Gestor" con la que mantiene una relación bidireccional, por cuestiones de trato y funcionamiento prácticamente idénticos, y a el visor "Rtsp" cuando previamente se ha seleccionado una cámara y un momento del historial.

Por otro lado tenemos las clases que nos sirven de apoyo para que las cuatro principales realizan su función correctamente, las cuales ya hemos comentado de forma somera anteriormente. La clase "Ficheros" nos proporciona los métodos necesarios para escribir en los ficheros y gestionar así las altas y bajas de cámaras y servidores de las clases "Gestor" y "Servidor", así como sus distintas modificaciones. La clase "Registro" nos proporciona la base para representar cada uno de los elementos que conforman las distintas listas de cámaras y servidores. Y por último tenemos la clase anidada "GestorAdapter", que nos permite gestionar los cambios realizados en las listas, mostrando las altas y las bajas en la pantalla.

3.4. Conclusiones

El sistema se ha diseñado de forma que su uso sea fácil e intuitivo. Se ha buscado la claridad y que el usuario pueda personalizarlo pudiendo dar de alta y de baja, tanto cámaras como servidores.

Por otra parte, para realizar esta aplicación, el entorno de desarrollo "Eclipse", ha sido muy útil ya que proporciona un *plugin* que permite emular el dispositivo, para poder hacer así las pruebas necesarias. Además, la forma que tiene de organizar las clases en una estructura de árbol, así como de organizar los distintos recursos y ficheros generados, facilita enormemente la labor del desarrollador.

En cuanto al lenguaje de programación, Java facilita mucho las cosas, ya que es sencillo de manejar. Además, la parte gráfica de la aplicación, al realizarse en `.xml` simplifica la creación de las distintas interfaces visuales del programa, ya que Eclipse muestra en una ventana como quedaría la interfaz según se va escribiendo el fichero `.xml`, ya que evita tener que estar ejecutando constantemente la aplicación para ver si el desarrollo es el deseado.

Capítulo 4

Pruebas y resultados

En este capítulo se van a presentar las pruebas realizadas a las distintas partes sistema. También se presentará un análisis de los resultados obtenidos y valoración de los mismos.

4.1. Entorno de pruebas

El lugar de pruebas es un recinto cerrado. En él se tiene una cámara IP y un router conectados a un PC. El router ejerce de punto de acceso del dispositivo móvil a la red y el PC como servidor de los vídeos del historial.

Se van a realizar tres tipos de pruebas:

- La primera es la verificación del funcionamiento global del sistema, es decir, ver que se gestionan correctamente las altas y bajas de cámaras y servidores, el paso de una pantalla a otra, los visionados de las cámaras y los servidores, los distintos mensajes, etc.

- La segunda prueba es obtener el tiempo que tarda en comenzar la visualización de los videos. La razón es que vamos a estudiarlo con distintas calidades para elegir una relación calidad-tiempo que nos permita alcanzar unos mínimos en ambos aspectos. Por último, las medidas de tiempo de cuanto se tarda en ver todo el vídeo. Esto es debido a que al ser de distinta calidad los videos, los procesos de buffering del streaming, serán más lentos y por lo tanto distintos.

- La tercera es mostrar pantallazos de la aplicación para verificar de forma visual el correcto funcionamiento de la parte visual de la misma.

A la hora de realizar las pruebas, hemos utilizado unos videos de prueba que vienen con el "Darwin Stream Server". En la tabla anterior, 4.1, se muestran dichos archivos y sus características principales.

Nombre	Tasa	Tamaño
sample_50kbit.3gp	50 kb/s	589.4 Kb
sample_100kbit.mp4	100 kb/s	911.6 Kb
sample_300kbit.mp4	300 kb/s	2.3 Mb
sample_h264_100kbit.mp4	100 kb/s	976 Kb
sample_h264_300kbit.mp4	300 kb/s	2.4 Mb
sample_h264_1mbit.mp4	1000 kb/s	8.5 Mb

Cuadro 4.1: Ficheros de video de prueba

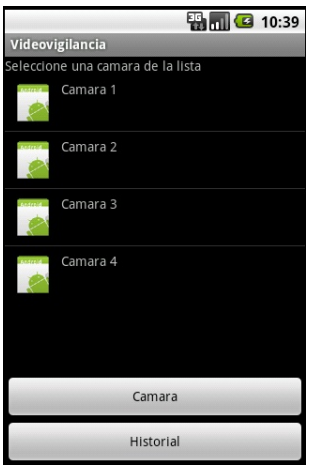
4.2. Resultados

1. Funcionalidad general: verificar que los pasos entre pantallas, mensajes, altas y bajas de cámaras y servidores funciona correctamente. En la tabla 4.2 se muestra un listado de las funcionalidades comprobadas.

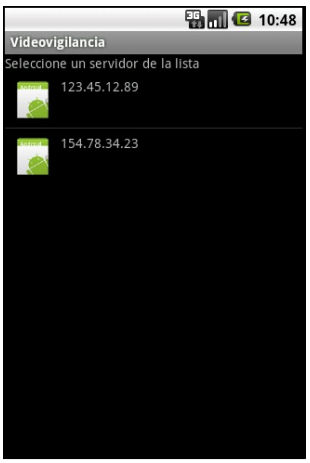
Funcionalidad	Resultado
Carga gestor	✔
Carga servidores	✔
Carga web	✔
Carga rtsp	✔
Alta cámara	✔
Baja cámara	✔
Pausa web	✔
Zoom web	✔
Paso de gestor a servidores	✔
Siguiente video	✔
Anterior video	✔
Play video	✔
Stop video	✔

Cuadro 4.2: Verificación de la funcionalidad general

2. A continuación se muestran algunos pantallazos de la aplicación para poder ilustrar el funcionamiento del sistema.



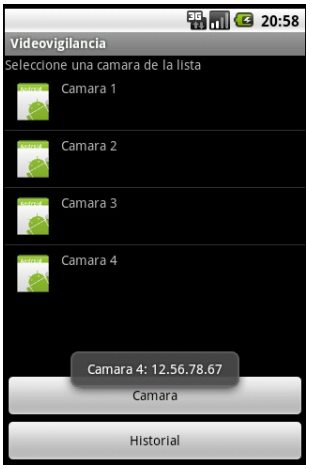
(a) Menú gestor



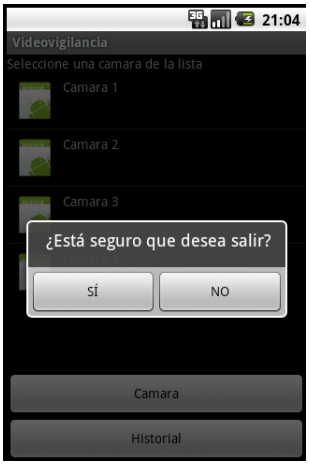
(b) Menú de servidores

Figura 4.1: Ejecuciones

En las figuras 4.1.a y 4.1.b se pueden ver los menús de gestión y de los servidores. Se observa como en cada uno de ellos se muestran las listas de las cámaras registradas en el sistema, así como los distintos servidores para realizar el streaming.



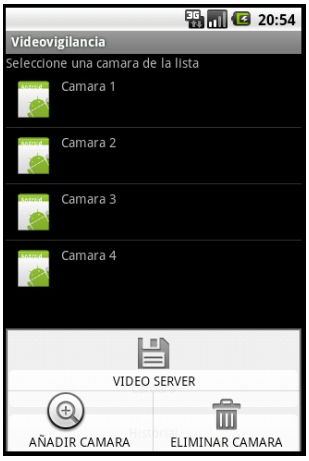
(a) Dirección IP de la cámara



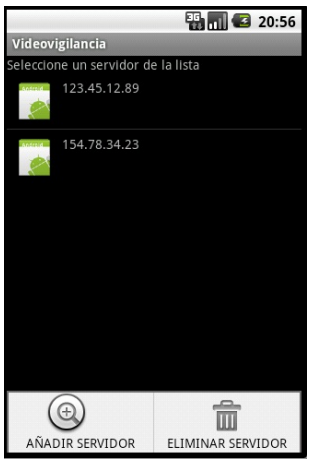
(b) Menú de servidores

Figura 4.2: Ejecuciones

En las figuras 4.2.a y 4.2.b se pueden ver los mensajes emergentes que indican la dirección IP de la cámara seleccionada y el menú para salir de la aplicación.



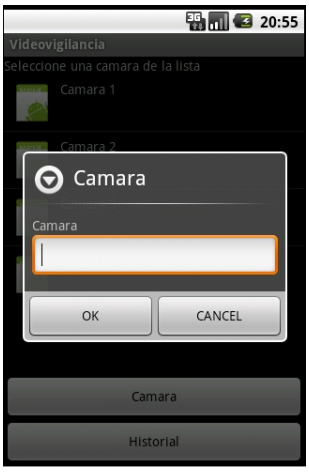
(a) Menú de la pantalla gestor



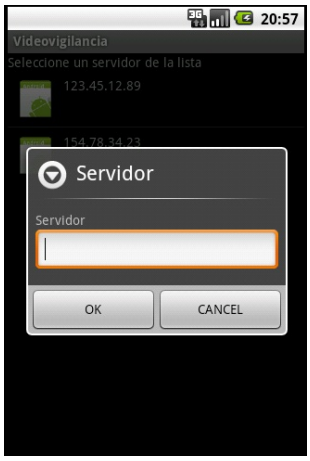
(b) Menú de la pantalla servidores

Figura 4.3: Ejecuciones

En las figuras 4.3.a y 4.3.b se pueden ver los menús que aparecen cuando se pulsa la tecla "Menú" en cada una de los dos pantallas principales. En el menú del gestor se tienen las opciones de dar de alta y baja las cámaras, así como de pasar al menú de los servidores. En el menú de éste tenemos las opciones de dar de alta y bajo los servidores de vídeo.



(a) Menú de la pantalla gestor



(b) Menú de la pantalla servidores

Figura 4.4: Ejecuciones

En las figuras 4.4.a y 4.4.b se pueden ver las pantallas emergentes que aparecen al seleccionar dar de alta o baja una cámara o un servidor. Si es la opción de dar de alta, se debe poner la dirección IP del elemento, con el formato

xxx.xxx.xxx.xxx. Si por el contrario se quiere dar de baja, se tiene que introducir su número identificador de posición.

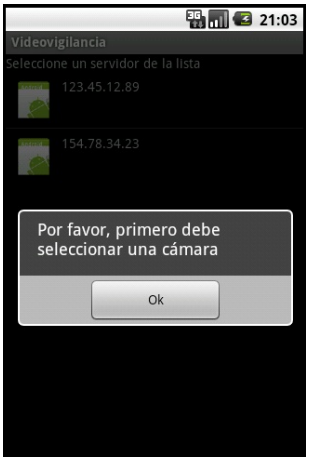
(a) Menú de la pantalla gestor

(b) Menú de la pantalla servidores

Figura 4.5: Ejecuciones

En las figuras 4.5.a y 4.5.b se observan los mensajes emergentes que indican que la dirección IP introducida es incorrecta y que no se puede realizar ninguna operación de visionado en directo o en diferido sin haber seleccionado previamente una cámara.

(a) Menú de la pantalla gestor



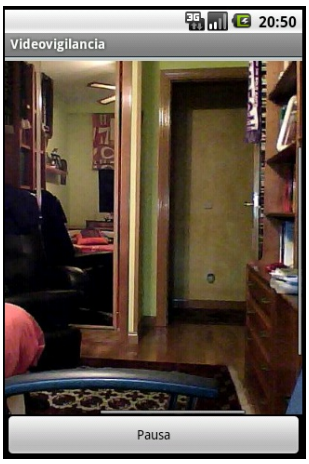
(b) Menú de la pantalla servidores

Figura 4.6: Ejecuciones

En las figuras 4.6.a y 4.6.b se pueden ver la ventana emergente en la que se

tiene que indicar que minuto de la grabación que se quiere ver, y el mensaje emergente que indica que se ha seleccionado un servidor de vídeo sin haber seleccionado previamente una cámara.

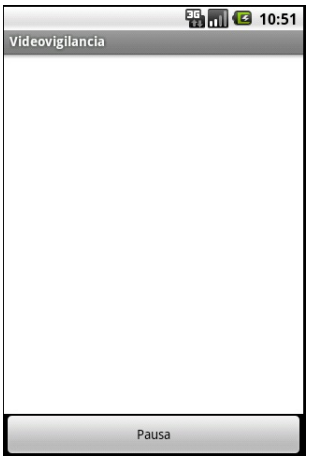
(a) Menú de la pantalla gestor



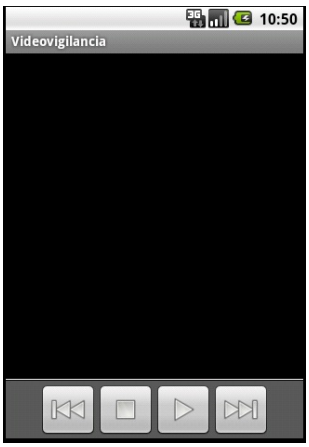
(b) Menú de la pantalla servidores

Figura 4.7: Ejecuciones

En las figuras 4.7.a y 4.7.b se observan los ejemplos de visualización en directo de distintas cámaras.



(a) Vista cámara IP



(b) Vista de la pantalla de vídeo de historiales

Figura 4.8: Ejecuciones

En las figuras 4.8.a y 4.8.b se pueden ver las pantallas de visualización en directo y en diferido justo antes de cargar las imágenes correspondientes.

3. Estudio del tiempo de inicio de streaming en función de la tasa de codificación del vídeo, figura 4.9.

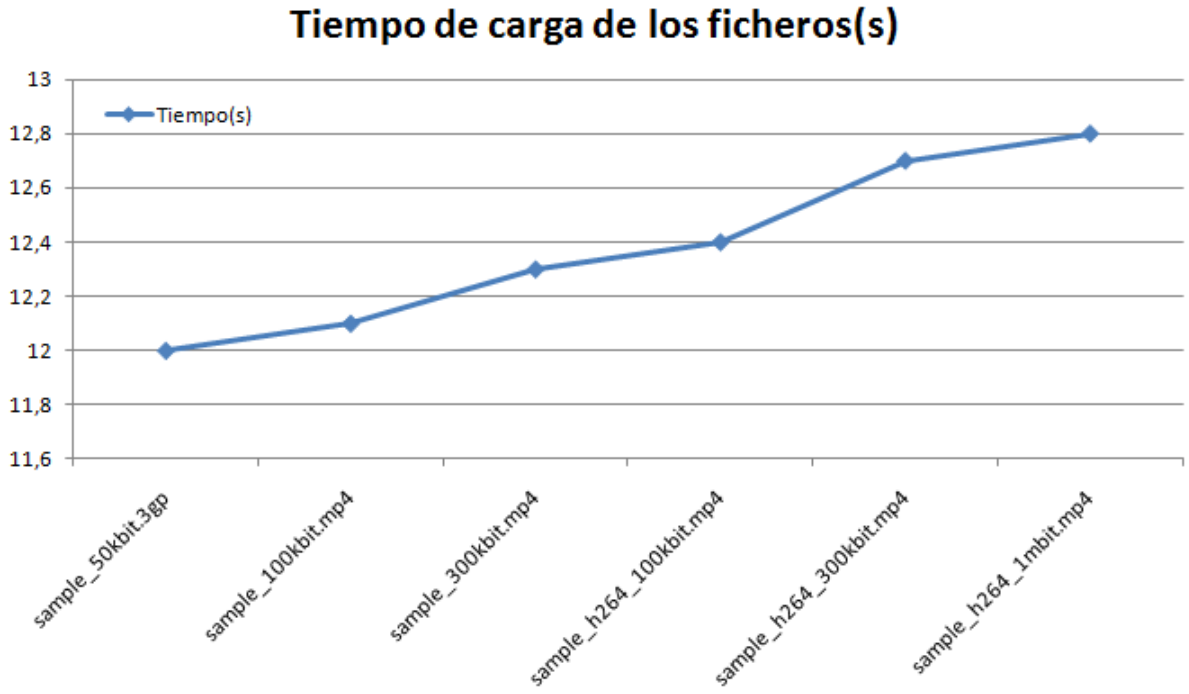


Figura 4.9: Tiempo de comienzo del streaming

Como se puede observar los tiempos son bastante similares, siendo más elevados los que tienen una tasa de codificación mayor.

4. Estudio del tiempo de duración de streaming en función de la tasa de codificación del vídeo, figura 4.10.

   Como se puede ver, según aumenta el tamaño del archivo, es necesario tener mayor tiempo de buffering para poder visualizarlo. Después de esta prueba, hemos podido comprobar, que si bien a mayor tasa mayor es la definición del vídeo, éste no se ve de forma fluida, sino que va dando cortes.

   Por ello, viendo la relación entre la calidad de visualización y los tiempos de carga, los formatos más apropiados para nuestra aplicación serían sample_100kbit.mp4 y sample_300kbit.mp4.

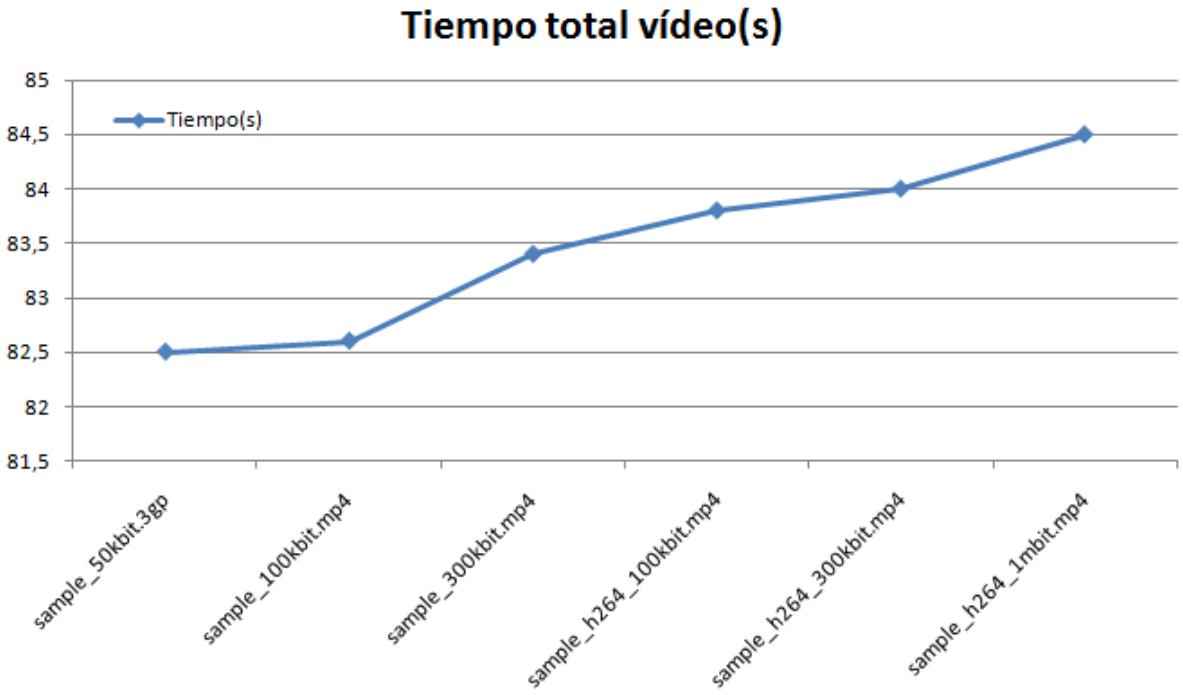


Figura 4.10: Tiempo total de streaming

4.3. Conclusiones

A la vista de los resultados obtenidos tras las realización de las pruebas, se pueden obtener dos conclusiones. La primera es que la funcionalidad general de la aplicación es correcta, ya que la navegación entre las distintas pantallas y menús se realiza de forma ágil y sin errores. La segunda es que el tamaño y la tasa de codificación de las grabaciones influyen en los tiempos de streaming, por lo que se debe llegar a un compromiso entre calidad y el tiempo para servir el vídeo, de forma que calidad del servicio de televigilancia no se vea comprometido.

Así, la elección de los parámetros de calidad de servicio (calidad del vídeo y tiempos de transmisión) dependerá de la red sobre la que se transmita dicho vídeo. Si se dispone de unas instalaciones en las que la velocidad de descarga sea lo suficientemente elevada, se podrá optar por una codificación de una tasa más alta y con mayor definición. Si no es el caso, habrá que ajustarlo de forma que el vídeo se reproduzca de forma fluida y sin cortes.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En este proyecto se ha creado un prototipo de sistema de videovigilancia utilizando dispositivos con sistema operativo Android y la infraestructura de cámaras IP del departamento de Telemática de la Universidad Carlos III de Madrid.

- Se ha conseguido implementar el desarrollo completo del sistema con herramientas Open Source lo que lo hace accesible a todo el mundo.
- Se ha tratado de realizar un diseño eficiente en el que no interviniesen más dispositivos de los necesarios para abaratar costes, por ello, aparte del dispositivo móvil, solo se ha utilizado un PC que actúa como servidor de vídeo RTSP y para alojar el programa Motion que procesa las imágenes recibidas por las cámaras.
- El uso de un fichero de texto sin formato para almacenar las direcciones IP de los servidores como de las cámaras, hace que sea más manejable ya que ocupan menos espacio que cualquier otro formato.
- El sistema operativo sobre el que se ejecutan las aplicaciones es Android. Esto implica que al ser libre, se pueden añadir módulos para distintas funcionalidades así como las modificaciones que se crean necesarias.
- Los mayores problemas encontrados en la realización del proyecto vinieron derivados de la interacción del dispositivo móvil con el servidor de vídeo, ya que Android solo soporta dos formatos de vídeo: `.3gp` y `.mp4`.
- Otro de los grandes problema fue el streaming para recibir vídeo, ya que Android no lo soporta y hubo que emplear el protocolo RTSP para llevarlo acabo, por lo que hubo que encontrar un servidor con dicho protocolo para servir el vídeo.
- También se han tenido problemas con las cámaras IP, ya que las cámaras Axis, para visualizarlas, hacen falta applets, y la JVM que lleva incorporada Android es parcial, no los soporta y por lo tanto no se podían ver las imágenes, por lo que es necesario que las cámaras emitan en *streaming*.

- Es necesario tener en cuenta la capacidad de la red para enviar el vídeo, ya que hay que dividir su capacidad por el número de usuarios que soliciten el historial al vez. A la hora del diseño de la infraestructura habría que especificar unos requisitos de servicio mínimo en el caso peor. Este tema es crítico ya que el tiempo de visionado del vídeo puede ser crucial cuando se produzca una alarma.

5.2. Líneas futuras de trabajo

Las ideas principales para el futuro son:

1. Reconocimiento facial. Con esto se pretende que se remarque la cara del individuo que ha hecho saltar las alarmas de seguridad.
2. Conexión con una base de datos de la policía de forma que si el individuo está fichado, poder acceder a su identidad.
3. Mayor número de formatos soportados por Android para la visualización del historial. Con ello se pretende tener más opciones a la hora de codificar y ver el vídeo.
4. Manejar el movimiento de las cámaras desde el dispositivo móvil. Con esto podríamos ver zonas del recinto que no se cubren de forma habitual con el fin de realizar alguna verificación.
5. Incorporación de *plugin* que permita la visualización de las cámaras en *applets*. La ventaja es que con ello podríamos ver cualquier tipo de cámara IP Axis.
6. Sistema de autentificación. Con ello se pretendería que al sistema solo accediese personal autorizado de seguridad. Con ello se podrían prevenir ataques exteriores para dañar el funcionamiento del sistema.
7. Registro de las personas que acceden al sistema mediante el sistema de autenticación citado en el punto anterior.

Capítulo 6

Presupuesto

En el presente capítulo se presenta un cálculo aproximado del coste de este proyecto. Para la evaluación de dicho coste es necesario un desglose del proyecto en tareas, a través de la planificación del mismo y la evolución temporal que sigue. Cada tarea será cuantificada en función de su duración y los recursos consumidos, proporcionando de manera global un presupuesto aproximado del trabajo desarrollado.

6.1. Descomposición de actividades

A continuación se exponen las cinco tareas llevadas a cabo:

- Fase 1: Documentación para realizar el proyecto y estudio del estado del arte.
- Fase 2: Implementación de la aplicación.
- Fase 3: Instalación y uso del servidor de streaming y el analizado de imágenes.
- Fase 4: Pruebas en entorno real.
- Fase 5: Documentación y memoria.

A continuación pasamos a detallar más en profundidad cada actividad y a especificar el tiempo que nos lleva cada una de ellas.

Fase 1: Documentación para realizar el proyecto y estudio del estado del arte.

a) Estudio de la plataforma Android y del entorno de desarrollo (Eclipse en este caso). Duración 4 semanas.

b) Búsqueda de servidores de vídeo y procesadores de imágenes. Comparación de cada uno de ellos. Duración 5 semanas.

c) Realización de pequeños programas para familiarizarnos con la programación del dispositivo y su funcionamiento. Duración 3 semanas.

Fase 2: Implementación de la aplicación.

a) Implementación del gestor. Duración 3 semanas.

b) Implementación de la clase servidor. Duración 2 semana.

c) Implementación de la clase Web. Duración 2 semanas.

d) Implementación de la clase Vídeo. Duración 6 semanas.

e) Implementación de clases auxiliares. Duración 1 semana.

Fase 3: Instalación y uso del servidor de streaming y el analizado de imágenes.

a) Estudio de la aplicación Motion. Instalación y uso. Duración 2 semanas.

b) Estudio de la aplicación Darwin Stream Server. Instalación y uso. Duración 2 semana.

Fase 4: Pruebas en entorno real.

a) Pruebas del correcto funcionamiento de la aplicación. Duración 2 semana.

b) Pruebas relativas al servidor de vídeo RTSP y a Motion. Duración 2 semana.

Fase 5: Documentación y memoria.

a) Redacción de la memoria. Duración 5 semanas.

b) Corrección y revisión. Duración 3 semana.

6.2. Resumen y duración

En la siguiente tabla se muestra el resumen de las tareas realizadas y de su duración.

Fase	Actividad	Duración (semanas)
Documentación para realizar el proyecto y estudio del estado del arte	a	4
	b	5
	c	3
Implementación de la aplicación	a	3
	b	2
	c	2
	d	6
	e	1
Instalación y uso del servidor de streaming y el analizado de imágenes	a	2
	b	2
Pruebas en entorno real	a	2
	b	2
Documentación y memoria	a	5
	b	3
TOTAL		42

Cuadro 6.1: Tabla de tareas

6.3. Presupuesto

PRESUPUESTO DEL PROYECTO

Autor: Iván Alejandro Fernández Pacheco
Departamento: Ingeniería telemática
Descripción del proyecto

- **Título:** DISEÑO Y DESARROLLO DE UN SISTEMA DE VÍDEO-VIGILANCIA BASADO EN TECNOLOGÍA ANDROID
- **Duración (meses):** 9.8

Presupuesto total del Proyecto (valores en Euros): 37045.66 euros

Desglose presupuestario (costes directos)

Apellidos y Nombre	Categoría	Dedicación (hombres/ mes)	Coste (hombre/ mes)	Coste(Euro)
Estévez Ayres, Iria Manuela	Ingeniero senior	1	4289.54	4289.54
Fernández Pacheco, Iván Alejandro	Ingeniero	9.8	2694.39	26405.02
TOTAL				30694.56

Cuadro 6.2: Costes directos personal

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
PC	599	100	7	60	69.88
Dispositivo móvil con Android	320	100	7	60	37.33
PC (servidor)	599	100	7	60	69.88
TOTAL					177.10

Cuadro 6.3: Costes equipos

Concepto	Presupuesto total(euros)
Coste personal	30694.56
Amortización	177.10
Costes indirectos (20 %)	6174
TOTAL	37045.66

Cuadro 6.4: Costes total

El coste total del proyecto es de treinta y siete mil cuarenta y cinco euros con sesenta y seis céntimos.

Leganés, 22 de Junio de 2010

El ingeniero proyectista

Bibliografía

[1] Murphy, M.L., "Beginning Android", 1ª Edición, Apress, Estados Unidos, 2009

[2] DiMarzio, J.F., "Android. A programmer's guide", 1ª Edición, McGraw Hill, Estados Unidos, 2008

[3] Android developers, Consulta año 2009, <http://developer.android.com>

[4] Open Handset Alliance, "Industry Leaders Announce Open Platform for Mobile Devices", Consulta Enero 2009, <www.openhandsetalliance.com/press_110507.html>

[5] Anddev, Consulta año 2009, <www.anddev.org>

[6] Android-Spa, Consulta año 2009, <www.androidspa.com>

[7] Pranevich, J., Consulta Noviembre 2008, <www.escomposlinux.org/wwol26/wwol26.html>

[8] Motion-Web Home, Consulta Enero 2010, <www.lavrsen.dk/twiki/bin/view/Motion/WebHome>

[9] Darwin Streaming Server , Consulta Febrero 2010, <http://dss.macosforge.org/>

[10] Wikipedia, Consulta Abril 2010, <http://es.wikipedia.org/wiki/RTSP>

[11] IETF, Consulta Abril 2010, <www.ietf.org/rfc/rfc2326.txt>

[12] Laboratorio Docente de Computación, Universidad Simón Bolivar, "RTSP", Consulta Abril 2010, <www.ldc.usb.ve/ poc/RedesII/Grupos/G10/transmision.htm>

[13] Wikipedia, Consulta Mayo 2010, <http://es.wikipedia.org/wiki/Google>

[14] Wikipedia, Consulta Diciembre 2008, <http://es.wikipedia.org/wiki/Android>

[15] IETF, Consulta Marzo 2010, <www.ietf.org/rfc/rfc2326.txt>

[16] Gracía Pañeda, X. y Melendi Palacio, D., "Servicios de Comunicaciones", Consulta Abril 2010, <www.it.uniovi.es/docencia/SistemasOviedo/svccom/material/Tema4.pdf>

[17] Salvachúa, J. y Fumero, A., "Aplicaciones y servicios multimedia", Consulta Abril 2010, <www.slideshare.net/jsalvachua/iba2008-servicios-296547>

[18] Apple, "¿Por qué QuickTime?", Consulta Junio 2009, <www.apple.com/es/quicktime/whyqt/>

[19] Alvarez, M.A., "Descubre la tecnología que nos acerca hacia una Internet de radio y televisión.", Consulta Abril 2010, <http://www.desarrolloweb.com/articulos/482.php>

[20] Wikipedia, Consulta Abril 2010, <http://es.wikipedia.org/wiki/Eclipse_(software)>

Apéndice A

Breve demostración de ejecución

A continuación se muestran algunas salidas por pantalla al ejecutar el emulador de Android sobre Eclipse.

Aquí mostramos la salida del programa más sencillo, el "HelloAndroid", figura A.1. Consiste en sacar por pantalla dicho mensaje. La pantalla negra es un *LinearLayout*, que es un elemento contenedor en el que se pueden posicionar distintos elementos, tales como etiquetas, botones, cuadros de texto, menús... El mensaje es una etiqueta en la que ya está escrito el mensaje.

Figura A.1: Ejecución del Hello android

Para realizar la configuración de la pantalla y sus elementos se ha ha hecho mediante código.*XML*, figura A.2. En el editor se especifican los elementos, su ordenación y su tamaño.

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

Figura A.2: Código XML

Primero se coloca el *LinearLayout* que es la pantalla que lo contiene todo. Después está el *TextView* que una etiqueta. El tamaño de los elementos viene marcado por las etiquetas *width* y *height*. Si se utiliza el valor *fill_parent*, su tamaño será el mayor del contenedor padre en la dirección indicada. Si se pone *wrap_content* será el más pequeño posible.

En el siguiente ejemplo, figura A.3, se muestra la salida del emulador, además de los elementos anteriores, un botón y un cuadro para introducir texto.

Figura A.3: Ejecución de un ejemplo sencillo

Si nos fijamos en el siguiente código, figura A.4, vemos que todos los elementos se estructuran de forma jerárquica. Primero tenemos un *LinearLayout* que es el contenedor principal. Como podemos ver, dentro de él están el resto de los elementos. Estos se ordenaran en vertical ya que así se lo indicamos en su atributo

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

    <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">

        <TextView android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="@string/etiqueta" />
        <EditText android:id="@+id/contenido"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content" />
    </LinearLayout>

        <Button android:id="@+id/boton1"
                android:text="@string/boton1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" />
        <Button android:id="@+id/salir"
                android:text="@string/salir"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" />
</LinearLayout>
```

Figura A.4: Código XML

android:orientation, diciéndole que lo queremos en vertical.

Dentro de este elemento tenemos en una primera línea otro contenedor *LinearLayout* que a su vez contiene una etiqueta (*TextView*) y un cuadro de texto (*EditText*) dispuestos de forma horizontal, ya que lo indicamos en su atributo de orientación. El tamaño de cada uno de sus elementos será el mínimo posible ya que se lo indicamos con *wrap_ content*.

Una vez tenemos situados los elementos en la pantalla, necesitamos tener algún tipo de referencia a ellos para poder usarlos y trabajar con ellos. Esto se consigue mediante el atributo *android:id*. Con este atributo y un segundo fichero *.XML* en el que damos nombre y valores a los componentes, podemos referirnos a ellos para poder trabajar con ellos, figura A.5.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Ejemplo etiqueta y caja de texto</string>
    <string name="etiqueta">Etiqueta 1</string>
    <string name="boton1">Boton1</string>
    <string name="salir">Salir</string>
</resources>
```

Figura A.5: Código XML

Como se puede observar hay unos atributos *string name* y un nombre entre comillas. Ese nombre es con el que referenciamos a los distintos elementos.

También podemos añadir un menú que contenga la misma funcionalidad que los botones, figura A.6. Para ello hace falta sobreescribir el método que nos proporciona Android que crea un menú vacío. En el añadimos tantos elementos como queramos.

Figura A.6: Ejecución del menú

Para añadir los elementos al menú hay que hacerlo programándolo en Java, figura A.7, y no mediante código .XML como los objetos anteriores. Para ello se añadiría el siguiente código.

```
private static final int b1 = Menu.FIRST;
private static final int salir = Menu.FIRST + 1;

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, b1,1, R.string.boton1);
    menu.add(0, salir,2, R.string.salir);
    return true;
}
```

Figura A.7: Código del menú

Una vez construida la interfaz gráfica de la aplicación deseada, tenemos que programar los distintos elementos para que realicen lo que nosotros queremos, figura A.8. Para ello tenemos que añadir manejadores o elementos que respondan a los eventos que se producen, tales como *clicks* de ratón, escritura...

```
Button boton1 = (Button) findViewById(R.id.boton1);
boton1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        texto.setText("Boton pulsado");
    }
});
```

Figura A.8: Manejador de eventos

Como podemos observar, creamos un objeto botón, referenciándolo al que hemos creado en la interfaz gráfica. Posteriormente le añadimos en manejador de eventos, *setOnClickListener*, el cual, dentro de él, se programará las acciones que queremos que realice el botón cuando pinchemos en él.

Por último, para completar esta miniaplicación, tenemos que añadirle la función correspondiente a los elementos del menú para que realicen la misma función que los botones, figura A.9. Para ello añadimos el siguiente código.

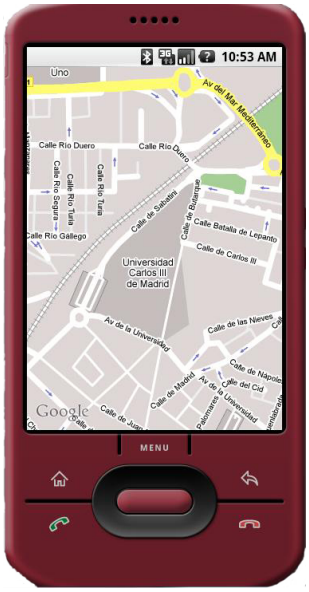
```
public boolean onMenuItemSelected(int featureId, MenuItem item){
    switch( item.getItemId()){
    case b1:
        texto.setText("Boton pulsado");
        break;
    case salir:
        setResult(RESULT_OK);
        finish();
        break;
    }
    return true;
}
```

Figura A.9: Código para la funcionalidad del menú

Como podemos observar, lo primero que hacemos es obtener el identificador del elemento del menú sobre el que hacemos click, figura A.10. Una vez hecho esto, no tenemos más que repetir el mismo código que en los botones para que tengan la misma funcionalidad.

(a) Menú de android

(b) Internet en Android



(c) Google maps

Figura A.10: Algunas ejecuciones

Apéndice B

Análisis del API de Android

Si analizamos el índice de los paquetes que se proporcionan para desarrollar aplicaciones nos encontramos lo siguiente[3]:

1. **android**: Clases usadas para las aplicaciones estándar de Android.

2. **android.app**: Clases de alto nivel que encapsulan el modelo de aplicación de Android. Dentro de este paquete nos encontramos clases como:

   - Activity: una cosa sola, enfocada que el usuario puede hacer.
   - ActivityGroup: pantalla que contiene múltiples activities embebidas.
   - AlarmManager: clase que proporciona acceso a los sistemas de alarma.
   - NotificationManager: clase que notifica los eventos que ocurren.
   - SearchManager: clase que proporciona acceso al sistema de servicios de búsqueda.
   - Service: componente de aplicación que corre en background, que no interactua con el usuario, durante un período indefinido de tiempo.

3. **android.content**: Clases para acceso y publicación de datos en el dispositivo. Algunas de las clases que contiene son:

   - BroadcastReceiver: recibe los Intents enviados por sendBroadcast.

- ComponentName: identificador para un componente de aplicación específico.
- ContentProvider: uno de los bloques primarios de aplicación de Android, preparado para contener aplicaciones.
- Intent: descripción abstracta de una operación que tiene que ser realizada.
- IntentFilter: descripción de los valores de Intent para ser encontrados.

4. **android.content.pm**: para acceder a la información sobre un paquete de aplicación. Algunas de las clases que podemos encontrar en este paquete son:
   - ActivityInfo: información que se puede restaurar sobre una activity o receiver en particular.
   - ApplicationInfo: información que se puede restaurar sobre una aplicación en concreto.
   - PermissionInfo: información que se puede restaurar acerca de permiso de seguridad conocido por el sistema.
5. **android.content.res**: Clases para acceder a los recursos de aplicación, como ficheros, colores, elementos para dibujar, media... Algunas clase que contiene:
   - AssetfileDescriptor: descriptor de fichero de una entrada en AssetManager.
   - AssetManager: proporciona acceso a los ficheros AAAA.
   - ColorStateList: permite mapear el estado de Views a colores.
   - Configuration: describe la información de configuración del dispositivo que puede tener impacto sobre los recursos al restaurar aplicaciones.
   - Resource:clases para acceder a los recursos de aplicación.
6. **android.database.sqlite** : Clases para manejar la base de datos de SQLite. Algunas clases que contiene el paquete son:

- SQLiteDatabase: propone métodos para manejar una base de datos SQLite.
- SQLiteCursor: implementación de cursor que pone los resultados de una DDDD en una SQLiteDataBase.
- SQLiteProgram: clase base para programas SQLite compilados.

7. **android.graphics**: Herramientas gráficas de bajo nivel. Algunas clases relevantes son:

   - BitmapFactory: crea objetos bitmap de varias fuentes, incluidos ficheros, flujos y byte-arrays.
   - Color: define métodos para y convertir colores.
   - Matrix: matriz de 3x3 que permite transformar coordenadas.
   - NinePatch: permite dibujar bitmap en nueve secciones.
   - Paint: clase que mantiene el estilo y la información de colors acerca de como dibujar figuras geométricas, texto y bitmaps.
   - Path: encapsula figurar geométricas compuestas consistentes en líneas rectas, curvas cuadráticas y cúbicas.

8. **android.graphics.drawable**: Clases para manejar los elementos visuales. Clases destacadas del paquete son:

   - AnimatioDrawable: objeto usado para definir animaciones frame a frame.
   - Drawable: es una abstracción acerca de algo que puede ser dibujado.
   - LayerDrawable: Drawable que maneja un array de otros drawables.
   - NinePatchDrawable: un bitmap que se puede modificar el tamaño, con áreas modificables que tu defines.

9. **android.graphics.drawable.shapes**: Clases para dibujar elementos geométricos. Algunas clases interesantes son:

   - ArcShape: crea un arco.
   - OvalShape: define un óvalo.
   - RectShape: define un rectángulo.
   - Shape: define una forma general.

10. **android.hardware**: Soporte para hardware que no esté presente en los dispositivos de Android. A destacar las siguientes clases:

    - Camera.AutoFocusCallBack: maneja la llamada para el autofocus de la cámara.
    - Camera.PictureCallback: maneja la llamada cuando se hace una foto.
    - SensorLister: se usa para recibir notificaciones del SensorManager cuando los valores del sensor han cambiado.

11. **android.location**: Clases para usar el sistema de localización. Podemos destacar las siguientes clases:

    - Address: representa una dirección.
    - Criteria: indica un criterio a la aplicación para seleccionar una localización.
    - Location: clase que representa una localización geográfica.
    - LocationManager: proporciona acceso al servicio de localización.

12. **android.media**: Clases para manejar los interfaces de audio y video. Algunas clases destacadas son:

- MediaPlayer.OnBufferingUpdateListener: definición de interfaz de una llamada invocada indicando status de buffering de un recurso que se trae por la red.

- MediaPlayer.OnCompletionListener: definición de interfaz de una llamada invocada cuando la reproducción de un fichero de audio o video ha sido completada.

- MediaPlayer.OnPreparedListener: indica que está listo para iniciar la reproducción.

- MediaScannerConnection.MediaScannerConnectionClient: interfaz para notificar a los clientes del MediaScannerConnection cuándo una conexión ha sido establecida y el escaneo ha sido finalizado.

13. **android.net**: Clases para el acceso a la red. Algunas clases representativas son:

    - Connectivitymanager: responde a las peticiones del estado de conectividad de la red.

    - MailTo: analiza un esquema URL de mail y luego puede ser preguntada por los parámetros analizados.

    - NervworkInfo: describe el estado de la interfaz de red de un tipo dado, móvil o Wifi.

14. **android.net.wifi**: Clases para manejar las funcionalidades Wi-Fi del dispositivo. Contiene las siguientes clases:

    - ScanResult: devuelve la información acerca de los puntos de acceso encontrados.

    - WifiConfiguration: representa una red Wifi configurada, incluyendo la seguridad de la red.

    - WifiConfiguration.AuthAlgorithm: algoritmos de autentificación.

    - WifoConfiguration.Protocol: protocolos de seguridad reconocidos.

- WifiConfiguration.Status: status de la configuración de una red.
- WifiInfo: describe el estado de una conexión Wifi que está activa o que está en proceso de establecerse.
- WifiManager: clases que proporciona el API primario para manejar los aspectos de conectividad.

15. **android.os**: El sistema operativo básico, paso de mensajes e intercomunicación entre procesos. Algunas de las clases que tiene son las siguientes:

- ConditionVariable: Clase que implementa la variable de condición del paradigma de cerrojo.
- CountDownTimer: planifica una cuenta atrás hasta un valor futuro, con notificaciones en intervalos regulares.
- Debug: proporciona varias funciones de depurado para las aplicaciones.
- Debug.MemoryInfo: se usa para restaurar estadísticas acerca del mapeado en memoria para este proceso.
- Environment: proporciona acceso a las variables de entorno.
- Handler: permite enviar y procesar objetos Message y Runnable asociados con la cola de mensajes.
- HandlerThread: para empezar nuevos hilos que contienen bucles.
- Looper: usada para correr un bucle de mensaje para un hilo.
- Message: define un mensaje que contiene una descripción y objetos datos arbitrarios que pueden ser enviados al manejador.
- MessageQueue: lista de mensaje para ser despachada en un bucle.
- PowerManager: control del estado de la batería del dispositivo.
- Process: herramientas para manejar los procesos del sistema operativo.

- Vibrator: para controlar el vibrador del dispositivo.

16. **android.preference**: Manejo e implementación de las preferencias de la interfaz de usuario. Algunas clases suyas son:

    - CheckBoxPreference: preferencia que proporciona la funcionalidad de un checkbox.
    - EditTextPreference: clase que permite introducir cadenas.
    - PreferenceActivity: muestra una jerarquía de preferencias como una lista.
    - PreferenceGroup: para almacenar múltiple preferencias.
    - PreferenceManager: para crear jerarquías de preferencias.
    - RingtonePreference: permite elegir uno de los tonos de llamada disponibles.

17. **android.telephony**: Clases para la monitorización de información básica del teléfono. Entre ellas contiene:

    - CellLocation: representa la localización del dispositivo.
    - PhoneNumberUtils: utilidades para llamar con cadenas de números.
    - PhoneStateListener: clase que monitoriza cambios en las estados específicos del teléfono, incluido el estado del servicio, la fuerza de la señal, el mensaje de espera y otros.
    - ServiceState: contiene el estado del teléfono y la información relacionada con el servicio.
    - TelephonyManager: proporciona acceso a la información de los servicios del teléfono en el dispositivo.

18. **android.telephony.gsm**: Clases para el uso de GSM. Algunas son:

- GsmCellLocation: representa la localización de la celda.
- SmsManager: maneja las operaciones sobre los SMS, como enviar datos, texto, pdu...
- SmsMessage: SMS.

19. **android.view.animation**: Clases para manejar animaciones. Su clases son:

- AccelerateInterpolator: interpolador donde la tasa de cambio empieza lentamente para luego acelerar.
- Animation: abstracción de una animación que puede ser aplicada a Views, Surfaces u otros objetos.
- AnimationSet: grupo de animaciones que pueden ser utilizadas juntas.
- AnimationUtils: utilidades para trabajar con animaciones.
- RotateAnimation: controla la rotación de un objeto.
- ScaleAnimation: controla la escala de un objeto.

20. **android.webkit**: Clases para recorrer la web. Estas son:

- CacheManager: proporciona una caché de contenido recibido por la red.
- CookieSyncManager: usada para sincronizar las cookies del navegador entre la RAM y FLASH.
- HttpAuthHandler: manejador de autentificación de HTTP.
- JsPromptResult: para manejar peticiones de javascript.
- WebBackForwardList: lista de Webview.

21. **android.widget**: Widgets para usar en las aplicaciones. Las clases que contiene son las siguientes:

- AutoCompleteTextView.Validator: usado para verificar que el texto introducido en un TextView cumple un cierto formato.
- DatePicker.OnDateChangedListener: para indicar al usuario cambios de fecha.
- ExpandableListView.OnGroupCollapseListener: usado para ser notificado cuando se colapsa un grupo.
- Filter.FilterListener: usado para recibir notificaciones después de una operación de filtrado.
- ListAdapter: sirve de puente entre un ListViewy los datos que devuelve la lista.
- RatingBar.OnRatingBarChangeListener: notifica clientes cuando la tasa ha cambiado.

22. **com.google.android.maps**: Paquete para usar Google maps. Sus clases son:
    - GeoPoint: representa la longitud y la latitud.
    - MapActivity: clase para manejar las necesidades que tiene la vista de un mapa.
    - MapController: para manejar el zoom de un mapa.
    - MapView: muestra el mapa.
    - MyLocationOverlay: dibuja la posicion del usuario en el mapa.
23. **java.net**:Paquetes con funcionalidades de red como streaming y sockets, manejo de direcciones de internet.
24. **java.nio.channels**: Clases para conectar fuentes de datos que permitan entrada y salida de datos. Algunas son:
    - Channels: utilidades para manejar flujos de entrada/salida.

- FileChannel: para interactuar con una plataforma de ficheros.
- FileLock: región protegida de un fichero.
- Pipe: contiene dos canales.
- SelectableChannel: canal que puede ser detectado por un detector.
- SelectionKey: clave que representa la relación entre un canal y un selector.

25. **java.security**: Clases para Java security framework. Estas son:

    - DomainCombiner: forma de actualizar la protección de dominios.
    - Guard: interfaz implementada por los objetos que desean mantener el control de acceso a otros objetos.

26. **java.security.interfaces**: Clases con el código necesario para generar claves para RSA, DSA, EC.

27. **java.util.concurrent**: Clases usadas comúnmente en programación concurrente. Algunas son:

    - BlockingQueue: cola que adicionalmente soporta operaciones de espera para la cola cuando esta vacía, y de espera cuando se quiere meter algo y está llena.
    - RejectedExecutionHandler: manejador para tareas que no pueden ser ejecutadas por el ThreadPoolExecutor.
    - ThreadFactory: objeto que crea hilos bajo demanda.

28. **java.util.concurrent.atomic**: Conjunto de utilidades que soportan cerrojo, hilos... Por ejemplo:

    - AtomicBoolean: valor que puede ser actualizado automáticamente.

- AtomicIntegerArray: array cuyos elementos pueden ser actualizados automáticamente.

29. **java.util.concurrent.locks**: Clases que permiten mecanismos para cerrojos, esperas... que son distintas que construir sobre sincronización y monitores. Entre ellas tenemos:

    - AbstractQueuedSynchronizer: proporciona un marco de trabajo para implementar cerrojos bloqueantes y sincronizadores relacionados(semáforos, eventos) que operan sobre una cola FIFO.
    - LockSupport: primitivas para hilos bloqueantes para creación de cerrojos y otras clases sincronizantes.
    - ReentrantLock: cerrojo de exclusión mutua, con los mismos comportamientos básicos y semánticos, como el acceso a monitores usando métodos con syncrhronized, pero con capacidades extendidas.

30. **java.util.logging**: Paquete que permite hacer logs en cualquier aplicación. Algunas de las clases que contiene son:

    - ConsoleHandler: manejador para escribir los mensajes de logueado en la salida estandar.
    - ErrorManager: devuelve errores producidos durante el logueado.
    - FileHandler: escribe descripciones de eventos de logging en un fichero.
    - Handler: objeto que acepta peticiones de logging y exporta el mensaje deseado a ficheros, la consola...
    - LoggingPermission: permisos requeridos para controlar el logging.

31. **javax.net.ssl**: Paquete que contiene todas las clases e interfaces para programar el socket de seguridad sobre la base del SSLv3.0 de protocolo de SSL o TLSv1.2.

32. **javax.sql**: Paquete con extensiones al interfaz estándar de acceso a bases de datos basadas en SQL. Algunas clases son:

- ConnectionEventListener: interfaz usada para recibir eventos generados por PolleConnection.
- PooledConnection: interfaz que proporciona facilidades para el manejo de conexiones a bases de datos que están reunidas.
- DataSource: interfaz para la creación de objetos Connection que representan la conexión a una base de datos.
- ConnectionEvent: evento enviado cuando ocurre un evento en un objeto PooledConnection.

33. **org.apache.http**: El núcleo de clases e interfaces de componentes HTTP. Algunas son:

- ConnectionReuseStrategy: para decidir si una conexión debería mantenerse viva.
- FormattedHeader: cabecera HTTP formateada.
- Header: campo de la cabecera HTTP.
- HttpClientConnection: conexión HTTP usada en el lado del cliente.
- HttpConnection: conexión HTTP genérica útil en el lado del cliente y del servidor.
- HttpEntity: Entity que puede ser enviada en un mensaje HTTP.
- HttpInetConnection: conexión HTTP sobre IP.
- HttpMessage: mensaje HTTP genérico.
- HttpRequest: petición HTTP.
- HttpResponse: respuesta HTTP.
- HttpRequestInterceptor: procesa una petición.

- HttpResponseInterceptor: procesa una respuesta.
- HttpStatus: estado de HTTP.
- HttpHost: mantiene las variables necesarias para describir una conexión HTTP a un host.

34. **org.apache.http.auth**: API del lado del cliente para autentificación HTTP contra el servidor. Algunas clases e interfaces interesantes son:

    - AuthScheme: interfaces que representan un reto orientado a un esquema de autentificación.
    - Credentials: nombre de usuario y contraseña basados en credenciales de autentificación.
    - AUTH: constantes relacionadas con la autentificación HTTP.
    - AuthState: información acerca del estado del proceso de autentificación.
    - BasicUserPrincipal: usuario principal básico para la autentificación HTTP.

Apéndice C

Código

C.1. Gestor

```
/**
*La clase Gestor actúa como menú principal. Desde ella se
    pueden seleccionar las
*cámaras y realizar su gestión. Acceder a su visionado en
    directo o a su historial,
*darlas de alta y de baja...
*
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/

//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;

//Importamos las clases necesarias
import java.io.File;
import java.util.ArrayList;
import java.util.Vector;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ListActivity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

//Clase pública que hereda de ListACtivity e implementa la
    interfaz DialogInterface.OnClickListener
public class Gestor extends ListActivity implements
    DialogInterface.OnClickListener{

  //Declaración de atributos de la clase
  private Button camara;
  private Button historial;
  private ArrayList<Registro> local = null;
  private GestorAdapter m_adapter;
  private AlertDialog historialDialog;
  private AlertDialog nuevaCamaraDialog;
  private AlertDialog eliminarCamaraDialog;
  private EditText mTiempo;
  private EditText mCamara;
  private Registro loc;
  public Ficheros cam;
  public Ficheros servidor;
  public Vector<String> camaras;
  public Intent i;
  public static String eleccion;


public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.menu);

  //Instanciamos los distintos elementos
  cam = new Ficheros(new File("/sdcard/camaras.txt"));
  camara = (Button) findViewById(R.id.camara);
  historial = (Button) findViewById(R.id.historial);
  local = new ArrayList<Registro>();
```

```
    this.m_adapter = new GestorAdapter(this, R.layout.linea,
        local);
    setListAdapter(this.m_adapter);

    //Inicializamos las cámaras según el fichero de
        configuración
    inicializarLocales();

    //Añadimos un evento de click al botón de cámara
    camara.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {

        //Si se ha seleccionado una cámara, se comienza la
            actividad
            if (i!=null){
                eleccion=loc.getLocalIP();
                i.putExtra("IP", eleccion) ;
                startActivity(i);
                i=null;
                //Mostramos mensaje de error si no se ha
                    seleccionado cámara
            }else{
                selecciona();
            }
      }
    });

    //Añadimos un evento de click al botón de historial
    historial.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {

        //Si se ha seleccionado una cámara, se comienza la
            actividad
          if (i!=null){
                tiempoHistorial();
                historialDialog.show();
                i=null;
                //Mostramos mensaje de error si no se ha
                    seleccionado cámara
          }else{
                selecciona();
          }
      }
    });
}
```

```
/**
*Metodo que registra los eventos de click en la lista
*
*@param    lista          Lista de elementos
*@param    v          Vista
*@param    position       Posición en la lista
*@param    id             Identificador
*/

protected void onListItemClick(ListView lista, View v, int
    position, long id) {

  loc = (Registro) lista.getItemAtPosition(position);
  Toast.makeText(this, loc.getLocalName()+": "+loc.getLocalIP
      (),Toast.LENGTH_LONG).show();
  i = new Intent(this, Web.class);
}



/**
*Metodo para inicializar y mostrar la lista de cámaras
*/

private void inicializarLocales(){

  try {
    local = new ArrayList<Registro>();
    camaras = cam.leerFichero();

    if (camaras!=null){
      int p = 1;
      for (int h=0; h<camaras.size(); h++){
        //Añadimos al ArrayList cada una de las cámaras
        local.add(new Registro("Camara "+p,R.drawable.icon,
            camaras.elementAt(h)));
        p++;
      }
    }
    //Capturamos las excepciones
  } catch (Exception e) {
```

```
    Log.e("BACKGROUND_PROC", e.getMessage());
  }

  if(local != null && local.size() > 0){
    for(int i=0;i<local.size();i++)    m_adapter.add(local.get
        (i));
  }
  //Notificamos los cambios a la clase interna para que actue
      oportunamente
  m_adapter.notifyDataSetChanged();
}
```

```
/**
*Metodo para añadir una cámara nueva al sistema
*/

private void newCamara(){

  try {
    local = new ArrayList<Registro>();
    camaras = cam.leerFichero();
    cam.nuevoElemento(getIP());

    if (camaras!=null){
      int tamaño = camaras.size()+1;
      //Añadimos la cámara
      local.add(new Registro("Camara "+tamaño,R.drawable.icon
          , getIP()));
    }
    //Capturamos las posibles excepciones
  } catch (Exception e) {
    Log.e("BACKGROUND_PROC", e.getMessage());
  }
  //Añadimos y notificamos el cambio a la clase interna
  m_adapter.add(local.get(0));
  m_adapter.notifyDataSetChanged();
}
```

```
/**
*Metodo para borrar una cámara de la lista
*/

private void deleteCamara(){

  try {
    cam.borraElemento(getCamara());
  } catch (Exception e) {
    Log.i("BACKGROUND_PROC", e.getMessage());
  }
  m_adapter.clear();
  inicializarLocales();
}




/**
*Metodo que crea el cuadro de diálogo en el que se puede
   indicar que momento
*de la grabación se quiere visualizar
*
*@return  hitorialDialog   Cuadro de diálogo
*/

protected Dialog tiempoHistorial() {

  AlertDialog.Builder builder = new AlertDialog.Builder(this)
     ;
  builder.setTitle(R.string.tiempo);
  LayoutInflater inflater = getLayoutInflater();
  View view = inflater.inflate(R.layout.tiempo, null);
  //Caja de texto donde se añade el tiempo
  mTiempo = (EditText) view.findViewById(R.id.tiempo);
  builder.setView(view);
  //Botones
  builder.setPositiveButton("OK", this);
  builder.setNegativeButton("CANCEL", this);
  historialDialog = builder.create();
  return historialDialog;
}
```

```
/**
*Metodo que crea el cuadro de diálogo en el que se puede
    indicar la IP
*de la cámara que se quiere registrar
*
*@return  nuevaCamaraDialog   Cuadro de diálogo
*/

protected Dialog nuevaCamara() {

   AlertDialog.Builder builder = new AlertDialog.Builder(this)
       ;
   builder.setTitle(R.string.camara);
   LayoutInflater inflater = getLayoutInflater();
   View view = inflater.inflate(R.layout.camara, null);
   //Caja de texto donde se escribe la dirección IP
   mCamara = (EditText) view.findViewById(R.id.camara);
   builder.setView(view);
   //Botones
   builder.setPositiveButton("OK", this);
   builder.setNegativeButton("CANCEL", this);
   nuevaCamaraDialog = builder.create();
   return nuevaCamaraDialog;
}



/**
*Metodo que crea el cuadro de diálogo en el que se puede
    indicar el identificador
*de la cámara que se quiere suprimir
*
*@return  eliminarCamaraDialog   Cuadro de diálogo
*/

protected Dialog eliminarCamara() {

   AlertDialog.Builder builder = new AlertDialog.Builder(this)
       ;
   builder.setTitle(R.string.camara);
   LayoutInflater inflater = getLayoutInflater();
   View view = inflater.inflate(R.layout.camara, null);
   //Caja de texto donde se indica el identifcador de cámara
   mCamara = (EditText) view.findViewById(R.id.camara);
```

```
    builder.setView(view);
    builder.setPositiveButton("OK", this);
    builder.setNegativeButton("CANCEL", this);
    eliminarCamaraDialog = builder.create();
    return eliminarCamaraDialog;
}



/**
*Metodo que crea el cuadro de diálogo en el que se puede
    indicar si se
*quiere salir del programa o no
*
*@return  dialog   Cuadro de diálogo
*/

protected Dialog salir() {

    final Dialog dialog;

    dialog = new AlertDialog.Builder(this).setMessage("¿Está␣
        seguro␣que␣desea␣salir?").setCancelable(false)
      .setPositiveButton("SÍ", new DialogInterface.
          OnClickListener() {

            //Botón del SÍ
            public void onClick(DialogInterface dialog, int id)
                {
              Gestor.this.finish();
            }
    })
            .setNegativeButton("NO",new DialogInterface.
                OnClickListener() {

                //Botón del NO
                  public void onClick(DialogInterface dialog,
                      int id) {
                       dialog.cancel();
                  }
          }).create();

          return dialog;
}
```

```
/**
*Metodo que gestiona los eventos de las distintas teclas del
   PAD
*
*@param    keyCode        Clave de la tecla
*@param    event          Evento
*@return   super.onKeyDown(keyCode, event)
*/

public boolean onKeyDown(int keyCode, KeyEvent event) {
  if (keyCode == KeyEvent.KEYCODE_BACK){
    salir().show();
    return true;
  }
  return super.onKeyDown(keyCode, event);
}




/**
*Metodo que crea el cuadro de diálogo en el que se puede
   indicar que se
*debe seleccionar una cámara
*/

public void selecciona (){

  AlertDialog.Builder builder = new AlertDialog.Builder(this)
     ;
  builder.setMessage("Por favor, seleccione una cámara");
  builder.setCancelable(false);
  builder.setNeutralButton("Ok", new DialogInterface.
     OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {

          }
  });
  builder.create();
  builder.show();
}
```

```
/**
*Metodo que gestiona los clicks que se hacen en los distintos
    cuadros de
*diálogo que se generan
*@param    dialog       Cuadro de diálogo
*@param    which        Indica que botón
*/

public void onClick(DialogInterface dialog, int which) {

  //Opción de historial
  if (dialog == historialDialog) {

    switch (which) {
      case DialogInterface.BUTTON_POSITIVE:

        if(Servidor.server!=null){
         //Si hay un servidor seleccionado, se inicia la
            actividad Rtsp
         final Intent j = new Intent(this, Rtsp.class);
         String fich= loc.getLocalName();
         int t=getTiempo();
         //Le indicamos el nombre del fichero a cargar
         j.putExtra("camara",fich );
         j.putExtra("tiempo",t );
         startActivity(j);
        }else{
         //Si no hay servidor seleccionado, se inicia la
            actividad
         //Servidor para seleccionar el correspondiente
         final Intent h = new Intent(this, Servidor.class);
         String fich= loc.getLocalName();
         int t=getTiempo();
         //Le indicamos el nombre del fichero a cargar
         j.putExtra("camara",fich );
         j.putExtra("tiempo",t );
         startActivity(h);
        }

        break;
      case DialogInterface.BUTTON_NEGATIVE:
        //Cancelamos
        dialog.dismiss();
```

```
            break;
        }
    }

    //Opción de agregar cámara
    if (dialog == nuevaCamaraDialog) {

        switch (which) {
            case DialogInterface.BUTTON_POSITIVE:
                //Agregamos nueva cámara
                if(!cam.verificaIP(getIP())){
                    Toast.makeText(this, "Dirección IP incorrecta",
                        Toast.LENGTH_LONG).show();
                    dialog.dismiss();
                }else{
                    newCamara();
                }
                break;
            case DialogInterface.BUTTON_NEGATIVE:
                //Cancelamos
                dialog.dismiss();
                break;
        }
    }

    //Opción de eliminar cámara
    if (dialog == eliminarCamaraDialog) {

        switch (which) {
            case DialogInterface.BUTTON_POSITIVE:
                //Eliminamos cámara
                deleteCamara();
                break;
            case DialogInterface.BUTTON_NEGATIVE:
                //Cancelamos
                dialog.dismiss();
                break;
        }
    }
}



/**
```

```
*Metodo que obtiene el tiempo introducido en la caja de texto
    del cuadro de
*diálogo del historial
*
*@return  Tiempo introducido
*/

private int getTiempo(){
  String stringValue = String.valueOf(mTiempo.getText());
  if (stringValue.length() != 0) {
    try {
      return Integer.parseInt(stringValue);
    } catch (NumberFormatException e) {
    }
  }
  return 0;
}




/**
*Metodo que obtiene el identificador de cámara de la caja de
   texto
*situada en el cuadro de diálogo de eliminar cámara
*
*@return  Identificador de cámara
*/
private int getCamara(){
  String stringValue = String.valueOf(mCamara.getText());
  if (stringValue.length() != 0) {
    try {
      return Integer.parseInt(stringValue);
    } catch (NumberFormatException e) {
    }
  }
  return 0;
}




/**
*Metodo que obtiene el tiempo introducido en la caja de texto
    del cuadro de
```

```
*diálogo del historial
*
*@return Dirección IP de la cámara
*/

private String getIP(){
  return String.valueOf(mCamara.getText());
}




/**
*Metodo que crea un menú con distintas opciones
*
*@param menu
*@return true
*/

public boolean onCreateOptionsMenu(Menu menu) {
  //Opción del Servidor de Vídeo
  MenuItem servidor = menu.add(Menu.NONE, 1, Menu.NONE, "
      VIDEO_SERVER");
  servidor.setIcon(android.R.drawable.ic_menu_save);

  //Opción de añadir cámara
  MenuItem añadir = menu.add(Menu.NONE, 2, Menu.NONE, "AÑADIR
      _CAMARA");
  añadir.setIcon(android.R.drawable.btn_plus);

  //Opción de eliminar cámara
  MenuItem eliminar = menu.add(Menu.NONE, 3, Menu.NONE, "
      ELIMINAR_CAMARA");
  eliminar.setIcon(android.R.drawable.ic_menu_delete);

  return true;
}




/**
*Metodo que configura cada una de las acciones del menú
*
*@param item
```

```
*@return true o super.onOptionsItemSelected(item)
*/
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case 1:
      //Se inicia la actividad el servidor
      final Intent j = new Intent(this, Servidor.class);
            startActivity(j);
      return true;
    case 2:
      //Se muestra el cuadro de diálogo para añadir cámara
      nuevaCamara();
      nuevaCamaraDialog.show();
      return true;
    case 3:
      //Se muestra el cuadro de diáologo para eliminar cámara
      eliminarCamara();
      eliminarCamaraDialog.show();
      return true;
      default:
      return super.onOptionsItemSelected(item);
  }
}



/**
*Clase interna para gestionar los eventos de la lista
*/

public class GestorAdapter extends ArrayAdapter <Registro> {

  //Atributo de la clase
  private ArrayList <Registro> items;



/**
*Metodo constructor de la clase
*
*@param context                    Contexto
*@param textViewResourceId         Identificador
*@param items                      Array de registros
*@return objeto de la clase
*/
```

```
public GestorAdapter(Context context, int textViewResourceId,
    ArrayList<Registro> items) {
  super(context, textViewResourceId, items);
  this.items = items;
}



/**
*Metodo que devuelve la vista de la lista
*
*@param position        Posición del elemento
*@param convertView
*@param parent
*@return v    Vista
*/

public View getView(int position, View convertView, ViewGroup
    parent) {

  View v = convertView;
  if (v == null) {
    LayoutInflater vi = (LayoutInflater)getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);
    v = vi.inflate(R.layout.linea, null);
  }

  //Se obtiene la posición del elemento
  Registro o = items.get(position);
  if (o != null) {

    //Texto e imagen de cada fila
    TextView tt = (TextView) v.findViewById(R.id.row_toptext)
        ;
    ImageView im = (ImageView) v.findViewById(R.id.icon);

    if (im!= null) {
      //Obtenemos la imagen
      im.setImageResource(o.getLocalImage());
    }
    if (tt != null) {
      //Obtenemos el texto
      tt.setText(o.getLocalName());
```

```
    }
  }
  return v;
}
}
}
```

C.2. Servidor

```
/**
*La clase Servidor actúa como gestor de los distintos
   servidores. Desde ella se pueden
*seleccionar los servidores y realizar su gestión. Darles de
   alta y de baja...
*
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/

//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;

//Importamos las clases necesarias
import java.io.File;
import java.util.ArrayList;
import java.util.Vector;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ListActivity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListView;
```

```
import android.widget.TextView;
import android.widget.Toast;

//Clase pública que hereda de ListACtivity e implementa la
    interfaz DialogInterface.OnClickListener
public class Servidor extends ListActivity implements
    DialogInterface.OnClickListener{

  //Declaración de atributos de la clase
  private ArrayList<Registro> local = null;
  private GestorAdapter m_adapter;
  private AlertDialog nuevoServidorDialog;
  private AlertDialog eliminarServidorDialog;
  private EditText mServidor;
  private Registro loc;
  public Ficheros servidor;
  public Vector<String> servidores;
  public Bundle extras;
  public static String server;


public void onCreate(Bundle savedInstanceState) {

  super.onCreate(savedInstanceState);
  setContentView(R.layout.menuservidor);

  //Instanciamos los distintos elementos
  extras = getIntent().getExtras();
  servidor = new Ficheros(new File("/sdcard/servidores.txt"))
      ;
  local = new ArrayList<Registro>();
  this.m_adapter = new GestorAdapter(this, R.layout.linea,
      local);
  setListAdapter(this.m_adapter);

  //Inicializamos las cámaras según el fichero de
      configuración
  inicializarLocales();
}



/**
*Metodo que registra los eventos de click en la lista
```

```
*
*@param    lista          Lista de elementos
*@param    v          Vista
*@param    position       Posición en la lista
*@param    id             Identificador
*/

protected void onListItemClick(ListView lista, View v, int
    position, long id) {

  loc = (Registro) lista.getItemAtPosition(position);

  if (Gestor.eleccion!=null){
    final Intent i = new Intent(this, Rtsp.class);
    server=loc.getLocalIP();
    i.putExtra("Servidor",loc.getLocalIP() );
    startActivity(i);
    Gestor.eleccion=null;
  }else{
    server=loc.getLocalIP();
    selecciona();
  }
}



/**
*Metodo para inicializar y mostrar la lista de cámaras
*/

private void inicializarLocales(){

  try {
    local = new ArrayList<Registro>();
    servidores = servidor.leerFichero();
    if (servidores!=null){
      for (int h=0; h<servidores.size(); h++){
        //Añadimos al ArrayList cada una de los servidores
        local.add(new Registro(servidores.elementAt(h),R.
            drawable.icon));
      }
    }
    //Capturamos las excepciones
  } catch (Exception e) {
     Log.e("BACKGROUND_PROC", e.getMessage());
```

```
  }

  if(local != null && local.size() > 0){
    for(int i=0;i<local.size();i++)    m_adapter.add(local.get
        (i));
  }

  //Notificamos los cambios a la clase interna para que actue
      oportunamente
   m_adapter.notifyDataSetChanged();
}



/**
*Metodo para añadir un servidor nuevo al sistema
*/

private void newServer(){

  try {
    local = new ArrayList<Registro>();
    servidor.nuevoElemento(getIPServer());

    if (servidores!=null){
      int tamaño = servidores.size()+1;
      //Añadimos el servidor
      local.add(new Registro("Servidor "+tamaño,R.drawable.
          icon, getIPServer()));
    }
    //Capturamos las posibles excepciones
  } catch (Exception e) {
    Log.e("BACKGROUND_PROC", e.getMessage());
  }

  //Añadimos y notificamos el cambio a la clase interna
  m_adapter.add(local.get(0));
  m_adapter.notifyDataSetChanged();
}



/**
*Metodo para borrar un servidor de la lista
*/
```

```
private void deleteServer(){

  try {
    servidor.borraElemento(getServidor());
  } catch (Exception e) {
    Log.e("BACKGROUND_PROC", e.getMessage());
  }
  m_adapter.clear();
  inicializarLocales();
}



/**
*Metodo que crea el cuadro de diálogo en el que se indicar la
    IP del
*servidor
*
*@return  hitorialDialog   Cuadro de diálogo
*/

protected Dialog nuevoServidor() {

  AlertDialog.Builder builder = new AlertDialog.Builder(this)
      ;
  builder.setTitle(R.string.servidor);
  LayoutInflater inflater = getLayoutInflater();
  View view = inflater.inflate(R.layout.servidor, null);
  //Caja de texto donde se añade la IP del servidor
  mServidor = (EditText) view.findViewById(R.id.servidor);
  builder.setView(view);
  //Botones
  builder.setPositiveButton("OK", this);
  builder.setNegativeButton("CANCEL", this);
  nuevoServidorDialog = builder.create();
  return nuevoServidorDialog;
}



/**
*Metodo que crea el cuadro de diálogo en el que se indicar el
    identificador
*del servidor a eliminar
```

```
*
*@return   hitorialDialog   Cuadro de diálogo
*/

protected Dialog eliminarServidor() {

  AlertDialog.Builder builder = new AlertDialog.Builder(this)
     ;
  builder.setTitle(R.string.servidor);
  LayoutInflater inflater = getLayoutInflater();
  View view = inflater.inflate(R.layout.servidor, null);
  //Caja de texto donde se añade identificador del servidor
  mServidor = (EditText) view.findViewById(R.id.servidor);
  builder.setView(view);
  //Botones
  builder.setPositiveButton("OK", this);
  builder.setNegativeButton("CANCEL", this);
  eliminarServidorDialog = builder.create();
  return eliminarServidorDialog;
}




/**
*Metodo que gestiona los clicks que se hacen en los distintos
    cuadros de
*diálogo que se generan
*@param   dialog      Cuadro de diálogo
*@param   which       Indica que botón
*/

public void onClick(DialogInterface dialog, int which) {

  //Opción de nuevo servidor
  if (dialog == nuevoServidorDialog) {

    switch (which) {
      case DialogInterface.BUTTON_POSITIVE:
        //Añadimos servidor
        if(!servidor.verificaIP(getIPServer())){
          Toast.makeText(this, "Dirección␣IP␣incorrecta",
              Toast.LENGTH_LONG).show();
          dialog.dismiss();
        }else{
```

```
            newServer();
          }
          break;
        case DialogInterface.BUTTON_NEGATIVE:
          //Cancelamos
          dialog.dismiss();
          break;
      }
    }

    //Opción de eliminar el servidor
    if (dialog == eliminarServidorDialog) {

      switch (which) {
        case DialogInterface.BUTTON_POSITIVE:
          //Borramos servidor
          deleteServer();
          break;
        case DialogInterface.BUTTON_NEGATIVE:
          //Cancelamos
          dialog.dismiss();
          break;
      }
    }
}



/**
*Metodo que crea el cuadro de diálogo en el que se puede
    indicar que se
*debe seleccionar una cámara
*/

public void selecciona (){

    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        ;
    builder.setMessage("Por favor, primero debe seleccionar una
        cámara");
    builder.setCancelable(false);

    builder.setNeutralButton("Ok", new DialogInterface.
        OnClickListener() {
```

```
    public void onClick(DialogInterface dialog, int id) {
      Servidor.this.finish();
    }
  });
  builder.create();
  builder.show();
}



/**
*Metodo que obtiene el tiempo introducido en la caja de texto
    del cuadro de
*diálogo
*
*@return Dirección IP del servidor
*/

private String getIPServer(){
  return String.valueOf(mServidor.getText());
}



/**
*Metodo que obtiene el identificador del servidor de la caja
   de texto
*situada en el cuadro de diálogo de eliminar servidor
*
*@return  Identificador de cámara
*/

private int getServidor(){
  String stringValue = String.valueOf(mServidor.getText());
  if (stringValue.length() != 0) {
    try {
      return Integer.parseInt(stringValue);
    } catch (NumberFormatException e) {
    }
  }
  return 0;
}
```

```
/**
*Metodo que crea un menú con distintas opciones
*
*@param menu
*@return true
*/

public boolean onCreateOptionsMenu(Menu menu) {
  MenuItem añadir = menu.add(Menu.NONE, 1, Menu.NONE, "AÑADIR
      _SERVIDOR");
  añadir.setIcon(android.R.drawable.btn_plus);
  MenuItem eliminar = menu.add(Menu.NONE, 2, Menu.NONE, "
      ELIMINAR_SERVIDOR");
  eliminar.setIcon(android.R.drawable.ic_menu_delete);
  return true;
}
```

```
/**
*Metodo que configura cada una de las acciones del menú
*
*@param item
*@return true o super.onOptionsItemSelected(item)
*/

public boolean onOptionsItemSelected(MenuItem item) {

  switch (item.getItemId()) {

    case 1:
      //Se muestra el cuadro de nuevo servidor
      nuevoServidor();
      nuevoServidorDialog.show();
      return true;
    case 2:
      //Se muestra el cuadro de borrar servidor
      eliminarServidor();
      eliminarServidorDialog.show();
      return true;
    default:
```

```
        return super.onOptionsItemSelected(item);
    }
}
```

```
/**
*Metodo que gestiona los eventos de las distintas teclas del
   PAD
*
*@param    keyCode         Clave de la tecla
*@param    event           Evento
*@return   super.onKeyDown(keyCode, event)
*/

public boolean onKeyDown(int keyCode, KeyEvent event) {

    if (keyCode == KeyEvent.KEYCODE_BACK){
      Servidor.this.finish();
      return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

```
/**
*Clase interna para gestionar los eventos de la lista
*/

public class GestorAdapter extends ArrayAdapter<Registro> {

    //Atributo de la clase
    private ArrayList<Registro> items;
```

```
/**
*Metodo constructor de la clase
*
*@param context                  Contexto
*@param textViewResourceId       Identificador
*@param items                    Array de registros
```

```
*@return objeto de la clase
*/

public GestorAdapter(Context context, int textViewResourceId,
    ArrayList<Registro> items) {
  super(context, textViewResourceId, items);
  this.items = items;
}

/**
*Metodo que devuelve la vista de la lista
*
*@param position        Posición del elemento
*@param convertView
*@param parent
*@return v    Vista
*/

public View getView(int position, View convertView, ViewGroup
    parent) {

  View v = convertView;
  if (v == null) {
    LayoutInflater vi = (LayoutInflater)getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);
    v = vi.inflate(R.layout.linea, null);
  }
  Registro o = items.get(position);
  if (o != null) {

    TextView tt = (TextView) v.findViewById(R.id.row_toptext)
        ;
    ImageView im = (ImageView) v.findViewById(R.id.icon);

    if (im!= null) {
      im.setImageResource(o.getLocalImage());
    }
    if (tt != null) {
      tt.setText(o.getLocalIP());
    }
  }
  return v;
}
}
}
```

C.3. Web

```
/**
*La clase Web permite visualizar las distintas cámaras web
    del sistema de
*seguridad, así como hacer zoom sobre las imágenes.
*
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/

//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;

//Importamos las clases necesarias
import java.io.FileOutputStream;
import java.io.OutputStream;

import android.app.Activity;
import android.content.ContentValues;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Picture;
import android.graphics.Bitmap.Config;
import android.net.Uri;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.webkit.WebView;
import android.widget.Button;


//Clase pública que hereda de Activity
public class Web extends Activity{

  //Declaración de atributos de la clase
  WebView wv;
  Uri uri;
  ContentValues values;
  Bitmap bitmap;
  Canvas mCanvas;
  private Button pausa;

public void onCreate(Bundle icicle){
```

```
  super.onCreate(icicle);
  Bundle extras = getIntent().getExtras();

  //Instanciamos los distintos elementos
        setContentView(R.layout.main);
        pausa = (Button) findViewById(R.id.pausa);
        //face = new FaceView(this);
        String cam = extras.getString("IP");

        wv = (WebView) findViewById(R.id.web);
        wv.getSettings().setJavaScriptEnabled(true);
        wv.loadUrl("http://"+cam+":8080");

  //Añadimos un evento de click al botón de pausa/reanudar
  pausa.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      if (pausa.getText()=="Pausa") {
        pausa.setText("Reanudar");
            }else{
              pausa.setText("Pausa");
            }
    }
  });
}



/**
*Metodo que gestiona los eventos de las distintas teclas del
   PAD
*
*@param    keyCode        Clave de la tecla
*@param    event          Evento
*/

public boolean onKeyUp(int keyCode, KeyEvent event){

  if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN){

  }

  //Hacer +Zoom
  if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT){
```

```
            wv.zoomIn();
            return super.onKeyUp(keyCode, event);
    }

    //Hacer -Zoom
    if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT){

            wv.zoomOut();
            return super.onKeyUp(keyCode, event);
    }

    if (keyCode == KeyEvent.KEYCODE_DPAD_UP){

            return super.onKeyUp(keyCode, event);
    }

    //Botón de volver atrás
    if (keyCode == KeyEvent.KEYCODE_BACK){
            Gestor.eleccion=null;
      Web.this.finish();
      return true;
    }
    return false;
}




/**
*Metodo que gestiona el botón de volver atrás
*/

public void onBackPressed(){

    Gestor.eleccion=null;
    Web.this.finish();
}

}
```

C.4. Rtsp

```
/**
*La clase Rtsp implementa el reproductor de vídeo para
    visualizar
*el streaming servido mediante RTSP
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/


//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;

//Importamos las clases necesarias
import android.app.Activity;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnBufferingUpdateListener;
import android.media.MediaPlayer.OnCompletionListener;
import android.media.MediaPlayer.OnPreparedListener;
import android.media.MediaPlayer.OnVideoSizeChangedListener;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;



//Clase que hereda de Activity e implementa
    OnBufferingUpdateListener, OnCompletionListener,
    OnPreparedListener, OnVideoSizeChangedListener,
    SurfaceHolder.Callback, OnClickListener

public class Rtsp extends Activity implements
    OnBufferingUpdateListener, OnCompletionListener,
    OnPreparedListener, OnVideoSizeChangedListener,
    SurfaceHolder.Callback, OnClickListener{
```

```
  //Declaración de atributos de la clase
  private static final String TAG = "MediaPlayerDemo";
  private int mVideoWidth;
  private int mVideoHeight;
  private MediaPlayer mp;
  private SurfaceView mPreview;
  private SurfaceHolder holder;
  private boolean mIsVideoSizeKnown = false;
  private boolean mIsVideoReadyToBePlayed = false;
  private ImageButton adelante;
  private ImageButton atras;
  private ImageButton pausa;
  private ImageButton stop;
  public Bundle extras;

public void onCreate(Bundle icicle){

  super.onCreate(icicle);
  setContentView(R.layout.streaming);
  extras = getIntent().getExtras();

  //Instanciamos los distintos elementos
  pausa = setupImageButton(R.id.pausa);
  stop = setupImageButton(R.id.stop);
  atras = setupImageButton(R.id.adelante);
  adelante = setupImageButton(R.id.atras);
  mPreview = (SurfaceView) findViewById(R.id.surface);
  holder = mPreview.getHolder();
  holder.addCallback(this);
  holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
}



/**
*Metodo que crea el MediaPlayer y lo ejecuta
*/

private void playVideo(){
  doCleanUp();
  mp = new MediaPlayer();
  try {
    mp.setDataSource("rtsp://"+extras.getString("Servidor")+"
        /"+extras.getString("camara")+"t"+extras.getInt("
```

```
        tiempo"));
    mp.setDisplay(holder);
    mp.prepare();
    mp.setOnBufferingUpdateListener(this);
    mp.setOnCompletionListener(this);
    mp.setOnPreparedListener(this);
    mp.setOnVideoSizeChangedListener(this);
    mp.setAudioStreamType(AudioManager.STREAM_MUSIC);
  } catch (Exception e) {

    e.printStackTrace();
  }


/**
*Metodo que implementa la interfaz y controla la
    actualización del buffer
*@param    arg0        MediaPlayer
*@param    percent     Porcentaje del buffer
*/

public void onBufferingUpdate(MediaPlayer arg0, int percent)
    {

  Log.d(TAG, "onBufferingUpdate percent:" + percent);
}


/**
*Metodo que implementa la interfaz y verefica si esta
    completo
*@param    arg0        MediaPlayer
*/

public void onCompletion(MediaPlayer arg0) {
  Log.d(TAG, "onCompletion called");
}




/**
*Metodo que implementa la interfaz y controla los cambios de
    tamaño
*@param    mp          MediaPlayer
```

```
*@param    width     Ancho
*@param    height    Altura
*/

public void onVideoSizeChanged(MediaPlayer mp, int width, int
    height) {

  Log.v(TAG, "onVideoSizeChanged called");

  if (width == 0 || height == 0) {
    Log.e(TAG, "invalid video width(" + width + ") or height(
        " + height + ")");
    return;
  }
  mIsVideoSizeKnown = true;
  mVideoWidth = width;
  mVideoHeight = height;
  if (mIsVideoReadyToBePlayed && mIsVideoSizeKnown) {
    startVideoPlayback();
  }
}




/**
*Metodo que implementa la interfaz y verifica que el
    reproductor
*esté preparado
*@param    mediaplayer        MediaPlayer
*/

public void onPrepared(MediaPlayer mediaplayer) {

  Log.d(TAG, "onPrepared called");

  mIsVideoReadyToBePlayed = true;
  if (mIsVideoReadyToBePlayed && mIsVideoSizeKnown) {
    startVideoPlayback();
  }
}
```

```
/**
*Metodo que implementa la interfaz e indica que la supercie
    ha cambiado
*
*@param    surfaceholder     Superficie
*@param    i
*@param    j
*@param    k
*/

public void surfaceChanged(SurfaceHolder surfaceholder, int i
    , int j, int k) {
  Log.d(TAG, "surfaceChanged called");
}



/**
*Metodo que implementa la interfaz y controla la destrucción
    del buffer
*
*@param    surfaceholder    Superfice
*/

public void surfaceDestroyed(SurfaceHolder surfaceholder) {
  Log.d(TAG, "surfaceDestroyed called");
}



/**
*Metodo que implementa la interfaz y controla la creación de
    la superficie
*
*@param    holder       Superfice
*/

public void surfaceCreated(SurfaceHolder holder) {
  Log.d(TAG, "surfaceCreated called");
  playVideo();
}
```

```
/**
*Metodo que implementa la pausa en el reproductor
*/

protected void onPause() {
  super.onPause();
  releaseMediaPlayer();
  doCleanUp();
}




/**
*Metodo que implementa el fin del reproductor
*
*@param    arg0        MediaPlayer
*@param    percent     Porcentaje del buffer
*/

protected void onDestroy() {
  super.onDestroy();
  releaseMediaPlayer();
  doCleanUp();
}




/**
*Metodo que implementa la continuación del reproductor
*/

private void releaseMediaPlayer() {
  if (mp != null) {
    mp.release();
    mp = null
  }
}




/**
*Metodo que implementa el reseteo del reproductor
```

```
*
*@param    arg0        MediaPlayer
*@param    percent     Porcentaje del buffer
*/

private void doCleanUp() {
  mVideoWidth = 0;
  mVideoHeight = 0;
  mIsVideoReadyToBePlayed = false;
  mIsVideoSizeKnown = false;

}




/**
*Metodo que implementa el play del reproductor
*/

private void startVideoPlayback() {
  Log.v(TAG, "startVideoPlayback");
  holder.setFixedSize(mVideoWidth, mVideoHeight);
  mp.start();
}



/**
*Metodo que gestiona las imágenes de los botones
*
*@param    viewId       MediaPlayer
*/

private ImageButton setupImageButton(int viewId) {

  ImageButton button = (ImageButton) findViewById(viewId);
  if (button != null) {
    button.setOnClickListener(this);
  }
  return button;
}
```

```
/**
*Metodo que gestiona los clicks de los botones
*/

public void onClick(View v) {
  if (v == adelante) {
    try {
      int tiempo=extras.getInt("tiempo")-1;
      mp.setDataSource("rtsp://"+extras.getString("Servidor")
          +"/"+extras.getString("camara")+"t"+tiempo);
    } catch (Exception e) {
      e.printStackTrace();
    }
  } else if (v == stop) {
    mp.pause();
  } else if (v == pausa) {
    mp.release();
  } else if (v == atras) {
    try {
      int tiempo=extras.getInt("tiempo")+1;
      mp.setDataSource("rtsp://"+extras.getString("Servidor")
          +"/"+extras.getString("camara")+"t"+tiempo);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
/**
*Metodo que gestiona los eventos de las distintas teclas del
   PAD
*
*@param    keyCode         Clave de la tecla
*@param    event    Evento
*@return   super.onKeyDown(keyCode, event)
*/

public boolean onKeyDown(int keyCode, KeyEvent event) {

  if (keyCode == KeyEvent.KEYCODE_BACK) Rtsp.this.finish();
```

```
    return true;
  }
}
```

C.5. Registro

```
/**
*La clase Registro representa cada una de los elementos de la
    línea y correspondientes
*atributos de cada línea de las listas
*
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/

//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;

//Declaración de la clase
public class Registro{

  //Declaración de los atributos de la clase
  private String localName;
  private int localImage;
  private String IP;



/**
*Metodo constructor de la clase
*
*@param   localName     Nombre identificador
*@param   localImage  Icono
*@param   IP            Dirección IP
*/

public Registro(String localName, int localImage, String IP){
  this.localName = localName;
  this.localImage = localImage;
  this.IP=IP;
}



/**
*Metodo constructor de la clase
*
*@param   localImage  Icono
*@param   IP            Dirección IP
```

```
*/

public Registro(String IP, int localImage){
  this.localImage = localImage;
  this.IP=IP;
}



/**
*Metodo constructor de la clase por defecto
*/

public Registro(){

}



/**
*Metodo que devuelve el nombre del elemento
*
*@return localName  Nombre del elemento
*/

public String getLocalName() {
  return localName;
}



/**
*Metodo para asignar el nombre del elemento
*
*@param localName  Nombre del elemento
*/

public void setLocalName(String localName) {
  this.localName = localName;
}



/**
*Metodo que devuelve la imagen
```

```
*
*@return localImage  Imagen
*/

public int getLocalImage() {
  return localImage;
}



/**
*Metodo para asignar la imagen
*
*@param localImage  Imagen
*/

public void setLocalImage(int localImage) {
  this.localImage = localImage;
}



/**
*Metodo que devuelve la IP
*
*@return IP  Dirección IP
*/

public String getLocalIP() {
  return IP;
}



/**
*Metodo para asignar la IP
*
*@param  IP  Dirección IP
*/

public void setLocalIIP(String IP) {
  this.IP = IP;
}
}
```

C.6. Ficheros

```
/**
*La clase Ficheros contiene las utilidades necesarias para
    escribir y leer
*los ficheros de configuración tanto de las cámaras como de
    los servidores,
*así como interactuar con ellos
*
*@author Iván Alejandro Fernández Pacheco
*@version 1.0 Abril 2010
*/

//Paquete de trabajo del proyecto
package proyecto.Videovigilancia;


//Importamos las clases necesarias
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Vector;

import android.util.Log;


//Declaración de la clase
public class Ficheros {

  //Declaración de los atributos de clase
  BufferedReader br;
  BufferedWriter bw;
  File f;



/**
*Metodo constructor de la clase
*
*@param    f     Fichero de configuración
*/
```

```
public Ficheros(File f){

  this.f=f;
  if(f.isDirectory()) Log.i("Error", "El fichero de camaras
     dado es un directorio");
        if(!f.exists()) Log.i("Error", "El fichero de camaras
           esta vacio");
}




/**
*Metodo para lectura del fichero
*
*@return   elementos    Elementos leídos del fichero
*/

public Vector<String> leerFichero() throws Exception{

  String s="";
  //Creación del vector a devolver
  Vector<String> elementos = new Vector<String>();

  try{
    //Creación del BufferedReader para leer del fichero
    br=new BufferedReader(new FileReader(f));

    //Comprobamos que no haya líneas al prinicipio
    if(s.equals("")){
      while(s.equals("") && s!=null){
        s=br.readLine();
      }
    }

    //Comprobamos que no haya líneas vacias ni nulas
    while(s!=null && !s.equals("")){

      elementos.add(s);
      s=br.readLine();
    }
    br.close();
  }
  catch(Exception excepcion){
```

```
    Log.i("Error", "Se ha producido errores en la lectura del
        fichero: ");
    Log.i("Error", excepcion.getMessage());
    return null;
  }
  return elementos;
}



/**
*Metodo que devuelve un elemento del fichero
*
*@param  id    Identificador del elemento
*@param  s     Dirección IP
*/

public String getElemento (int id){

  int i = 1;
  String s;

  try{
    //Creación del BufferedReader para leer del fichero
    br = new BufferedReader(new FileReader(f));
    s = br.readLine().trim();

    //Comprobamos que no haya líneas vacías ni nulas
    while(s!=null && !s.equals("")){

      if (id == i) return s;
      else s = br.readLine().trim();
    }
    br.close();
  }
  catch(IOException excepcion){

    Log.i("Error", "Se ha producido errores en la lectura del
        fichero: ");
    Log.i("Error", excepcion.getMessage());
  }
  return null;
}
```

```
/**
*Metodo para añadir un nuevo elemento
*
*@param    IP
*/

public void nuevoElemento(String IP){

  try {
    //Leemos el fichero de configuración
    Vector<String> c = leerFichero();
    if (c==null)c= new Vector<String>();

    c.addElement(IP);

    //Creamos un BufferedWriter para escribir
    bw=new BufferedWriter(new FileWriter(f));

    for (int i = 0; i<c.size(); i++){
      bw.write((String)c.elementAt(i));
      bw.newLine();
    }
    bw.close();
    bw.flush();

  } catch (Exception e) {
    Log.i("Error", "Se ha producido un error al añadir la
        camara: ");
           Log.i("Error", e.getMessage());
  }
}
```

```
/**
*Metodo para borrar un elemento
*
*@param  id    Identificador del elemento
*/

public void borraElemento(int id){
```

```
  try {
    //Leemos el fichero de configuración
    Vector<String> c = leerFichero();
    Vector<String>nuevo=new Vector<String>();

    for (int i = 0; i<c.size(); i++){
      if (id!=(i+1)) {
        nuevo.addElement((String)c.elementAt(i));
      }
    }

    //Reescribimos el fichero con los cambios realizados
    bw=new BufferedWriter(new FileWriter(f));

    for (int i = 0; i<nuevo.size(); i++){
      bw.write((String)nuevo.elementAt(i));
      bw.newLine();
    }
    bw.close();

  } catch (Exception e) {
    Log.i("Error", "Se ha producido un error al borrar la
       camara: ");
          Log.i("Error", e.getMessage());
  }
}



/**
*Metodo para verificar que la estructura de la direcciín IP
   es correcta.
*
*@param   ip    Dirección a vaerificar
*/

public boolean verificaIP(String ip){

  String octetos[] = ip.split("\\.");
  int num;

  if (octetos.length!=4){
    return false;
  }
```

```
  else{

    for (int i=0; i<octetos.length; i++){
      try{
        num = Integer.parseInt(octetos[i]);
        System.out.println(octetos[i]);
        if (num>255 || num <0) throw new Exception("Error");
      }catch(Exception e){
        return false;
      }
    }
  return true;
}
}
}
```