

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros de Telecomunicación



TRABAJO DE FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

DISEÑO DE ESTRATEGIAS PARA LA GESTIÓN DEL AUDIO EN
TELÉFONOS INTELIGENTES SOBRE EL SISTEMA OPERATIVO
ANDROID

DESIGN OF STRATEGIES FOR AUDIO MANAGEMENT IN
SMARTPHONES OVER ANDROID OPERATIVE SYSTEM

Diego Poderoso Novo
Junio 2017

Trabajo de Fin de Grado

Título: Diseño de estrategias para la gestión del audio en teléfonos inteligentes sobre el sistema operativo Android.

Autor: Diego Poderoso Novo

Tutor: Alvaro Araujo Pinto

Departamento: Ingeniería Electrónica

Tribunal

Presidente:

Vocal:

Secretario:

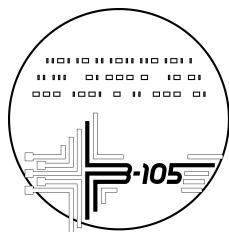
Suplente:

Calificación:

Fecha de lectura: Madrid, a de Junio de 2017

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros de Telecomunicación



Departamento de
Ingeniería
Electrónica



Universidad Politécnica de Madrid



TRABAJO DE FIN DE GRADO

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

DISEÑO DE ESTRATEGIAS PARA LA GESTIÓN DEL AUDIO EN
TELÉFONOS INTELIGENTES SOBRE EL SISTEMA OPERATIVO
ANDROID

DESIGN OF STRATEGIES FOR AUDIO MANAGEMENT IN
SMARTPHONES OVER ANDROID OPERATIVE SYSTEM

Diego Poderoso Novo
Junio 2017

Resumen

Palabras clave:

Summary

Keywords:

Índice

1	Análisis	1
1.1	Evolución del sistema operativo Android	1
1.2	Situación del audio sobre el sistema operativo Android	4
1.2.1	Cambios en las últimas versiones	4
1.2.2	Situación actual sobre Android N	5
2	Entorno de trabajo	7
2.1	Código base sobre el que se desarrolla	7
2.1.1	Estructura del código	7
2.2	Programas de utilidad para el análisis y diseño	8
2.3	Otros recursos utilizados	11
3	Modos de audio	14
3.1	¿Qué son los modos de audio en Android?	14
3.2	Diseño de un nuevo modo de audio para tono de llamada VoIP.	16
3.2.1	Implementación del modo de audio.	16
3.2.2	Pruebas y problemas en su uso.	18
4	Curvas de volumen	20
4.1	¿Qué son las curvas de volumen?	20
4.2	Diferencia entre Android M y Android N	25
	Bibliografía	26
	Acrónimos	27

Índice de figuras

1.1	Capas del audio en Android	5
2.1	Página inicial de QPST.	9
2.2	Página inicial de QACT.	9
2.3	Página inicial de QXDM.	10
2.4	Página inicial de QCAT.	10
2.5	Página inicial de Adobe Audition.	11
2.6	Página inicial de Jenkins.	12
2.7	Mi página de Confluence.	12
2.8	Mis tareas pendientes de JIRA.	13
3.1	Declaración de los modos de audio.	15
3.2	Declaración de los flujos de audio.	16
3.3	Tono de llamada VoIP como MODE_RINGTONE.	17
3.4	Modo por defecto tras tono de llamada VoIP.	17
3.5	Declaración de MODE_RINGTONE_IP.	17
3.6	Trazas Skype con MODE_RINGTONE_IP.	18
3.7	Mensaje de error de Hangouts con MODE_RINGTONE_IP.	19
4.1	Declaración de los dispositivos de salida.	21
4.2	Declaración de las categorías de dispositivos.	21
4.3	Asociación de las categorías de dispositivos.	22
4.4	Definición de un punto de la curva de volumen.	23
4.5	Declaración del control de volumen en la interfaz de usuario.	23
4.6	Comparación de barras de volumen de la interfaz de usuario.	24

Índice de tablas

1.1 Distribución de versiones Android en mayo de 2017	3
---	---

Capítulo 1

Análisis

En este apartado se tratará la evolución del sistema operativo Android a lo largo de su historia, tanto en la implantación en el mercado como en el desarrollo de nuevas funcionalidades [1]. Se busca ofrecer una visión general de las posibilidades que permite desarrollar sobre este sistema operativo. Por otra parte, se pretenden observar los cambios que ha habido con respecto al tratamiento del audio en el sistema con el paso de las diferentes versiones y dar una visión más específica de la situación sobre la versión actual, Android 7.1 Nougat, que es sobre la que se ha desarrollado gran parte de este proyecto.

1.1 Evolución del sistema operativo Android

El sistema operativo Android, pese a lo que muchos puedan creer, no es un desarrollo original de Google Inc., sino que esta adquirió en julio de 2005 una pequeña empresa llamada Android Inc. que se encargaba de desarrollar software para teléfonos móviles. Fue esta empresa, al amparo de Google, la que comenzó a trabajar en un sistema operativo para móviles basado en el kernel de Linux.

El 5 de noviembre de 2007 la Open Handset Alliance, una asociación de compañías de la que Google forma parte y que se dedica a desarrollar estándares abiertos para dispositivos móviles, presentó Android, una plataforma para dispositivos móviles desarrollada sobre la versión 2.6 del kernel de Linux.

La versión 1.0 o también conocida como Apple Pie (las versiones de este sistema van por orden alfabético y tienen siempre el nombre de un postre) fue lanzada el 23 de septiembre de 2008. El 9 de febrero de 2009 fue presentada la versión 1.1, Banana Bread. Ninguna de estas dos versiones llegó a ser comercial.

La primera versión comercial fue la 1.5 Cupcake lanzada en ese mismo año el 30 de abril. Fue en esta versión en la que se incorporaron mejoras como: soporte para Bluetooth, teclado predictivo, y la posibilidad de grabar y reproducir video. Con la versión 1.6 Donut lanzado también en 2009 se incorporan: soporte para resoluciones

Wide Video Graphics Array (WVGA), mejoras en el mercado de aplicaciones de Android, y múltiples mejoras a nivel de interfaz.

Por aquel entonces Android era un sistema muy inestable, al que le faltaba optimización a nivel de hardware, así como soporte para diferentes estándares de conectividad, de ahí que cada pocos meses se presentará una versión nueva. Para tratar de solventar estas carencias se presentaron las versiones 2.0/2.1 Éclair y 2.2 Froyo que añaden soporte para: Bluetooth 2.1, mayores tamaños de pantalla y resoluciones, HTML5 y JavaScript V8, añaden las actualizaciones automáticas para el mercado de aplicaciones, y muchas otras mejoras con el objetivo de optimizar tanto la interfaz de usuario como el sistema operativo a más bajo nivel.

La versión 2.3 Gingerbread fue presentada en diciembre de 2010 y supuso un gran paso adelante en cuanto a estabilidad del sistema se refiere, incorporando: soporte para Near Field Communication (NFC) y para llamadas Voice Over Internet Protocol (VoIP), administrador de energía, de descargas y de tareas, soporte nativo para múltiples cámaras, y nuevos efectos de audio. Cabe destacar que con esta versión Android superó a iOS en cuota de mercado con un 27% y se encontraba sólo por detrás de BlackBerry OS.

Las versiones 3.0/3.1/3.2 Honeycomb fueron lanzadas inicialmente en febrero de 2011 y se centraban únicamente en mejorar la compatibilidad del sistema operativo Android en el formato tablet.

No fue hasta la versión 4.0 Ice Cream Sandwich cuando se unificó el sistema para todos los dispositivos móviles. Entre las mejoras incorporadas tenemos: soporte para la multitarea, la implantación de un único *framework* para facilitar el desarrollo, soporte para que la interfaz pueda ser manejada por la Graphics Processor Unit (GPU), botones virtuales, y un gestor de tráfico de datos de internet. Con esta versión, Android se convierte en el sistema operativo para dispositivos móviles con mayor cuota de mercado.

Durante las siguientes versiones 4.1/4.2/4.3 Jelly Bean destaca la incorporación del asistente Google Now [2]. Ya en octubre de 2013 se presenta Android 4.4 KitKat, que no supone una gran revolución, pero sí que trae la mayor estabilidad al sistema hasta la fecha [3].

Dicha revolución sí que llegó con la siguiente versión, 5.0/5.1 Lollipop, en noviembre de 2014 [4]. A día de hoy se trata de la versión que más cambios ha introducido en el sistema: una completa renovación visual con la incorporación de Material Design que pretende unificar la estética de Android mediante una guía de diseño, modos de prioridad en las notificaciones, funciones de ahorro de energía, incorporación de Bluetooth Low Energy (BLE), compatibilidad con dispositivos de 64 bits, reducción de la latencia de audio, y un largo etcétera. Pero esto también trajo de vuelta la inestabilidad al sistema que no se solventó hasta la versión 6.0 Marshmallow en octubre de 2015 [5]. Además, incorporó: Google Now On Tap, que utiliza al asistente Google Now para proporcionar información de lo que está en ese momento en pantalla, soporte para sensores de huellas, un gestor de permisos mucho más

amplio y un gestor energético inteligente llamado Doze. Con estas versiones, Android empieza a estabilizarse en una cuota de mercado en torno al 87 %, muy lejos del segundo, 12 % de iOS.

Tras esto se llega a las versiones actuales 7.0/7.1 Nougat lanzadas en agosto de 2016 [6]. Como principales mejoras incorporan soporte para multiventana y para realidad virtual con Daydream. Se trata de una versión, que como hizo Android 4.4, viene a estabilizar el sistema operativo de manera definitiva tras los cambios introducidos en las versiones anteriores.

En marzo de este mismo año Google ha presentado la versión de Android 8.0, aún sin nombre, para desarrolladores. Aún no están confirmadas las mejoras definitivas que incorporará esta versión, pero sí que se ha asegurado que seguirá la línea de las últimas versiones.

Finalmente, destacar el que ha sido desde sus inicios uno de los grandes problemas de Android como es la distribución de las distintas versiones en el mercado actual. Esto es un inconveniente para los desarrolladores, tanto de alto nivel, que deben asegurar la compatibilidad de sus aplicaciones para diferentes versiones de la Application Programming Interface (API), como de bajo nivel, que tienen más difícil hacer compatibles sus dispositivos con las actualizaciones del sistema operativo. Actualizar un dispositivo no depende únicamente del fabricante, sino también de que su proveedor del System on Chip (SoC) haga compatible su desarrollo. Debido a esto, la implantación de las nuevas versiones de Android se produce paulatinamente. En la siguiente tabla se puede observar cómo se encuentra la situación actual [7]:

Versión	Nombre	API	Distribución
2.3.3-2.3.7	Gingerbread	10	1 %
4.0.3-4.0.4	Ice Cream Sandwich	15	0.8 %
4.1.x		16	3.2 %
4.2.x	JellyBean	17	4.6 %
4.3.x		18	1.3 %
4.4	KitKat	19	18.8 %
5.0		21	8.7 %
5.1	Lollipop	22	23.3 %
6.0	Marshmallow	23	31.2 %
7.0		24	6.6 %
7.1	Nougat	25	0.5 %

Tabla 1.1: Distribución de versiones Android en mayo de 2017

1.2 Situación del audio sobre el sistema operativo Android

Con respecto a la forma de tratar el audio y las posibilidades que proporciona Android es destacable la estabilidad en la que se encuentra con el paso de las últimas versiones sin grandes novedades en este apartado y mejorando donde debía hacerlo.

1.2.1 Cambios en las últimas versiones

Las primeras versiones de Android desde la primera API publicada, ya incorporaba las funciones básicas para poder realizar llamadas, grabar audios, reproducir música y otros muchos usos convencionales.

Originalmente se tenía clasificado el audio según tres tipos: normal, tono de llamada y en llamada. A partir de la API 11, con Android 2.3, se incorpora el soporte para llamadas VoIP y con él un cuarto modo de audio denominado en comunicación, usado para clasificar el audio durante una llamada VoIP. En la versión más actual, Android 7.1.1, se siguen manteniendo estos cuatro modos de audio. Este tema se aborda más en profundidad en el capítulo 3.

Otro añadido destable tiene lugar en la versión 2.1 donde se añade la diferencia entre la grabación de audio mientras se está grabando un vídeo o si por el contrario se trata únicamente de una grabación de voz. Este añadido se realiza en las denominadas fuentes de audio, entre las que destacan: por defecto, grabación de audio en vídeo, micrófono, voz en llamada, reconocimiento de voz, etc. Como se ha comentado anteriormente, en la API 11 se incluye el soporte para llamadas VoIP y con él también la fuente de audio de voz en comunicación para este caso de uso. En una de las últimas versiones, Android 7.0, se ha incorporado la fuente de audio sin procesar que permite mejorar la latencia de manera considerable.

Con respecto al problema de la latencia en Android, este ha sido uno de los más destacados y que más se ha intentado solucionar desde Android 4.4, portando la API OpenSL de manera parcial. Con la versión 5.0 y 6.0 se consigue reducir el tiempo del rutado de audio por el sistema a través del uso de búferes más pequeños y de una nueva característica para peticiones de audio profesional con latencias en torno a 20 milisegundos, ya existía una característica de baja latencia que conseguía a los 50 milisegundos.

Además de la mejora en la latencia, con la versión 5.0 se incluyeron: el soporte para datos en punto flotante, compatibilidad con frecuencias de muestreo de 96 kHz, ya eran compatibles las de 44.1 y 48 kHz, y para la resolución de 24 bits por muestra, frente a los 16 bits por muestra que ya se soportaban. Por su parte, Android 6.0 además de continuar optimizando la latencia del audio, incorporó compatibilidad con el protocolo Musical Instrument Digital Interface (MIDI).

1.2.2 Situación actual sobre Android N

El salto de Android 6.0 a la versión 7.0 ó 7.1 no ha traído grandes cambios a nivel de audio.

Es destacable, eso sí, lo que es una tendencia en las últimas versiones, que es el uso de ficheros XML para la descripción de la configuración, sustituyendo a ficheros escritos en C++ habitualmente. En relación al audio, se ha incorporado la descripción de las curvas de volumen para los distintos casos de uso en diferentes ficheros XML, lo que permite al desarrollador sustituir dichos ficheros sin necesidad de recompilar el código. Este cambio se aborda más en profundidad en el apartado 5.Curvas de Volumen.

Profundizando algo más sobre la estructura actual del audio en Android, se pueden diferenciar cinco capas [8]: el *framework* a nivel de aplicación, la Java Native Interface (JNI), el *framework* nativo, la Hardware Abstraction Layer (HAL) y el Kernel de Linux. En la siguiente imagen se puede observar dichas capas:

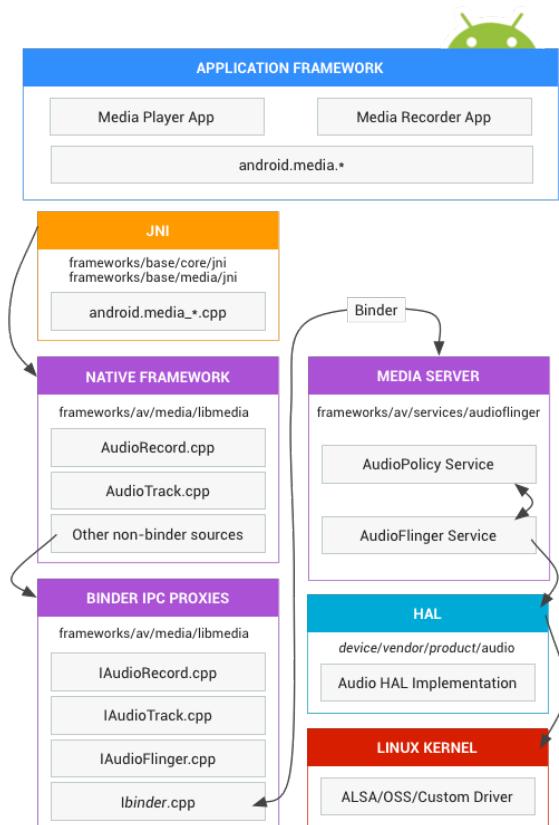


Figura 1.1: Capas del audio en Android.

- *Framework* a nivel de aplicación: es el código desarrollado por el programador de aplicaciones en alto nivel. Utiliza las API para interactuar con el hardware de audio.
- *JNIs*: esta capa se encarga de traducir las peticiones por parte de la aplicación, programada en Java, a peticiones al código nativo de Android, escrito en C y C++.
- *Framework* nativo: es la capa intermedia del código Android. Se compone de tres grandes bloques: el primero se encarga de mandar las distintas órdenes para reproducir o grabar audio; el segundo describe la comunicación entre componentes y se encarga de hacer de interlocutor entre los otros dos bloques; y el tercero se encarga de abrir flujos de datos de entrada o salida, del acceso a los dispositivos, manejo del volumen, prioridades, manejar eventos, etc. En definitiva, esta capa incluye toda la política de audio, cómo se interpreta y por dónde se debe rutaar,

el camino que debe seguir el audio en función de como se quiera interpretar y procesar.

- HALs: define una interfaz estándar que el *framework* nativo utiliza para comunicarse y configurar el hardware. En esta capa se implementan las rutas necesarias para cumplir la política de rutado y se configuran los módulos en función de cómo se ha interpretado el audio en capas superiores. Siendo más específicos, es aquí dónde se decide por cual topología del Application Digital Signal Processing (ADSP) y por cuales elementos hardware, relacionados con el audio, pasa el flujo de datos.
- Kernel de Linux: en esta capa se encuentran los controladores de audio necesarios para el uso del hardware de cada dispositivo. En Android, el controlador de audio utilizado, al igual que en Linux, es la Advanced Linux Sound Architecture (ALSA). Otras posibles opciones son utilizar el Open Sound System (OSS) o un controlador propio.

Por último, aunque en el código base de Android no haya habido cambios significativos, a más bajo nivel Qualcomm, el proveedor de los SoC de los terminales con los que se ha trabajado en este proyecto. Este módulo ha incorporado mejoras en el uso del ADSP, el cual nos permiten un procesado del audio más efectivo y una mayor variedad en los efectos aplicados. El uso del ADSP y las posibilidades que este nos permite se estudian más a fondo en el apartado 6.Rutas del audio.

Capítulo 2

Entorno de trabajo

Durante este capítulo se recopilarán todas las herramientas de interés con las que he trabajado durante este proyecto, con el objetivo que los capítulos posteriores se tenga una visión general de la aplicación que tienen.

2.1 Código base sobre el que se desarrolla

El código base sobre el que se ha desarrollado el proyecto, es el código de Android optimizado por parte de la empresa BQ.

Es destacable que, pese a no introducir BQ muchos cambios sobre el código nativo, el código sin compilar tiene un peso de unos 100 Gigabytes (GB).

Por tanto, en su gran mayoría dicho código proviene de parte de Google, el nativo de Android, y por parte de Qualcomm, los controladores necesarios para el uso del hardware. Respecto a esto último, hay parte de dicho código que viene precompilado por parte de Qualcomm, para evitar posibles modificaciones por parte de los desarrolladores.

2.1.1 Estructura del código

El código Android se estructura de manera muy similar al diagrama de capas visto en la Figura 1.1. Recordar que la primera de las capas hace referencia al código de la aplicación que este siendo utilizada, y por tanto no se incluye dicha capa en el código.

Se tienen diferentes carpetas que equivalen a cada una de las capas:

- *frameworks*: incluye tanto las capas superiores como las intermedias. Por un lado, se encuentra la JNI programada en Java en la carpeta base. Por otro lado, la otra carpeta que nos es relevante en este proyecto es av, la cual incluye la política de audio en su totalidad.

- hardware: incluye la capa superior de la HAL. En ella encontramos la carpeta qcom cuyo desarrollo es parte de Qualcomm y en la cual se incluye toda la configuración del hardware en función de los distintos casos de uso. Incluye, junto con el *framework*, la mayor parte del código que se ha modificado durante la realización de este proyecto.
- vendor: incluye la capa inferior de la HAL. En ella se encuentran todos los controladores propietarios de Qualcomm.
- kernel: como ya se ha comentado, Android está basado en el kernel de Linux, y es aquí dónde dicha modificación se encuentra.
- device: aquí se incluyen ficheros propios de cada dispositivo, como puede ser el de la configuración de las curvas de volumen, o los ficheros de configuración de su ADSP.

Por supuesto, el código es mucho más extenso, pero con respecto al tratamiento del audio en el sistema operativo Android, estas serían las carpetas más importantes.

Debido a esta extensión, la compilación local sería inviable si cada vez que quisésemos compilar hubiese que hacerlo de todo el código. Por esa razón, se utiliza la herramienta Make, una herramienta de gestión de dependencias existentes en el código fuente que facilita su compilación. Para que esta herramienta sea útil es necesaria la presencia de ficheros Makefiles, los cuales establecen dichas dependencias entre ficheros. De esta manera, si queremos compilar un conjunto limitado de ficheros, habrá que buscar el Makefile del que dependan y ejecutar la herramienta Make desde ese directorio, proporcionando librerías en lugar de una nueva versión del textitfirmware.

2.2 Programas de utilidad para el análisis y diseño

El fabricante del SoC de los dispositivos con los que se ha trabajado es Qualcomm. Esta empresa proporciona software propio dedicado al análisis y configuración del hardware que fabrican. De esta manera, relacionados con el tema del audio, se han utilizado los siguientes programas:

- Qualcomm Product Support Tool (QPST): proporciona un servidor automático entre un lenguaje, u otro programa, y un puerto del ordenador, al que conectamos el dispositivo móvil. Nos permite analizar y configurar el dispositivo en tiempo real. Es compatible con el resto de software de Qualcomm.

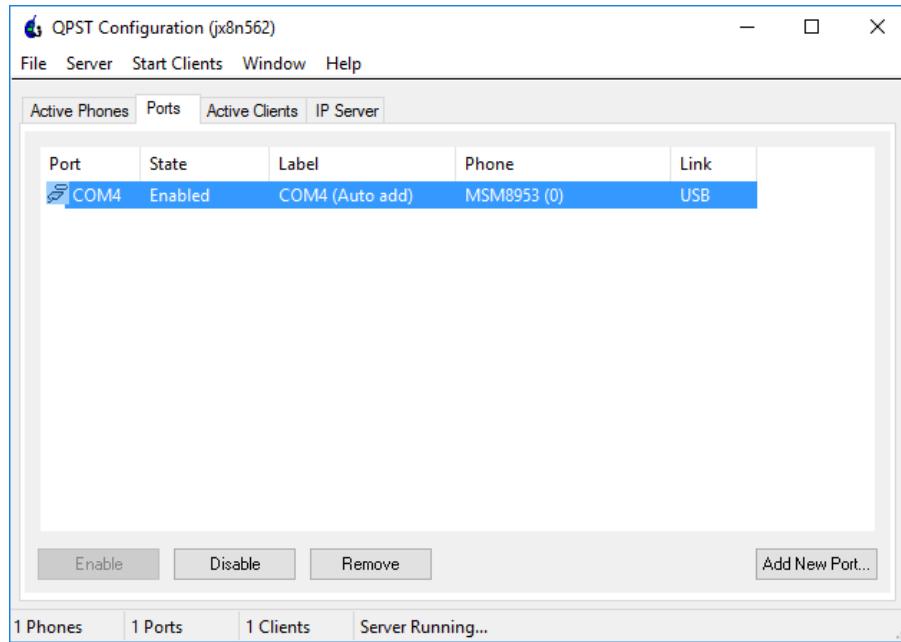


Figura 2.1: Página inicial de QPST.

- Qualcomm Audio Calibration Tool (QACT): proporciona una interfaz gráfica que nos permite visualizar y modificar la calibración de las diferentes topologías, los distintos caminos que puede seguir el audio, que existen en el ADSP. Tiene dos modos de uso, un modo sin conexión, en el que los cambios se guardan en ficheros de configuración que posteriormente hay que cargar en el teléfono, y un modo en tiempo real, en el cual podemos ver y calibrar la topología que se está aplicando en cada momento. Para que sea posible utilizar el modo en tiempo real es necesario que el dispositivo esté conectado por medio del QPST.

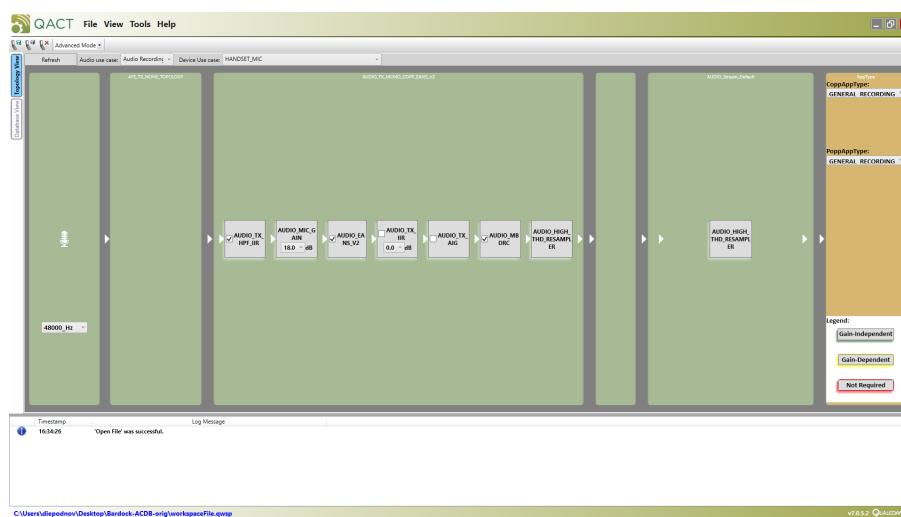


Figura 2.2: Página inicial de QACT.

- Qualcomm Extensible Diagnostic Monitor (QXDM): aunque su uso en el apartado del audio está muy limitado, permite obtener flujos de datos y trazas informativas en diferentes puntos del camino que sigue el audio a bajo nivel. Estos conjuntos de datos se deberán procesar posteriormente, pues el formato con el que te proporciona la información es propietario de Qualcomm.

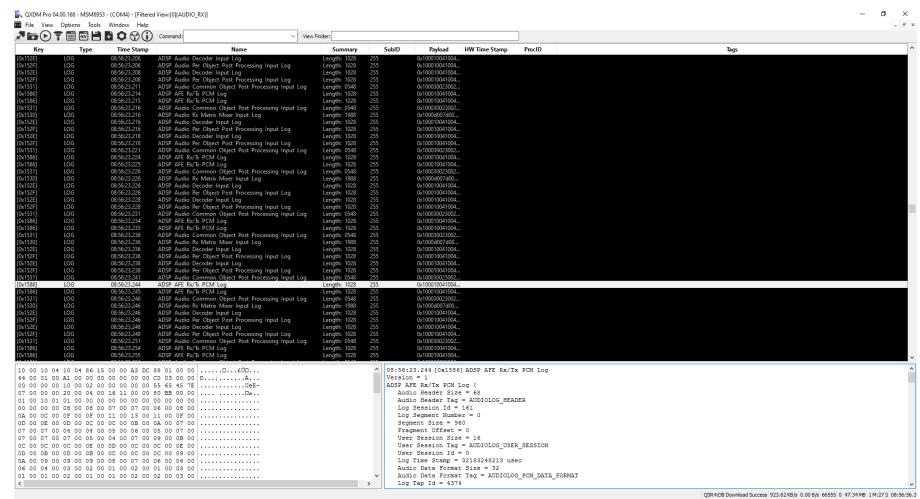


Figura 2.3: Página inicial de QXDM.

- Qualcomm CDMA Analysis Toolkit (QCAT): como se ha comentado en el programa anterior, el formato obtenido únicamente es interpretable por parte de Qualcomm, que nos proporciona este software con el objetivo de poder obtener ficheros de audio analizables y reproducibles a través de los datos obtenidos con QXDM. Además, es un programa de análisis estadístico, proporcionando información del número de trazas obtenidas en cada punto, el tiempo entre trazas, etc.

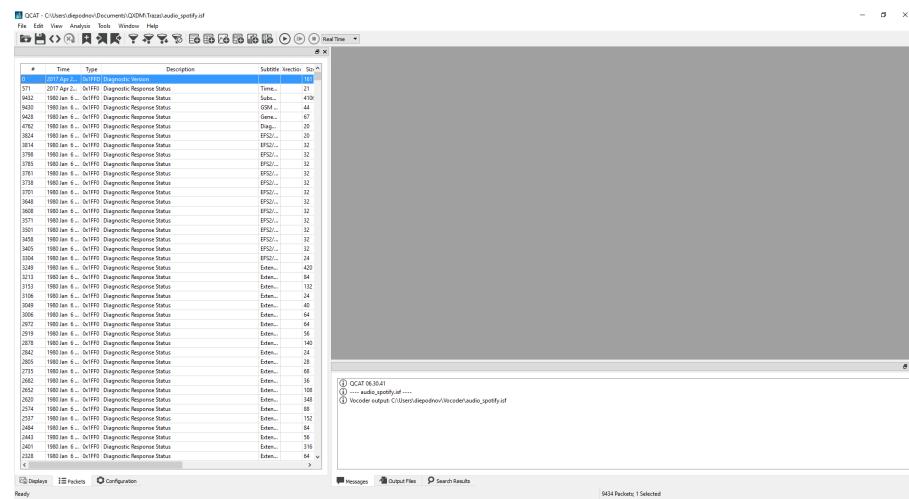


Figura 2.4: Página inicial de QCAT.

- Adobe Audition: es el programa utilizado por parte del departamento de audio de la empresa BQ, para el análisis de los distintos ficheros de audio. Es propiedad de Adobe. En el caso de este proyecto, se ha utilizado para analizar los ficheros de audio obtenido mediante el programa QCAT. Destacar que se trata de un programa con un gran número de herramientas, sobre todo dedicadas a la edición de audio profesional. Sin embargo, durante la realización de este proyecto se ha usado con un enfoque más analítico.

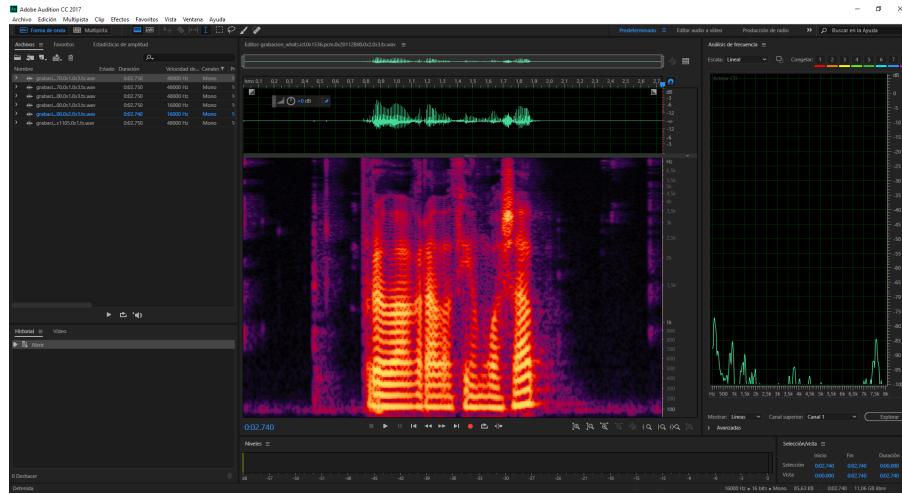


Figura 2.5: Página inicial de Adobe Audition.

- Android Debug Bridge (ADB): no es un programa de Qualcomm, sino que es la herramienta que proporciona Google para poder conectarnos al de manera remota, mediante comandos desde el ordenador. Esta herramienta proporciona diferentes acciones en dispositivos, como la instalación de librerías y la depuración mediante trazas.
- *Fastboot*: tiene un uso similar al ADB. Sirve para cambiar el *firmware* del dispositivo desde un modo dedicado para ello. Los cambios suelen ser completos, no como en el ADB donde sobre un mismo *firmware* se cambian librerías.

2.3 Otros recursos utilizados

Como ya se ha comentado, este proyecto está al amparo de la empresa Mundo Reader S.L., cuyo nombre comercial es BQ, la cual trabaja con distintas herramientas en su día a día. Con el objetivo de familiarizarse al máximo con la metodología que allí se sigue, se han utilizado dichos recursos durante la realización de este proyecto. Cuales son y el uso que se le han dado se explica a continuación:

- Jenkins: se trata de un servidor de integración continua, gratuito y abierto. Permite a los desarrolladores compilar sus proyectos en servidores remotos, de una manera más rápida que si lo hicieran en local. Se basa en un sistema de

colas, para mantener en espera una compilación hasta que un servidor quede libre.

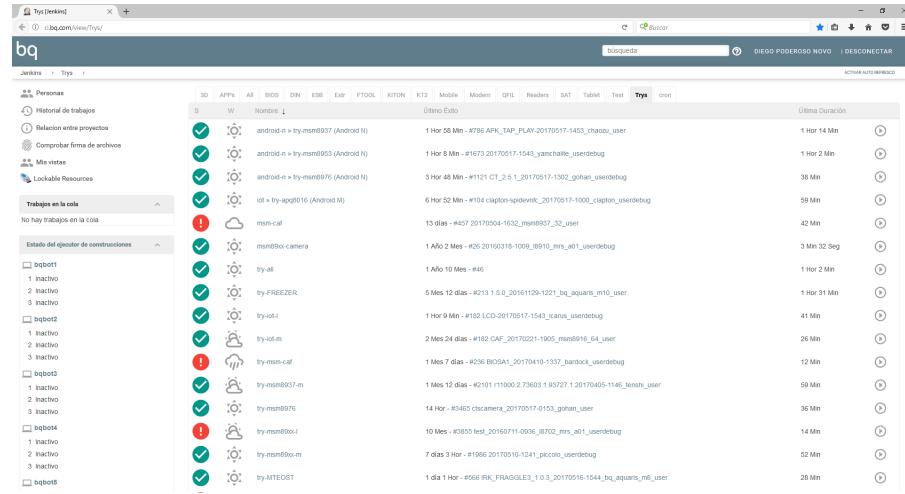


Figura 2.6: Página inicial de Jenkins.

- Gerrit: es un repositorio donde se desarrollan los proyectos de automatización. Los desarrolladores publican su código que tiene que ser aprobados por el resto de compañeros y pasar pruebas de compilación ejecutadas en Jenkins.
- Confluence: es un software de colaboración de contenidos, propiedad de Atlassian. De esta manera, cada vez que desarrolles alguna nueva funcionalidad o analizas algún punto de interés, se debe subir a esta plataforma una breve documentación de lo realizado que es accesible y editable por parte del resto de compañeros.

Intern	Acronym	Label	Start
Diego.Poderoso Novo	2016-Audio	[C02016-Audio]	11-03-2016

Categoría	Resumen	T	Creada	Actualizada	Fecha De Entrega	Responsable	Informante	P	Estatus	Resolución
CATEDRAIBQ-1730	Ver trazos de las asociaciones del mixer	FW Task	may 17, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Major	In progress	Unresolved
CATEDRAIBQ-1724	Mixer	Epic	may 17, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Major	Open	Unresolved
CATEDRAIBQ-1721	Escribir apartado de entorno	Documentation	may 15, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Major	In progress	Unresolved
CATEDRAIBQ-1718	Handset voice call Rx is mute	Bug	may 12, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Blocker	In progress	Unresolved
CATEDRAIBQ-1716	Estudio del fichero mixer.mml	FW Task	may 08, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Major	In progress	Unresolved
CATEDRAIBQ-1715	Buscar documentación del MSM8993-Audio Codec	Documentation	may 08, 2017	may 17, 2017		Diego Poderoso Novo	Ramiro Urtiza	Major	In progress	Unresolved

Figura 2.7: Mi página de Confluence.

- JIRA: permite planificar y supervisar las distintas tareas de un departamento, así como indicar las horas dedicadas a cada tarea por parte de cada trabajador.

El objetivo es poder realizar un seguimiento ordenado sobre el estado en que se encuentra cada una de las tareas, si están finalizadas o si ha surgido algún problema, así como saber quienes están al cargo de dicha realización, facilitando la interacción entre ellos en el caso de no pertenecer al mismo grupo de trabajo. También es propiedad de Atlassian, y la integración con Confluence es total.

The screenshot shows a JIRA interface with the following details:

- Project:** CATEDRABQ-1718
- Title:** Handset voice call RX is mute
- Status:** IN PROGRESS
- Assignee:** Diego Poderoso Novo
- Reporter:** Ramiro Utrilla
- Created:** Friday 9:32 AM
- Updated:** Today 1:52 PM
- Time Tracking:**
 - Estimated: 1d 4h
 - Remaining: 7h
 - Logged: 5h
- Agile:** Sprint 14 ends 29May17

Figura 2.8: Mis tareas pendientes de JIRA.

Capítulo 3

Modos de audio

En este capítulo se explicarán que son y para qué se utilizan los distintos modos de audio existentes en el sistema operativo Android. Además, se explicará la implementación de nuevos modos de audio y se evaluará su posible implementación en dispositivos comerciales.

3.1 ¿Qué son los modos de audio en Android?

El sistema operativo Android se divide en distintas capas. El programador de alto nivel, el desarrollador de aplicaciones, únicamente tiene acceso a la capa superior del sistema. Por tanto, debe tener recursos a su disposición para que en las capas inferiores el audio se interprete como él quiera.

Con este fin existen las API, que proporcionan al programador estos recursos. Los modos de audio se incluyen en dichas API desde la primera versión. El objetivo de estos modos es clasificar el audio en función de su uso. Los modos de audio existentes en la API 25 [9], son:

- Modo normal: en el código se declara como MODE_NORMAL. Es el modo de audio por defecto. Es el que se utiliza en la reproducción de contenido multimedia, sonidos del sistema o alarmas. Además, es el utilizado por defecto en tonos de llamada VoIP.
- Modo tono de llamada: en el código se declara como MODE_RINGTONE. Es el utilizado cuando se reproduce un tono de llamada tradicional, Global System for Mobile communications (GSM).
- Modo en llamada: en el código se declara como MODE_IN_CALL. Es el utilizado para indicar que se está en una llamada GSM.
- Modo en comunicación: en el código se declara como MODE_IN_COMMUNICATION. Es el utilizado para indicar que se está en una llamada VoIP o en una video-llamada. Es el único modo que no está presente en la primera versión de la

API, se incorpora en la versión 11.

- Modo actual: en el código se declara como MODE_CURRENT. Sirve para referirse al modo actual de audio.
- Modo no válido: en el código se declara como MODE_INVALID. Sirve para clasificar como no válido el audio.

Los dos últimos modos de audio no se utilizan habitualmente, sino que suelen indicar errores.

Estos modos de audio se encuentran declarados en el fichero audio.h del sistema.

```
typedef enum {
    AUDIO_MODE_INVALID          = -2,
    AUDIO_MODE_CURRENT          = -1,
    AUDIO_MODE_NORMAL           = 0,
    AUDIO_MODE_RINGTONE         = 1,
    AUDIO_MODE_IN_CALL          = 2,
    AUDIO_MODE_IN_COMMUNICATION = 3,

    AUDIO_MODE_CNT,
    AUDIO_MODE_MAX              = AUDIO_MODE_CNT - 1,
} audio_mode_t;
```

Figura 3.1: Declaración de los modos de audio.

El uso de un modo de audio u otro depende del desarrollador de la aplicación. A lo largo del sistema, únicamente se realizan las siguientes tareas:

1. Comprobación del modo de audio seleccionado por la aplicación mediante el uso de métodos de obtención. De esta manera, en cada una de las capas se sabe que modo de audio se está usando y se aplicará la política de audio correspondiente.
2. Aplicación de la política de audio a casos de uso particulares. Por ejemplo, si se termina de aplicar el modo de audio en llamada, de manera automática se pasa al modo de audio normal.
3. Elección del uso de los modos de audio por parte del desarrollador de bajo nivel. Se puede hacer que el sistema interprete un caso de uso , con un modo de audio elegido por la aplicación, como un modo de audio distinto, que puede ser cualquiera de los otros o uno propio del desarrollador. En el siguiente apartado se explica este uso más en profundidad.

3.2 Diseño de un nuevo modo de audio para tono de llamada VoIP.

Esta implementación se ha realizado sobre la versión de Android 6.0, Marshmallow. Esto se debe a que fue la primera parte del proyecto, que tuvo lugar antes de que se actualizasen los dispositivos a la versión 7.0, Nougat. Pese a ello, su implementación sobre la nueva versión del sistema operativo no varía.

Las llamadas VoIP tienen un modo de audio propio, en comunicación. Sin embargo, los tonos de llamada de estas llamadas VoIP no tienen un modo de audio dedicado, sino que usan el modo de audio normal.

3.2.1 Implementación del modo de audio.

El primer desarrollo ha consistido en la utilización de un modo de los ya existentes. De esta manera, en vez de usar el modo de audio por defecto se utiliza el modo tono de llamada. Esto implica que se interprete de la misma manera un tono de llamada GSM y uno VoIP.

Saber cuando se trata de un tono de llamada VoIP es posible debido a la existencia de otra forma de clasificación del audio en el código. No solo existen los modos de audio, sino que también existen los tipos de flujos [9]. La declaración de los tipos de flujos se realiza también en el fichero audio.h:

```
typedef enum {
    AUDIO_STREAM_DEFAULT      = -1,
    AUDIO_STREAM_MIN          = 0,
    AUDIO_STREAM_VOICE_CALL   = 0,
    AUDIO_STREAM_SYSTEM        = 1,
    AUDIO_STREAM_RING          = 2,
    AUDIO_STREAM_MUSIC         = 3,
    AUDIO_STREAM_ALARM         = 4,
    AUDIO_STREAM_NOTIFICATION  = 5,
    AUDIO_STREAM_BLUETOOTH_SCO = 6,
    AUDIO_STREAM_ENFORCED_AUDIBLE = 7,
    AUDIO_STREAM_DTMF           = 8,
    AUDIO_STREAM_TTS            = 9,
    AUDIO_STREAM_ACCESSIBILITY = 10,
    AUDIO_STREAM_REROUTING     = 11,
    AUDIO_STREAM_PATCH          = 12,
    AUDIO_STREAM_PUBLIC_CNT    = AUDIO_STREAM_TTS + 1,
    AUDIO_STREAM_FOR_POLICY_CNT = AUDIO_STREAM_PATCH,
    AUDIO_STREAM_CNT             = AUDIO_STREAM_PATCH + 1,
} audio_stream_type_t;
```

Figura 3.2: Declaración de los flujos de audio.

Debido a la extensión de su código, la clasificación del audio se realiza de distintas maneras. Los tipos de flujos de datos permiten seleccionar la curva de volumen concreta, esto se aborda con mayor profundidad en el apartado 5.Curvas de Volumen.

Una vez se han introducido los flujos de datos, se pueden aprovechar estos para realizar la comparación. En el fichero MediaPlayer.java, que se utiliza para controlar la reproducción de audio y vídeo, cuando se inicia la reproducción se comprueba si el flujo de datos está clasificado como timbre, AUDIO_STREAM_RING.

De esta manera saber si se trata de un tono de llamada VoIP se implementa de la siguiente manera:

```
public void start() throws IllegalStateException {
    baseStart();
    if(getAudioStreamType() == mAudioManager.STREAM_RING && mAudioManager !=null){
        mAudioManager.setMode(MODE_RINGTONE);
    }
    stayAwake(true);
    _start();
}
```

Figura 3.3: Tono de llamada VoIP como MODE_RINGTONE.

Pero de esta manera, al acabar de sonar el tono de llamada pueden darse dos situaciones. Si se contesta la llamada el modo de audio se cambiará a MODE_IN_COMMUNICATION. Pero si no se contesta, el modo que se quedaría es MODE_RINGTONE. Para que vuelva al modo por defecto, en el método que para la reproducción, se debe volver a fijar dicho modo:

```
public void stop() throws IllegalStateException {
    stayAwake(false);
    if(mAudioManager !=null && getAudioStreamType() == mAudioManager.STREAM_RING &&
        mAudioManager.getMode()!=MODE_IN_CALL && mAudioManager.getMode()!= MODE_IN_COMMUNICATION) {
        mAudioManager.setMode(MODE_NORMAL);
    }
    _stop();
}
```

Figura 3.4: Modo por defecto tras tono de llamada VoIP.

El siguiente paso en el desarrollo ha sido la implementación de un modo propio. Para ello, es necesario declarar un nuevo modo de audio en las distintas capas del *framework*.

Se ha creado un nuevo modo de audio denominado MODE_RINGTONE_IP. Hay que declararlo tanto a nivel de JNI como de *framework* nativo.

AUDIO_MODE_RINGTONE_IP = 4,	public static final int MODE_RINGTONE_IP = AudioSystem.MODE_RINGTONE_IP;
(a) audio.h	(b) AudioManager.java

Figura 3.5: Declaración de MODE_RINGTONE_IP.

Tras esto, únicamente es necesario sustituir MODE_RINGTONE por MODE_RINGTONE_IP en la implementación realizada en MediaPlayer.java.

A la hora de compilar la nueva implementación, es recomendable realizar una compilación completa del código. Esto se debe a que se han modificado diferentes partes. Por un lado, el fichero audio.h se encuentra en el directorio de sistema. Por otro, el fichero AudioManager.java y MediaPlayer.java se encuentran en el JNI.

Para realizar este proceso de una manera más rápida, se ha hecho uso de la herramienta Jenkins, añadiendo las modificaciones en el repositorio de Gerrit. Más información sobre la función de estas herramientas en 2.3.

3.2.2 Pruebas y problemas en su uso.

Una vez modificado el *firmware* del dispositivo, se ha procedido a comprobar si se realiza de manera correcta la asociación del nuevo modo de audio.

Se han realizado dos pruebas:

1. Llamada VoIP con la aplicación de Skype: los resultados son satisfactorios. Se aplica el modo de audio MODE_RINGTONE_IP sin ningún tipo de problema. En la siguiente imagen se observa una traza del dispositivo que indica que modo de audio se está aplicando:

```
audio_hw_primary: adev_set_mode: mode 0
audio_hw_primary: adev_set_mode: mode 1
audio_hw_primary: adev_set_mode: mode 4
audio_hw_primary: adev_set_mode: mode 0
audio_hw_primary: adev_set_mode: mode 3
audio_hw_primary: adev_set_mode: mode 0
```

Figura 3.6: Trazas Skype con MODE_RINGTONE_IP.

Dichas trazas muestran que se está en el modo por defecto, el cero. Llega una llamada de Skype y se pone el modo de tono de llamada VoIP. Una vez que se contesta, se pasa al modo en comunicación. Finalmente, al colgar se vuelve al modo por defecto.

2. Llamada VoIP con la aplicación de Hangouts: en este caso los resultados no han sido los esperados. Cuando se realiza la llamada y se establece el nuevo modo de audio, la aplicación se cierra automáticamente. La captura de pantalla del dispositivo, con el mensaje de error, se puede observar a continuación:

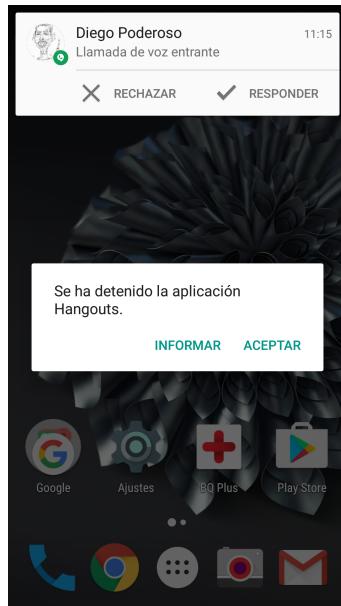


Figura 3.7: Mensaje de error de Hangouts con MODE_RINGTONE_IP.

Finalmente, esta implementación no se ha llevado al producto final. Esto se ha debido a los problemas surgidos con la aplicación de Hangouts. Cabe destacar, que esta aplicación está diseñada por Google y por tanto es un indicativo de la incompatibilidad de crear nuevos modos de audio.

Como conclusiones de este apartado sacamos que, pese a su fácil implementación, no es conveniente el uso de modos de audio propios. Esto tiene sentido, ya que los modos de audio se incluyen en las API para los desarrolladores de alto nivel. Y si se modifican dichas API, esto será una fuente importante de futuras incompatibilidades.

Capítulo 4

Curvas de volumen

En este capítulo se explicarán el funcionamiento de las curvas de volumen en Android. Además, se abordarán los cambios que han tenido lugar con el cambio de versión, de 6.0 Marshmallow a 7.0 Nougat, implementándose y realizando los ajustes necesarios de dichas curvas.

4.1 ¿Qué son las curvas de volumen?

Las curvas de volumen marcan como se atenuará o aumentará el audio según el usuario modifique el volumen de su dispositivo.

Existen múltiples curvas de volumen, una por cada tipo de audio que vaya a reproducirse por cada clase de dispositivo. Esto quiere decir, que habrá una curva de volumen que controle el volumen de las notificaciones cuando suenen por el altavoz, y habrá otra que lo controle cuando suenen por los auriculares.

Para poder entender dicha clasificación es necesario entender los distintos flujos de audio que puede haber en el sistema, ya comentados en la figura 3.2. Además, es necesario comprender los tipos de dispositivos que requieren curvas de volumen propias. Los dispositivos que tendrán relación con las curvas de volumen son los de salida, no los de entrada, ya que son los que reproducen el audio. La declaración de los distintos tipos de dispositivos tiene lugar en el fichero audio.h del sistema:

```

AUDIO_DEVICE_OUT_EARPIECE           = 0x1,
AUDIO_DEVICE_OUT_SPEAKER            = 0x2,
AUDIO_DEVICE_OUT_WIRED_HEADSET      = 0x4,
AUDIO_DEVICE_OUT_WIRED_HEADPHONE    = 0x8,
AUDIO_DEVICE_OUT_BLUETOOTH_SCO      = 0x10,
AUDIO_DEVICE_OUT_BLUETOOTH_SCO_HEADSET = 0x20,
AUDIO_DEVICE_OUT_BLUETOOTH_SCO_CARKIT = 0x40,
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP     = 0x80,
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES = 0x100,
AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER = 0x200,
AUDIO_DEVICE_OUT_AUX_DIGITAL       = 0x400,
AUDIO_DEVICE_OUT_HDMI               = AUDIO_DEVICE_OUT_AUX_DIGITAL,
AUDIO_DEVICE_OUT_ANLG_DOCK_HEADSET   = 0x800,
AUDIO_DEVICE_OUT_DGTL_DOCK_HEADSET   = 0x1000,
AUDIO_DEVICE_OUT_USB_ACCESSORY     = 0x2000,
AUDIO_DEVICE_OUT_USB_DEVICE        = 0x4000,
AUDIO_DEVICE_OUT_REMOTE_SUBMIX      = 0x8000,
AUDIO_DEVICE_OUT_TELEPHONY_TX       = 0x10000,
AUDIO_DEVICE_OUT_LINE               = 0x20000,
AUDIO_DEVICE_OUT_HDMI_ARC          = 0x40000,
AUDIO_DEVICE_OUT_SPDIF              = 0x80000,
AUDIO_DEVICE_OUT_FM                 = 0x100000,
AUDIO_DEVICE_OUT_AUX_LINE          = 0x200000,
AUDIO_DEVICE_OUT_SPEAKER_SAFE      = 0x400000,
AUDIO_DEVICE_OUT_IP                 = 0x800000,
AUDIO_DEVICE_OUT_BUS                = 0x1000000,
AUDIO_DEVICE_OUT_PROXY              = 0x2000000,

```

Figura 4.1: Declaración de los dispositivos de salida.

Como no sería mantenible tener curvas de volumen para cada uno de estos dispositivos, pues son demasiados, estos se agrupan en categorías. De esta manera, se tendrá una curva de volumen para cada par de valores, en función del flujo de audio y de la categoría del dispositivo. La declaración de las cuatro categorías de dispositivos se realiza en el fichero Volume.h, ya orientado plenamente al control del audio:

```

enum device_category {
    DEVICE_CATEGORY_HEADSET,
    DEVICE_CATEGORY_SPEAKER,
    DEVICE_CATEGORY_EARPIECE,
    DEVICE_CATEGORY_EXT_MEDIA,
    DEVICE_CATEGORY_CNT
};

```

Figura 4.2: Declaración de las categorías de dispositivos.

También en este fichero se lleva a cabo la asignación de los distintos dispositivos de salida a cada una de las posibles categorías, mediante el siguiente método:

```

static device_category getDeviceCategory(audio_devices_t device)
{
    switch(getDeviceForVolume(device)) {
        case AUDIO_DEVICE_OUT_EARPIECE:
            return DEVICE_CATEGORY_EARPIECE;
        case AUDIO_DEVICE_OUT_WIRED_HEADSET:
        case AUDIO_DEVICE_OUT_WIRED_HEADPHONE:
        case AUDIO_DEVICE_OUT_BLUETOOTH_SCO:
        case AUDIO_DEVICE_OUT_BLUETOOTH_SCO_HEADSET:
        case AUDIO_DEVICE_OUT_BLUETOOTH_A2DP:
        case AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_HEADPHONES:
            return DEVICE_CATEGORY_HEADSET;
        case AUDIO_DEVICE_OUT_LINE:
        case AUDIO_DEVICE_OUT_AUX_DIGITAL:
            return DEVICE_CATEGORY_EXT_MEDIA;
        case AUDIO_DEVICE_OUT_SPEAKER:
        case AUDIO_DEVICE_OUT_BLUETOOTH_SCO_CARKIT:
        case AUDIO_DEVICE_OUT_BLUETOOTH_A2DP_SPEAKER:
        case AUDIO_DEVICE_OUT_USB_ACCESSORY:
        case AUDIO_DEVICE_OUT_USB_DEVICE:
        case AUDIO_DEVICE_OUT_REMOTE_SUBMIX:
        default:
            return DEVICE_CATEGORY_SPEAKER;
    }
}

```

Figura 4.3: Asociación de las categorías de dispositivos.

Los distintos tipos de dispositivos quedan agrupados en las siguientes categorías:

- Auricular: únicamente se incluye en esta categoría el auricular del teléfono.
- Cascos: en esta categoría se incluyen tanto los cascos conectados por cable, mediante un conector de audio analógico *jack* de 3.5mm, como los que se conectan de manera inalámbrica, según la especificación Bluetooth.
- Dispositivos externos: se incluyen aquellos que se conectan a través del puerto Universal Serial Bus (USB), mediante el protocolo High-Definition Multimedia Interface (HDMI).
- Altavoz: se incluyen tanto el altavoz propio del teléfono, altavoces conectados mediante Bluetooth, los sistemas de conexión Bluetooth de los coches, dispositivos conectados por USB mediante el protocolo MIDI, y dispositivos conectados de manera inalámbrica mediante Wireless Fidelity (WiFi).

Una vez se ha explicado como se clasifican las distintas curvas de volumen, falta explicar como se definen.

Una curva de volumen, en el sistema operativo Android, está compuesta por un conjunto de puntos. Estos puntos deben seguir la siguiente estructura:

```

class VolumeCurvePoint
{
public:
    int mIndex;
    float mDBAttenuation;
};

```

Figura 4.4: Definición de un punto de la curva de volumen.

En su definición podemos observar que los puntos se definen con dos valores. Por un lado, tenemos el índice el cual nos marca la posición del punto en el eje x. Por otro lado, tenemos la atenuación en decibelios (dB) que implicará situar el volumen en ese punto, dicha atenuación se corresponde con el eje y.

La manera que tiene el sistema de definir una curva, a raíz de un conjunto de puntos configurables por parte del desarrollador, es mediante el uso de la interpolación lineal entre puntos.

La interpolación lineal consiste en tratar los segmentos definidos entre dos puntos como una recta. De esta manera, si nuestra curva tiene cuatro puntos, tendremos un total de tres posibles segmentos. En estos segmentos, la atenuación aumenta o disminuye de manera lineal. La pendiente de la recta dependerá de lo cerca que se encuentren los puntos que definen el segmento, según el primer parámetro de su definición, y de la diferencia de atenuación que haya entre ellos, marcada por el segundo parámetro.

Pero, como ya se ha comentado antes, el volumen lo marca el usuario. La relación, entre las curvas definidas en el sistema y la interfaz de volumen que se encuentra el usuario, se encuentra definida en la JNI. La interfaz de usuario proporciona una barra de sonido para modificar el volumen. Dicha barra está definida mediante un número diferente de puntos en el fichero AudioService.java como se muestra a continuación:

<pre> private static int[] MAX_STREAM_VOLUME = new int[] { 10, // STREAM_VOICE_CALL 7, // STREAM_SYSTEM 7, // STREAM_RING 25, // STREAM_MUSIC 7, // STREAM_ALARM 7, // STREAM_NOTIFICATION 15, // STREAM_BLUETOOTH_SCO 7, // STREAM_SYSTEM_ENFORCED 7, // STREAM_DTMF 15 // STREAM_TTS }; </pre>	<pre> private static int[] MIN_STREAM_VOLUME = new int[] { 1, // STREAM_VOICE_CALL 0, // STREAM_SYSTEM 0, // STREAM_RING 0, // STREAM_MUSIC 0, // STREAM_ALARM 0, // STREAM_NOTIFICATION 0, // STREAM_BLUETOOTH_SCO 0, // STREAM_SYSTEM_ENFORCED 0, // STREAM_DTMF 0 // STREAM_TTS }; </pre>
--	--

(a) Volumen máximo.

(b) Volumen mínimo.

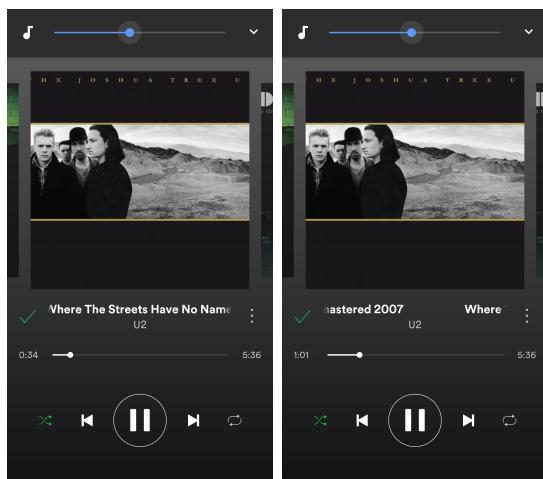
Figura 4.5: Declaración del control de volumen en la interfaz de usuario.

De esta manera se define para cada tipo de flujo de audio una barra de control del

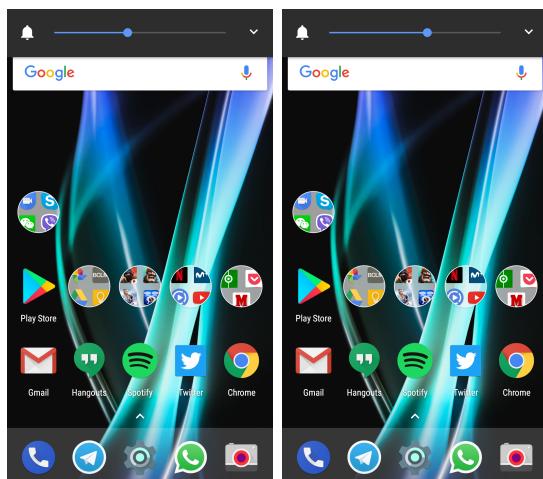
volumen. En la que se empieza en cero, o uno en el caso de las llamadas de voz para que no se puedan silenciar, y se divide en tantos saltos como el volumen máximo indique.

Por ejemplo, en el flujo de música, que sirve para cualquier reproducción multimedia, se define un volumen mínimo de cero y se puede aumentar hasta un total de veinticinco veces.

La interfaz de las barras de volumen de cada uno de los flujos son iguales en longitud. Por tanto, cuanto mayor sea la distancia entre el volumen mínimo y máximo, menor será el cambio de la barra al subir o bajar el volumen. A continuación se muestra una captura para comparar el aumento de la barra de volumen que controla la música y la que controla los sonidos del sistema, al pulsar una vez en el botón de subir volumen:



(a) Volumen de la música original. (b) Volumen de la música aumentado.



(c) Volumen del sistema original. (d) Volumen del sistema aumentado.

Figura 4.6: Comparación de barras de volumen de la interfaz de usuario.

Como se puede observar, el cambio en la barra de usuario al aumentar el volumen del sistema es mucho mayor al cambio al aumentar el de la música. Esto implicará que el salto en las curvas definidas en el sistema también será mayor.

La relación, entre la barra de control de la interfaz de usuario y las curvas de volumen, a cambiado ligeramente con el cambio de versión de Android M a N. Las implicaciones de estos cambios, así como la relación anteriormente comentada, se explicarán en el siguiente apartado.

4.2 Diferencia entre Android M y Android N

Bibliografía

- [1] Ron Amadeo. The (updated) history of android. <https://arstechnica.com/gadgets/2016/10/building-android-a-40000-word-history-of-googles-mobile-os/>. Accedido: 2017-05-04.
- [2] Google. Jelly Bean 4.3. https://www.android.com/intl/en_us/versions/jelly-bean-4-3/. Accedido: 2017-05-04.
- [3] Google. KitKat 4.4. https://www.android.com/intl/en_us/versions/kitkat-4-4/. Accedido: 2017-05-04.
- [4] Google. Android 5.0, Lollipop. https://www.android.com/intl/en_us/versions/lollipop-5-0/. Accedido: 2017-05-04.
- [5] Google. Android 6.0, Marshmallow. https://www.android.com/intl/en_us/versions/marshmallow-6-0/. Accedido: 2017-05-04.
- [6] Google. Android 7.0 Nougat. https://www.android.com/intl/en_us/versions/nougat-7-0/. Accedido: 2017-05-04.
- [7] Android Developers. Dashboards. <https://developer.android.com/about/dashboards/index.html>. Accedido: 2017-05-04.
- [8] Android Source. Audio. <https://source.android.com/devices/audio/>. Accedido: 2016-12-15.
- [9] Android Developers. AudioManager API 25. <https://developer.android.com/reference/android/media/AudioManager.html>. Accedido: 2017-05-19.

Acrónimos

ADB Android Debug Bridge. 11

ADSP Application Digital Signal Processing. 6, 8, 9

ALSA Advanced Linux Sound Architecture. 6

API Application Programming Interface. 3–5, 14, 15, 19

BLE Bluetooth Low Energy. 2

dB decibelios. 23

GB Gigabytes. 7

GPU Graphics Processor Unit. 2

GSM Global System for Mobile communications. 14, 16

HAL Hardware Abstraction Layer. 5, 6, 8

HDMI High-Definition Multimedia Interface. 22

JNI Java Native Interface. 5, 7, 17, 18, 23

MIDI Musical Instrument Digital Interface. 4, 22

NFC Near Field Communication. 2

OSS Open Sound System. 6

QACT Qualcomm Audio Calibration Tool. 9

QCAT Qualcomm CDMA Analysis Toolkit. 10, 11

QPST Qualcomm Product Support Tool. 8, 9

QXDM Qualcomm Extensible Diagnostic Monitor. 10

SoC System on Chip. 3, 6, 8

USB Universal Serial Bus. 22

VoIP Voice Over Internet Protocol. 2, 4, 14, 16–18

WiFi Wireless Fidelity. 22

WVGA Wide Video Graphics Array. 2