Departamento de Ingeniería Telemática

Trabajo Fin de Grado

Implementación de una arquitectura para movilidad de sesiones entre dispositivos

Autor: Javier Estévez Aguado

Tutor: Patricia Arias Cabarcos

Director: Florina Almenárez Mendoza

Leganés, febrero de 2015

Título: Implementación de una arquitectura de movilidad de sesiones entre dispositivos

Autor: Javier Estévez Aguado

Director: Florina Almenárez Mendoza

EL TRIBUNAL

Presidente:_____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN

VOCAL

SECRETARIO PRESIDENTE

“It’s not about ideas.
It’s about making ideas happen”
Scott Belsky

“If you find a path with no obstacles,
it probably doesn’t lead anywhere”
Frank A. Clark

Agradecimientos

Agradecer la elaboración de un proyecto siempre es un momento difícil. Hay demasiada gente a la que agradecer y muy poco espacio, por lo que, de antemano, pido perdón si hay alguien que no está en estas líneas.

En primer lugar, mi más profundo agradecimiento a mi tutora, Patricia Arias, y a mi directora, Florina Almenárez, por darme la oportunidad de participar en este proyecto, su paciencia y su simpatía frente a todos los contratiempos que han ocurrido (que no han sido pocos), además de por su ayuda y disponibilidad.

He de dar las gracias a mi familia por haber hecho que escribir estas líneas sea posible. A mis padres, Jesús y Conchi, por el esfuerzo realizado y apoyarme en la decisión de elegir una carrera, sin imponer su criterio. A mi hermana María, que sé que está orgullosa, y a mi abuelo José, aunque no haya llegado a ver este momento, y mi tío Jose Ignacio, por lo que ha conseguido superar. También a Paco y a Isa, que aunque no lo sean de sangre, ya se encargan ellos de decir con orgullo que son mis tíos.

La etapa en la universidad me ha permitido conocer a mucha gente. Me es imposible no dar las gracias a mis compañeros. A los que estuvieron allí desde el principio: Álex, Isra, Paloma y Cristina, mi primer grupo de amigos, al que pronto se unieron otros como Adrián, Lara, Leticia o Rodrigo, y a los que llegaron en la mitad, con mención especial a Paula, Patricia, Laura, Dani, Carlos y Javi. Poner el nombre de todos me llevaría varias páginas, pero no me olvido del resto.

Y, por supuesto, al resto de mis amigos. A Tainá, por sus consejos y su creatividad. A Ernesto, por ejercer de usuario exigente. A Sara, porque 20 años después de conocernos sigo pudiendo contar con ella. A Silvia, Miriam y Eli, que siempre confiaron en que todo me iba bien, y si no, seguro que mejoraría. A Raúl, ejemplo a seguir y, posiblemente, quien más me ha enseñado el valor del trabajo continuo. Gracias por confiar en mi más que yo mismo y por toda tu ayuda y alegrías compartidas. A Javi, por melón, por su desorden y por desordenarme, y por todas las playlists que han puesto banda sonora a este proyecto. A Cris y el resto de Galileos, por todas sus visitas a la capital y todas las que quedan, ya sea a la española o a alguna otra, y a Sergio por los viajes hechos y los que nos esperan.

En definitiva, gracias a todos por haberos tomado un tiempo para estar ahí durante estos años.

Resumen

Las aplicaciones de reproducción musical y de video en *streaming* han visto su uso aumentado en los últimos años. Vivimos en la cultura del aquí y ahora, y gracias a internet y la proliferación de los dispositivos móviles con conexión a la red, ya sean teléfonos móviles, *tablets* u otros, han hecho que estos contenidos estén disponibles en todas partes. A su vez, los servicios de la nube han hecho que nuestros datos estén interconectados entre todos y cada uno de nuestros dispositivos. Aprovechando esta interconexión nace “SuSSo”, un *framework* para transmisión de sesiones entre dispositivos.

Android es, actualmente, el sistema operativo móvil más utilizado en *smartphones*, y el segundo más utilizado en *tablets*.

En cuanto a los servicios en *streaming*, si nos centramos en los musicales, Spotify presenta más de 40 millones de usuarios activos en 58 países, cifra que le ha valido para convertirse en el principal proveedor de estos servicios. De estos 40 millones, más de 12 son usuarios Premium.

Este proyecto tiene como objetivo la movilidad de sesiones de Spotify entre dispositivos Android, haciendo así que el usuario pueda continuar su actividad en un dispositivo distinto de aquel en el que comenzó.

Para ello, se ha hecho uso del recién estrenado Android SDK de Spotify para crear una aplicación que utiliza las credenciales del usuario para reproducir una lista musical. Una vez el usuario termina su actividad en el dispositivo, esta se transfiere a otro mediante el uso de *sockets*, donde puede continuar desde el punto en el que quedó.

Cabe destacar que SuSSo presenta muchas más posibilidades, y esto es tan sólo un prototipo de ejemplo de utilización, pudiendo implementarse en otros servicios.

Palabras clave: Android, Spotify, Movilidad de sesión, Continuidad

Abstract

Applications of music and video in streaming have increased their use in the last few years. We live in the culture of here and now, and, thanks to the Internet and the new devices capable of connecting to the network, like smartphones, tablets or others, these contents available everywhere. Also, the cloud services have made our private data interconnected in all our devices.

Taking advantage of this interconnection we have created “SuSSo”, a framework of session transmission between devices.

Android is, right now, the mobile operating system used the most in smartphones all around the world, and the second one in the tablets market.

Referring to streaming services, if we focus on the musical ones, Spotify has got more than 40 million users around 58 countries. These numbers have made it become the principal provider of these services. Of the 40 million users, more than 12 own Premium accounts.

This project has got as a goal the transference of Spotify’s sessions between Android devices, so the user can continue his activity in a different device from the one he started it on.

In order to do this, the brand new Spotify Android SDK has been used to create an application that uses the user’s credentials to reproduce a playlist. Once the user finishes his activity in the devices, this is transferred to another one using sockets, making it possible to continue from the point it ended.

It is worth to notice that SuSSo presents a lot more of possibilities, and this is only a prototype of an example of use, being able to be implemented in some other services.

Keywords: Android, Spotify, Session Mobility, Continuity

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

Contenido

A lo largo de este primer capítulo se expondrán la motivación y objetivos que se buscan con la realización de este proyecto. Además, se comentarán las fases de desarrollo por las que ha pasado el proyecto, los medios empleados, una guía de términos utilizados y una descripción de la estructura de esta memoria.

1.1 Motivación

Es una realidad que la utilización de los servicios de *streaming* para el ocio cobran cada día más protagonismo. Los resultados de una encuesta realizada a usuarios de redes 4G [4] muestran que el 44% lo utilizan para ver vídeos en *streaming*, el 24% para escuchar música online y el 21% para escuchar la radio. A su vez, España está en los puestos superiores de penetración de *smartphones* en países desarrollados, pues el 85% de la población es poseedora de uno de estos dispositivos inteligentes.

No es de extrañar la evolución que está ocurriendo en los servicios de convergencia tecnológica: el usuario utiliza la misma aplicación en su teléfono móvil, *tablet* e incluso televisión. Dentro de la utilización de estos servicios, el sistema operativo más utilizado en dispositivos móviles es Android, mientras que el servicio de música en *streaming* más demandado es Spotify.

Actualmente buscamos la movilidad, el poder usar la mejor opción dependiendo del contexto. Esto nos motiva a, si estamos usando un servicio de larga duración, querer utilizar esta misma sesión a través de diferentes dispositivos, dependiendo de la situación en la que nos encontremos. Con el objetivo de hallar esta movilidad se propone la creación de una tecnología *Multi Device–Single Sign-On* (de ahora en adelante, MD-SSO), que se define como “inicio de sesión único para usuarios que alternan dispositivos”. La tecnología base necesaria para materializar el concepto ya existe hoy en día, sin embargo, apenas está explotada y no hay un estándar de uso.

Un ejemplo de uso de esta tecnología sería el siguiente:

María se ha suscrito a un curso de idiomas online a través de *podcasts*. Ella aprovecha la vuelta a casa del trabajo para escuchar las lecciones durante el trayecto a través de su teléfono móvil. Gracias a la tecnología MD-SSO, una vez ha llegado a casa la sesión se transfiere de forma transparente para ella a otro de sus dispositivos: ordenador personal, tablet, equipo de música o televisión, continuando desde el punto en el que lo dejó de forma automática. Esto le aporta comodidad y ahorro de batería en su teléfono móvil, y le permite continuar la escucha sin tener que buscar el punto en el que se quedó.

Para este proyecto se ha tomado como base el artículo *“SuSSO: Seamless and Ubiquitous Single Sign-on for Cloud Service Continuity Across Devices”* [5], publicado en el año 2012 y escrito por P.A. Cabarcos et al. En él, se propone una arquitectura capaz de enviar la sesión de una o varias aplicaciones entre distintos dispositivos de un mismo usuario con el fin de que la sesión continúe en el dispositivo destino.

La novedad de la propuesta consiste en la creación de un *framework* universal, contrario a las tecnologías existentes actualmente, basadas en SIP (Protocolo de Inicio de Sesiones) o sólo funcionales entre dispositivos propios de un mismo fabricante. Sus dos grandes fortalezas son la interoperabilidad entre diferentes plataformas y la flexibilidad que presenta.

Este proyecto consiste en la realización de un prototipo de la arquitectura propuesta por *SuSSo*, implementando una aplicación capaz de continuar la sesión de un usuario en Spotify a través de dispositivos Android. Esta sesión se iniciaría en un dispositivo para después ser transmitida a otro. Así, se cubriría la posibilidad de la utilización de distintos dispositivos para escuchar una misma lista de canciones dependiendo del contexto en el que el usuario se encuentre.

1.2 Objetivos

El objetivo global de este trabajo de fin de grado es implementar una tecnología MD-SSO basada en la arquitectura propuesta en [5] para la aplicación de música en *streaming* Spotify a través de dispositivos Android. Para ello, se ha programado, haciendo uso del Android SDK *(Kit* de desarrollo de *software)* de Spotify, una pequeña aplicación para Android basada en Spotify que reproduce una lista de canciones predefinida. Teniendo esta aplicación instalada en dos dispositivos y una aplicación capaz de recibir sesiones podemos transferir la sesión de uno a otro de forma automática, así, la canción que se está escuchando en el primero pasa a reproducirse desde el mismo punto en el segundo, pudiendo continuar con la escucha del resto de la lista de reproducción.

La transferencia de esta sesión se realiza conectando ambos dispositivos a través de un *socket*. Antes del fin de la sesión ha de guardarse el estado actual de reproducción en un fichero para transmitírselo al segundo dispositivo, de forma que este tome el relevo y comience en el punto donde se dejó.

A más bajo nivel, podrían tomarse como sub-objetivos la familiarización con Android a nivel programático y el conocimiento de las librerías de Spotify para la realización de aplicaciones de dicho *software* para el sistema operativo de Google. Dentro de la programación en Android, uno de los aspectos más importantes para la realización del presente proyecto es la utilización de una arquitectura basada en *sockets* de comunicación.

Además, este proyecto sirve como precedente para futuras aplicaciones basadas en la misma arquitectura, de forma que podría tomarse la arquitectura utilizada como estándar y adecuarla a la aplicación de la cual se desee continuar la sesión. Pese a que en el presente trabajo se utilice la música en *streaming* como base, podría adaptarse a aplicaciones de video o navegadores web entre otros. En el capítulo 7, “Conclusiones y Líneas Futuras de Investigación”, se tratará este aspecto con más detalle.

1.3 Fases de desarrollo

Debido a la falta de conocimientos sobre la programación en Android, la primera fase de desarrollo de este proyecto ha consistido en el aprendizaje de la arquitectura del sistema operativo móvil y las bases de su programación. Dado que está basado en Java, lenguaje que sí se conocía previo a la realización de este proyecto, esta fase ha resultado relativamente sencilla y rápida. Se ha ahondado especialmente en la comunicación con otros dispositivos y en el sistema de ficheros. En esta fase además se buscó el entorno de desarrollo óptimo para el objetivo de la aplicación entre todos los disponibles para el desarrollo en Android.

A la vez, se establecieron las funciones con las que debería contar la aplicación para su correcto funcionamiento, así como los parámetros necesarios para la continuación de la sesión, que quedarían guardados en el fichero a transmitir. Una vez determinadas estas, se produjo una investigación en las librerías de Spotify para comprobar si estaban disponibles y, por tanto, la aplicación era viable. Esta fase ha estado en constante cambio durante todo el proyecto debido a la actualización continua de versiones de la librería de Spotify durante su realización. El cambio más grande vino con la aparición de un SDK específico para Android de Spotify, cuyo lanzamiento en versión beta se realizó a finales de Junio de 2014.

Una vez finalizado el proceso de investigación de funciones y el de aprendizaje de programación en Android, se procedió a la creación de una pequeña aplicación que conectaba dos dispositivos a través de *sockets* con el fin de enviar un fichero entre ambos.

Tras esta fase, y teniendo claro que quería seguirse la arquitectura de [5] se barajaron varias posibilidades de implementación de la aplicación. Las posibilidades que ofrecía la librería de Spotify fueron determinantes para la elección final, en la cual se decidió crear dos aplicaciones: Una aplicación cliente, que reproduce música y guarda el archivo con los datos de sesión y una aplicación servidor, que recibe este fichero y abre la aplicación cliente para la restauración. El resto de opciones barajadas se comentarán en el capítulo 3 de este documento, “Análisis y Diseño de la Solución Técnica”.

Así pues, la primera etapa del proyecto en su conjunto ha consistido en la definición de objetivos y el planteamiento inicial de la solución.

La segunda etapa consistió en la implementación del sistema. Con el diseño elegido se procedió a la creación de estas dos aplicaciones. En principio se creó la aplicación cliente, capaz de reproducir música. Conseguido el hito de la reproducción, se procedió al de guardado de estado de la sesión en fichero, para lo cual era necesaria la captura de eventos. Dado que en este punto aún no existía la comunicación entre dispositivos, se simuló la recepción de este fichero para realizar y probar las funciones de lectura del mismo y restauración de la sesión con la información aportada en él.

Probado el funcionamiento de la aplicación cliente, se programó la aplicación servidor. Esta, estando instalada en un dispositivo distinto, recibe el fichero creado por la primera y, basándose en la información que aporta, abre la aplicación pertinente para la continuación de sesión. En esta fase se programaron en consonancia las dos aplicaciones: la aplicación cliente debería ser capaz de enviar el fichero, mientras que la aplicación servidor debía ser capaz de recibirlo, guardarlo y actuar en consecuencia de su lectura, abriendo la aplicación cliente, que debía procesar el fichero y restaurar la sesión.

Ambas aplicaciones se probaron primero por fases y después de forma global. Parte de estos *tests* se realizaron en un emulador, mientras que otros se realizaron directamente con dos dispositivos físicos, especialmente la parte de comunicación.

Finalizadas las pruebas y comprobado el correcto funcionamiento de la aplicación, se dio por concluida la parte técnica del proyecto y se pasó su documentación, tercera y última etapa del proyecto.

En el capítulo 6 de este documento, “Gestión del Proyecto”, se tratarán en detalle cada una de las fases por las que pasó el desarrollo del proyecto, cuantificando su duración, exponiendo los problemas encontrados durante cada una de ellas y mostrándolas en conjunto de forma gráfica para establecer la duración total del mismo.

1.4 Medios Empleados

Como se ha comentado en el apartado anterior, previo al desarrollo de este proyecto se realizó una búsqueda y estudio de herramientas para la creación del mismo.

Para la programación de las aplicaciones se sopesó entre la utilización de dos herramientas *software*: Eclipse y Android Studio. Por su mayor facilidad de uso y claridad se utilizó la segunda, creada por Google. Esta herramienta incluye todo lo necesario para la programación de aplicaciones móviles, desde el SDK Android hasta el Java Development Kit (JDK). También incluye un emulador de dispositivos (AVD,

Android Virtual Device), que se utilizó en ciertas fases del proyecto para la realización de pruebas. En este emulador se crearon varios dispositivos, cuya configuración se muestra en la figura siguiente:

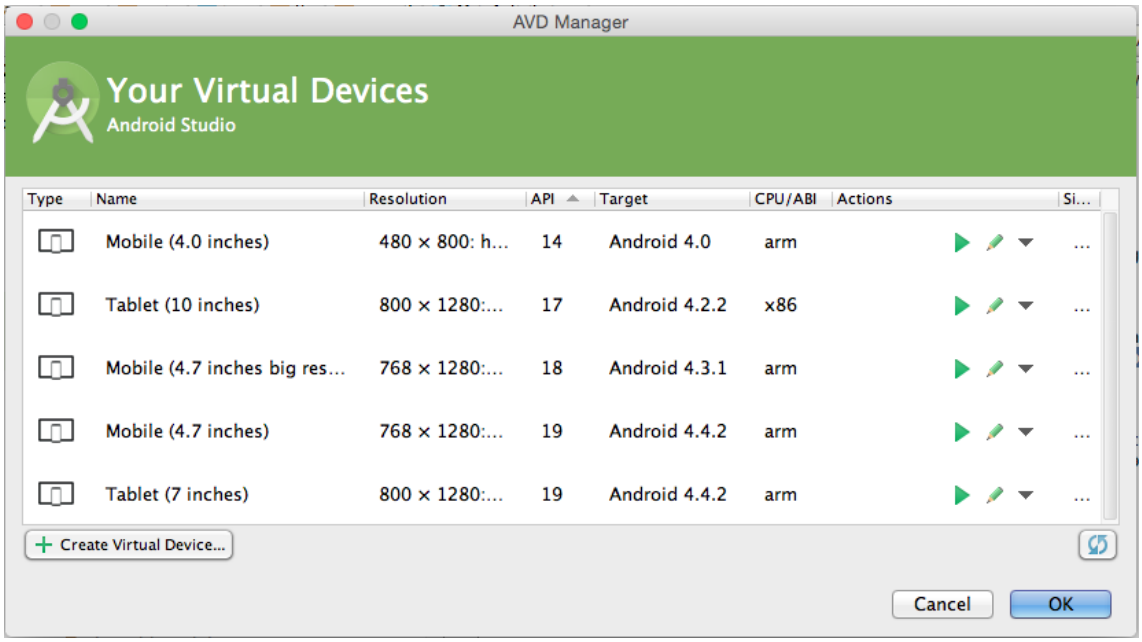

Figura 1: Dispositivos Virtuales Android

Como puede observarse, se hicieron pruebas en diferentes versiones de Android (todas superiores o iguales a la 4.0) y con diferentes resoluciones. A su vez, se modificaron estos dispositivos para la realización de los *tests*, probando así el uso de CPU y las distintas capacidades de memoria interna y/o externa.

De las versiones de la librería de Spotify para Android la más adecuada fue la versión 1.0.0beta6, pues contenía todas las funciones necesarias para el correcto funcionamiento de la aplicación que se buscaba crear.

La programación se ha realizado en un ordenador portátil personal con procesador Intel Core i7 de 2,9 GHz, memoria RAM de 8 GB 1600 MHz DDR3 y sistema operativo OS X Yosemite 10.10.1.

Las pruebas de la aplicación se han realizado en diferentes dispositivos:

-Teléfono móvil HTC Wildfire S con KitKat Rom de CyanogenMod (Android 4.0)

-Teléfono móvil HTC Desire con KitKat Rom de CyanogenMod (Android 4.0.4)

-Teléfono móvil Sony Xperia E3 con Android 4.4

-Tablet Wolder miTabPop con Android 4.1.1

Además, fue necesaria la utilización de un cable *microUSB* para cada dispositivo, así como una tarjeta *SD*. La capacidad de esta tarjeta fue de 4GB para cada uno de los dispositivos.

1.5 Terminología

Como anexo a este documento se incluye un glosario de términos, sin embargo, se utilizará este apartado para la aclaración y definición de conceptos que se utilizarán a lo largo de la memoria.

Android: Sistema operativo de Google para dispositivos móviles, ya sean estos *smartphones*, *tablets* u otros como relojes, televisiones o navegadores GPS *(Global Positioning System).*

Aplicación Cliente: Se utilizará este término para referirse a la aplicación capaz de reproducir música con un usuario autenticado en su cuenta de Spotify, así como el guardado, envío del fichero con datos de la sesión y restauración de la sesión en base a dicho fichero.

Aplicación Servidor: Este término se utilizará para referirse a la aplicación capaz de recibir el fichero con datos de la sesión enviado por el dispositivo donde se creó. Esta aplicación, una vez recibe el fichero, inicia la aplicación cliente para posibilitar el restaurado de sesión.

***Smartphone*:** Teléfono móvil semejante a un pequeño ordenador personal de bolsillo capaz de realizar funciones tales como el acceso a Internet, funciones multimedia (reproducción de música y vídeo, cámara de fotos), agenda o navegación GPS entre otras muchas. Habitualmente no cuentan con un teclado físico, sino que se sustituye por uno virtual.

Socket: Canal de comunicación entre dos programas, que permite que un proceso hable con otro en la misma o, de forma general, en distintas máquinas.

Spotify: Aplicación de reproducción musical en *streaming*.

Spotify Android SDK: Librería que permite al programador crear aplicaciones capaces de realizar funciones propias de Spotify, como son la autenticación de usuario o la reproducción musical.

***Streaming*:** Distribución digital de multimedia (generalmente vídeo o audio) que permite al usuario consumir el producto a la vez que se descarga.

SuSSO: A través de este acrónimo se hace referencia al artículo en el cual se ha basado la arquitectura de este proyecto, *"Seamless and Ubiquitous Single Sign-On for Cloud Service Continuity Across Devices" [5]*.

Tablet: Dispositivo móvil cuyas funciones son semejantes a las del *smartphone*, a excepción de las funciones propias de telefonía (envío y recepción de llamadas o mensajes de texto), normalmente de mayor tamaño que estos.

Usuario *Premium*: Se refiere al usuario que abona una cuota mensual a Spotify para tener funciones especiales, como la reproducción en alta calidad y sin anuncios o la reproducción en dispositivos móviles.

1.6 Estructura de la memoria

El presente documento consta de 7 capítulos, siendo este el primero de ellos, en el cual se presentan la motivación y los objetivos de este proyecto, las fases de desarrollo, los medios utilizados y la terminología que se seguirá en los capítulos siguientes.

El segundo capítulo, “Planteamiento del Problema” analiza el estado del arte para comprobar si el proyecto es viable con la tecnología actual y analizando trabajos relacionados para destacar la novedad del propuesto, y expone las restricciones y el marco regulador del mismo.

El capítulo número 3, “Análisis y Diseño de la Solución Técnica”, muestra los requisitos que debe cumplir la aplicación. Además, para la solución elegida, indica sus restricciones de uso, la arquitectura utilizada, las alternativas de diseño planteadas y dos casos de uso.

En “Implementación del Sistema”, el cuarto de los siete capítulos, se profundiza en cómo se ha solucionado el problema planteado a un nivel técnico. Es en este capítulo donde se desarrolla con detalle la arquitectura del sistema y se muestran las clases y funciones utilizadas en la programación para el correcto funcionamiento de la aplicación.

El quinto capítulo, “Evaluación y resultados”, consta del plan de pruebas seguido para comprobar el funcionamiento de cada uno de los bloques de la aplicación, así como la aplicación de forma global. También muestra una evaluación del rendimiento.

El penúltimo capítulo, “Gestión del Proyecto”, se centra en la planificación y el análisis económico del mismo.

El último capítulo “Conclusiones y Líneas Futuras”, expone las conclusiones extraídas de la realización de este proyecto y propone líneas futuras de investigación y desarrollo.

Además, se incluye como anexo un glosario de términos.

Capítulo 2
Planteamiento del Problema

Contenido

Este segundo capítulo muestra el planteamiento del problema mediante el análisis del estado del arte de los medios utilizados, trabajos relacionados y su marco regulador y restricciones. Con este análisis se buscan los siguientes objetivos:

1) Situar al sistema operativo Android dentro de su mercado de competencia.

2) Situar la aplicación Spotify dentro de su mercado de competencia.

3) Conocer el grado de investigación e implantación de las soluciones de movilidad de sesión existentes.

Una vez conocidos estos datos será más fácil realizar una definición de la solución buscada, conociendo las limitaciones que presentan las tecnologías utilizadas y enfocarlo desde la perspectiva más adecuada.

2.1 Estado del Arte

El estado del arte describe las investigaciones más recientes y actuales sobre un tema específico con el objetivo de conocer cuáles son los últimos avances. En este apartado se analiza el estado del arte de las tecnologías empleadas para el desarrollo de esta aplicación, así como el de la movilidad de sesión. A lo largo de este capítulo se muestra el estado del arte de las tecnologías base utilizadas y de las soluciones existentes para movilidad de sesión a nivel de aplicación.

2.1.1 Tecnologías Base

2.1.1.1 Android

2.1.1.1.1 Introducción

Android [1] es un sistema operativo para dispositivos móviles basado en el núcleo de Linux y desarrollado por Google. La interfaz de usuario de este sistema operativo se basa en la manipulación directa, lo que hace que su utilización sea mayoritariamente en dispositivos con pantalla táctil, como teléfonos móviles y *tablets*. Se trata de un conjunto de *software* que agrupa un sistema operativo, *middleware* y aplicaciones móviles. Esta estructura hace que se pueda abstraer al sistema operativo de las aplicaciones a través de un *middleware* basado en Dalvik, una máquina virtual.

Android se basa en el código libre para el desarrollo de aplicaciones. Mediante el SDK que Google proporciona, los desarrolladores obtienen las herramientas y APIs (Interfaz de Programación de Aplicaciones) necesarias para el desarrollo de aplicaciones. El lenguaje utilizado para esta programación es Java.

2.1.1.1.2 Historia

Pese a su largo recorrido y gran penetración en el mercado móvil, Android cuenta con apenas 7 años de historia. Fue en Noviembre de 2007 cuando se lanzó la versión beta, mientras que la primera versión comercial, Android 1.0, apareció en 2008 junto al teléfono “T-Mobile G1” de la compañía T-Mobile. Antes de ser parte de Google, Android era un sistema operativo en fase de desarrollo por una pequeña compañía de nombre Android Inc, adquirida por Google en el año 2005.

Los orígenes de Android van ligados al OHA, un consorcio de 48 empresas tecnológicas que busca el desarrollo de estándares abiertos para dispositivos móviles. El código fuente del sistema operativo se liberó a finales de 2008, bajo licencia de código abierto Apache. Fue también en 2008 cuando se creó el “Android Market”, precursor del actual “Play Store”, para la subida y descarga de aplicaciones.

Aún contando sólo con 5 versiones, cada una de ellas ha contado con numerosas subversiones (como curiosidad, todas ellas con nombres de alimentos dulces). La última de sus versiones hasta el momento, Android 5.0, fue lanzada en Noviembre de 2014.

En el tercer trimestre de 2014, el sistema operativo móvil de Google mantiene el liderazgo adquirido a finales de 2010 como el más utilizado en smartphones, con una cuota de mercado del 84,4%, seguido muy de lejos por iOS, el sistema operativo de Apple, con un 11,7%, que cada vez queda a más distancia.

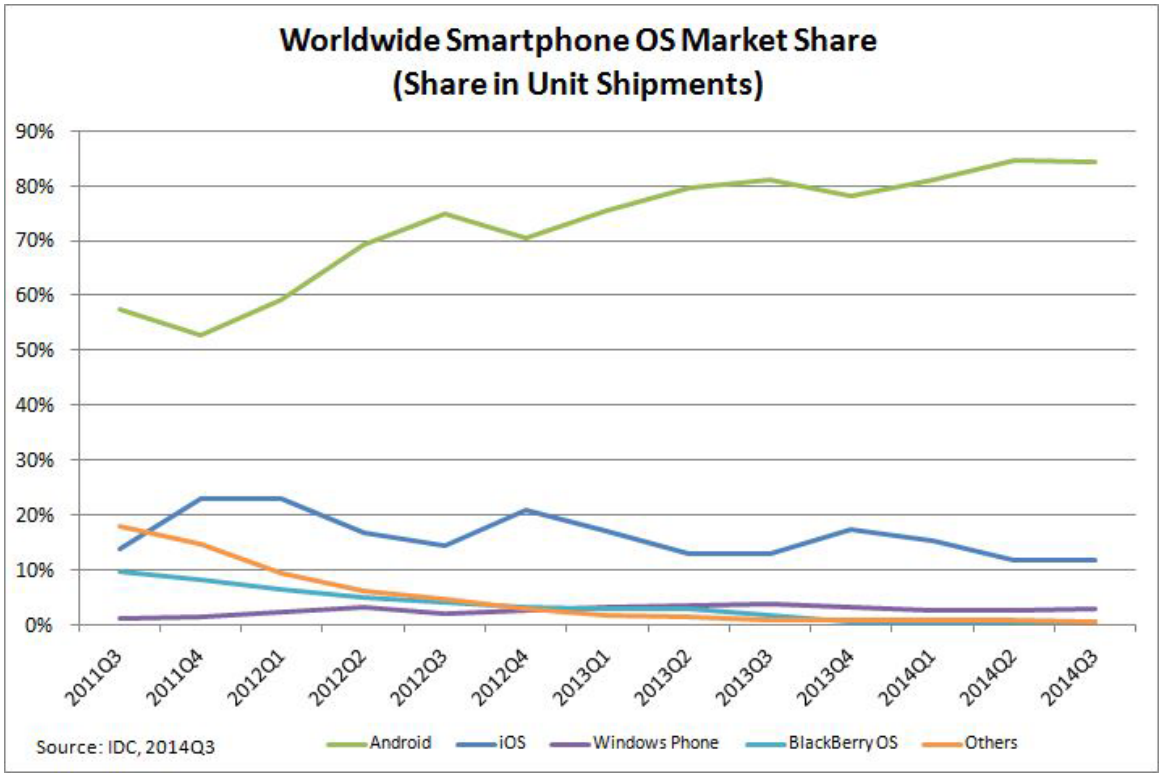

Figura 2: Cuota de mercado de Smartphones por sistema operativo

Fuente: IDC [6]

En el mercado de las tablets, Android está por debajo de iOS, aunque sus ventas cuatrimestrales crecen más que las del iPad de Apple. En diciembre de 2013, el 40% del mercado correspondía a Android, frente al 51% de iOS. [7]

2.1.1.1.3 Características y Arquitectura

Si hubiera que definir este sistema operativo de forma breve a través de una de sus características, sin duda sería “código abierto”. El hecho de ser una plataforma basada en Linux facilita el desarrollo de nuevas aplicaciones de forma gratuita para el desarrollador, quien además cuenta con librerías para la reutilización de código y entornos de desarrollo con emuladores de diferentes dispositivos.

Actualmente existen dispositivos *hardware* de todo tipo adaptados a Android. El hecho de que sus aplicaciones estén implementadas en Java garantiza su ejecución en cualquier tipo de procesador. Así pues, la portabilidad es otra de sus características. Esta portabilidad hace que las prestaciones multimedia sean variadas según el dispositivo. No obstante, las codificaciones de imagen y audio más habituales son soportadas en todos ellos (MP3, MPEG4, JPG, PNG, GIF).

El sistema dispone de una máquina virtual propia que se encarga de la gestión de recursos y memoria de forma eficiente, además de una base de datos, SQLite.

En cuanto a su arquitectura, alguno de sus componentes están basados en Internet, como los ficheros XML (*eXtensible Markup Language*) que conforman la interfaz de usuario, facilitando así la correcta visualización en pantallas de diferente forma, tamaño y resolución.

Android se compone de cuatro capas funcionales basadas en software libre: el núceo de Linux, el entorno de ejecución, las librerías nativas, el marco de aplicación y las aplicaciones.

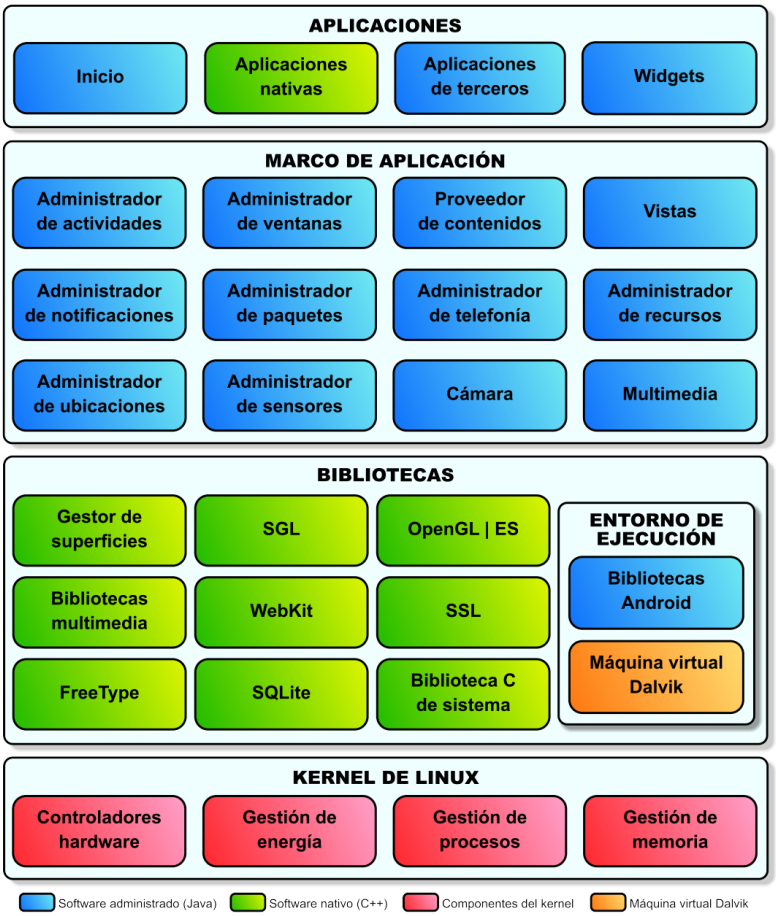

Figura 3: Arquitectura de Android

Núceo de Linux

Esta capa es la base de Android para la gestión de distintos elementos (memoria, procesos, energía y pila de protocolos) y el control de los controladores de dispositivos. Android emplea el sistema operativo Linux en su versión 2.6.

Entorno de Ejecución

Consta de dos componentes: Las bibliotecas de Android, con la mayoría de funcionalidades de Java, y la máquina virtual Dalvik, componente principal. Esta máquina es la encargada de la ejecución de aplicaciones. Se trata de una versión de la máquina virtual de Java optimizada para dispositivos móviles. Su diseño permite que un mismo dispositivo pueda ejecutar múltiples máquinas virtuales. Está basada en registros y utiliza ficheros Dalvik ejecutables (con formato .dex) compilados y optimizados para el ahorro de memoria. Cada aplicación se ejecuta en su propio proceso y tiene su propia instancia de esta máquina.

Bibliotecas

Las bibliotecas nativas de Android facilitan las funcionalidades a las aplicaciones mediante la utilización de código ya implementado. Están escritas en C++ mayoritariamente (aunque existe alguna escrita en C) y compiladas en función del código nativo de cada procesador.

Marco de Aplicación

Es la capa conformada por las clases y servicios que utilizan las aplicaciones para realizar sus funciones (sensores, servicios, localización…). Se apoya en las

bibliotecas y en el entorno de ejecución. La mayoría de sus componentes son bibliotecas Java que acceden a los recursos a través de la máquina virtual Dalvik.

Aplicaciones

Es la capa superior de la pila de software, y gran parte del éxito de Android se debe a ellas, pues se permite al usuario tener el control total del software en ejecución. Se incluyen en esta capa tanto las aplicaciones nativas del dispositivo (con o sin interfaz de usuario, y programadas en C++) como las administradas (programadas en Java), ya sean instaladas por el usuario o incluidas de serie con el dispositivo. Dentro de las aplicaciones, la más importante es la aplicación principal del sistema: “inicio”, también conocida como “lanzador”. Es la que permite ejecutar otras aplicaciones, proporcionando la lista de instaladas y mostrando diferentes escritorios con accesos directos o pequeñas aplicaciones incrustadas (*widgets*, también incluidos en esta capa).

Que todas las aplicaciones estén dentro del mismo marco implica que podemos crear aplicaciones que utilicen los mismos recursos que las nativas y que estas pueden ser reemplazadas por aplicaciones de nuestra elección.

2.1.1.1.4 Fundamentos de las aplicaciones

Habitualmente, una aplicación Android suele ser una aplicación escrita en Java y compilada a un fichero .apk, reconocido como instalable por el sistema operativo. Para su desarrollo se utiliza el SDK de Android. Existe además la posibilidad de programación en C o C++ haciendo uso del NDK (*Native Development Kit*).

Una aplicación puede estar compuesta por uno o más componentes, que realizan funciones diferentes para dar el comportamiento a la aplicación. Cada uno de ellos puede ser activado de manera individual.

Android sigue un principio de mínimos privilegios: a cada proceso se le adjudica acceso únicamente a los componentes que necesita. Aún así, se pueden tener varias aplicaciones compartiendo un recurso (siempre y cuando estén firmadas con el mismo certificado) y, si una aplicación sabe de antemano que quiere acceder a ciertos recursos, debe pedírselo al usuario de forma explicita durante el proceso de instalación.

2.1.1.1.4.1 Componentes de una aplicación

Existen cuatro componentes dentro de una aplicación Android: Actividades (interfaces visuales con las que interactúa el usuario), Servicios (procesos que se ejecutan en segundo plano), Proveedores de contenidos (permiten que una aplicación ponga datos a disposición de otra, también conocidos como “*Content Providers*”) y Receptores de eventos (conocidos como “*Broadcast Receivers*”, esperan a que se produzca un determinado evento para realizar alguna acción con él).

2.1.1.1.4.2 Ciclo de vida

El ciclo de vida de una aplicación recoge las etapas que pueden seguir sus procesos desde que comienza hasta que finaliza. Podemos dividir el ciclo de vida de una aplicación en dos: el ciclo de vida de la aplicación como tal y el de sus componentes.

El sistema es el encargado de controlar el ciclo de vida de una aplicación, pues es quien se encarga de la gestión de procesos Linux de cada una de ellas en función de las necesidades del usuario y los recursos disponibles. En memoria se mantienen los procesos el mayor tiempo posible, sin embargo, a veces es necesario eliminar alguno de ellos. Para decidir que qué procesos han de mantenerse se define una jerarquía de procesos.

Existen cinco niveles de jerarquía de importancia de procesos en Android, que determinan si un proceso ha de eliminarse o no dependiendo de su prioridad:

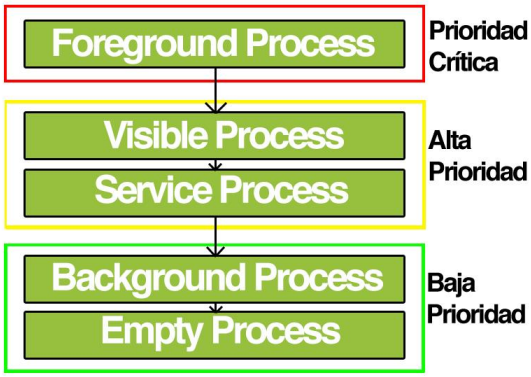

Figura 4: Jerarquía de procesos en Android

El ciclo de vida de una actividad depende de su relación con las demás actividades y de la pila de actividades. Existen cuatro estados para una actividad en Android: Activa (*Running*, la primera en la pila y, por lo tanto, visible y enfocada), Visible (*Paused*, visible pero no enfocada, es decir, se ha pasado a otra actividad que no ocupa toda la pantalla o que deja ver la actividad anterior), Parada (*Stopped*, no visible en pantalla al estar oculta por otras actividades) y Destruida (*Destroyed*, actividad terminada que sale de la pila de actividades). El paso de un estado a otro produce un evento, que puede ser capturado por ciertos métodos de la actividad. La siguiente figura muestra un diagrama de estados y qué métodos pueden realizar la captura de estados:

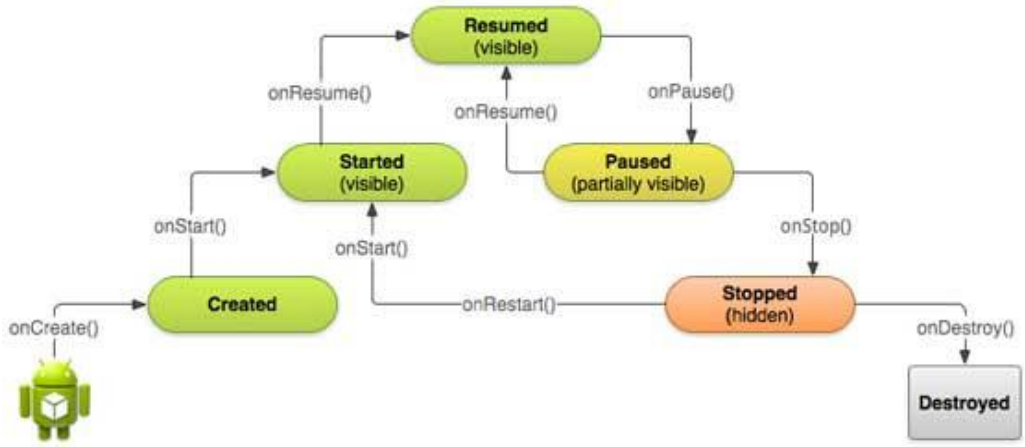

Figura 5: Diagrama de estados de una aplicación Android

La siguiente tabla describe estos métodos:

Método	Utilización
onCreate()	Creación de la actividad
onStart()	Hace visible la actividad
onResume()	Actúa antes de que la actividad comience a interactuar con el usuario
onStop()	Utilizado cuando la actividad deja de ser visible
onRestart()	Utilizado justo antes de volver a comenzar una actividad parada
onPause()	Cuando otra actividad la que pasa a estado activo
onDestroy()	Destrucción de la actividad

Tabla 1: Métodos de estado de Android

2.1.1.1.4.3 Principales ficheros de una aplicación

A la hora de programar una aplicación en Android, hay tres ficheros muy importantes que recogen la estructura de la aplicación y definen sus componentes, recursos e interfaz de usuario. Estos son los ficheros AndroidManifest.xml, main.xml y R.java.

AndroidManifest.xml

Este archivo es la espina dorsal de toda aplicación Android, pues en él se definen la estructura y los componentes de la misma. Todo componente que necesite la aplicación debe estar declarado en este fichero, donde además se acompañará la información de permisos para interactuar con otras aplicaciones y que otras aplicaciones puedan interactuar con ella.

main.xml

Es el *layout* de la actividad, contenedor del diseño de los diferentes elementos que el usuario ve a través de la interfaz gráfica. Existen varios ficheros de este tipo, normalmente uno por actividad. En los actuales entornos de desarrollo de Android existen dos formas de modificar este fichero: mediante código Java o a través de una interfaz gráfica que muestra un dispositivo sobre el cual realizar el diseño de la interfaz gráfica, bien mediante elementos prediseñados o propios del usuario.

R.java

Contiene los punteros a los recursos empleados, aquellos que no forman parte del código Java. Un ejemplo de estos recursos serían las cadenas de texto, archivos, *layouts* o imágenes.

2.1.1.2 Spotify

2.1.1.2.1 Introducción

Spotify [2] es, actualmente, el servicio de música en *streaming* más utilizado en el mundo. Cuenta con más de 50 millones de usuarios activos, de los cuales 12 utilizan cuentas de pago. Dispone de más de 30 millones de canciones y acuerdos con algunas de las principales discográficas del mundo.

La aplicación sigue un modelo *freemium*: Los servicios básicos están disponibles de forma gratuita, mientras que las características avanzadas necesitan de una cuenta *Premium*. Esta cuenta *Premium* permite la reproducción sin anuncios, en alta calidad y, para las versiones móviles, la elección de la canción a reproducir en cada momento. La versión gratuita para dispositivos móviles sólo permite la reproducción de listas o el contenido disponible de un artista de forma aleatoria. El 70% de los beneficios que se obtienen se pagan en forma de *royalties* a artistas y discográficas, mientras que el 30% restante queda para la compañía.

Actualmente, está disponible en 58 países en el continente europeo, América y Oceanía. Cada día se añaden unas 20000 canciones, que están disponibles tanto en la aplicación de escritorio para PC y Mac como para dispositivos móviles con sistema operativo Android, iOS, Symbian, Blackberry OS, Windows Phone, Windows Mobile y WebOS. Desde sus inicios, Spotify no ha estado exento de polémica por los derechos musicales de artistas y compañías.

2.1.1.2.2 Historia

Spotify fue creado en Estocolmo, Suecia, en 2006, por Daniel Ek y Martin Lorentzon. Actualmente cuenta con dos sedes: Spotify Ltd., en Londres y Spotify AB, en Estocolmo.

La aplicación fue lanzada al público en Octubre de 2008, a la cual se podía acceder mediante pago o invitación gratuita. Estas invitaciones estaban disponibles para los usuarios de pago. En 2009 comenzó a utilizarse el registro gratuito sin invitación en Reino Unido. Debido a que los usuarios crecían de forma exponencial, Spotify tomó medidas: Se redujo la escucha de música a 10 horas mensuales por usuario y un máximo de 5 escuchas por canción, a la vez que se crearon nuevos tipos de cuentas en 2010: “Spotify Unlimited”, con las mismas características que las cuentas *Premium* pero sin acceso a dispositivos móviles, y “Spotify Open”, con 20 horas de escucha.

2011 fue el año de la gran expansión de Spotify con su llegada a Estados Unidos. Además se lanzó el servicio Spotify Apps, por el cual desarrolladores podían crear una aplicación web a la que se podía acceder desde el programa con sus propias listas de usuario. Para ello se puso a disposición de los desarrolladores una librería web con el API para estas aplicaciones, “Web API”.

Desde su creación hasta este año el servicio sufrió grandes cambios, como acceso desde cuentas de Facebook y la creación de “Spotify Social”, servicio por el cual los usuarios podían seguir a artistas u otros usuarios para enterarse de sus novedades.

La limitación de 10 horas y 5 escuchas por canción fueron abolidas gradualmente, hasta su completa desaparición en 2013, año en el que aparecieron las aplicaciones móviles en el App Store de Apple, Google Play Store de Android y Windows Phone Store de Windows. El uso de estas aplicaciones estaba limitado para usuarios *Premium*, sin embargo, para final de año se abrió a aquellos con cuenta gratuita, con la restricción

de que sólo podrían escuchar listas de música de forma aleatoria, sin poder elegir qué canción escuchar en cada momento.

Poco a poco surgieron las opciones para desarrolladores, sin liberar nunca el código de la aplicación. Así, se abrió la puerta a la creación de aplicaciones basadas en Spotify, que podían reproducir música disponible en el catálogo del programa a través de la cuenta del usuario. Actualmente existen 3:

- Libspotify SDK, una librería en lenguaje C para añadir música a aplicaciones, disponible para Windows, Mac, iOS y Linux
- iOS SDK, actualmente en versión Beta, para la programación de aplicaciones para los dispositivos móviles de Apple
- Android SDK, la más reciente y también en versión Beta, para la programación de aplicaciones para los dispositivos Android

2.1.1.2.3 Funcionamiento y arquitectura

La infraestructura de funcionamiento del programa está basada en Debian y *software* abierto en general. La arquitectura de Spotify está fuertemente orientada a servicios. El motor está compuesto por unos cien servicios, la mayoría de ellos pequeños y simples, escritos en Java y Python en su mayoría. Se comunican utilizando un protocolo propio, Hermes, basado en mensajes y escrito en ZeroMQ y Protobuf. Algunos servicios antiguos utilizan http y XML/JSON. El almacenamiento es, sobretodo, realizado en servidores y accedido a través de la nube. Los archivos de audio se almacenan en Amazon S3 y se utiliza una caché de ellos en el motor de Spotify.

Los clientes mantienen una conexión permanente a un servicio llamado "accesspoint", que funciona como un *router* muy potente, manejando la comunicación con los servicios necesitados. El protocolo entre los clientes y accesspoint es propio, y utiliza un socket multiplexado sobre TCP. Está escrito en C++. Accesspoint maneja el estado de la autenticación, login del usuario, tasa de enrutamiento y mucho más.

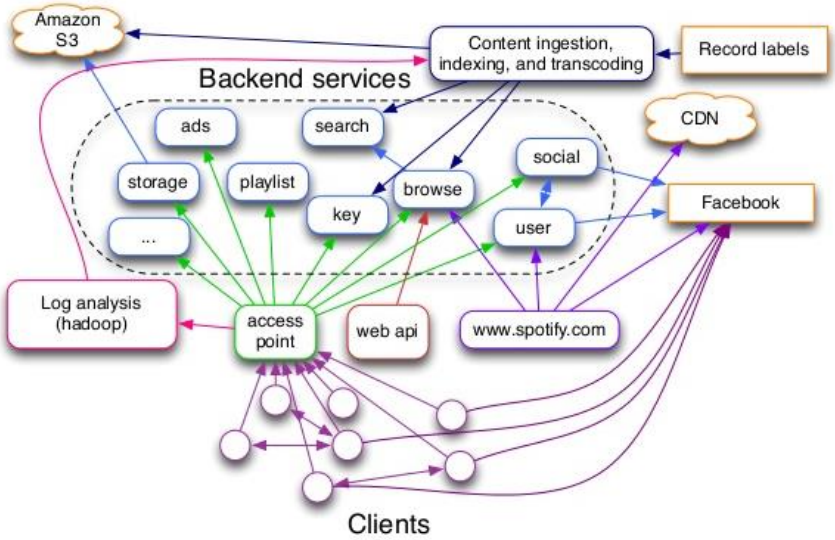

Figura 6: Arquitectura de Spotify

Fuente: [8]

Todos los usuarios de la aplicación de escritorio, móvil y embebida (libspotify) comparten un código base escrito en C++. Cada cliente utiliza este núcleo para proveer la interfaz de usuario, que se traduce al código utilizado por cada plataforma.

El audio tiene tres fuentes: caché, P2P y los servidores de almacenamiento de la compañía, siendo la primera la más utilizada. Debido al gran contenido musical del que se dispone, la compañía no puede mantener todo. Por ello, se llegan a acuerdos con las compañías discográficas, que almacenarán los archivos en sus servidores y se adaptarán al formato utilizado por Spotify.

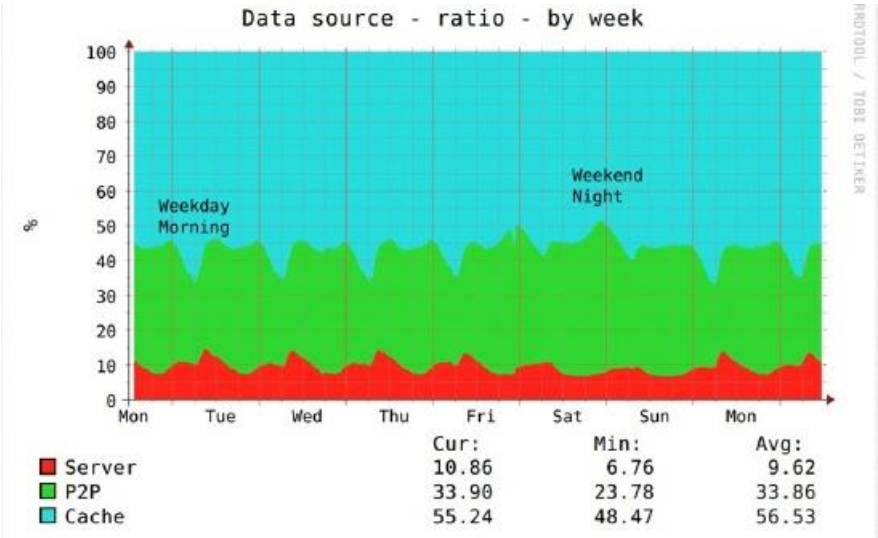

Figura 7: Fuentes de datos de Spotify

Fuente: [9]

El audio puede estar codificado en múltiples formatos: Ogg Vorbis 96, 160 y 320 000 y MP3 320 000. Es el cliente quien elige el más adecuado.

2.1.1.2.4 Spotify Android SDK

El 25 de Junio de 2014 Spotify lanzó su versión beta del *Software Development Kit* para Android, que se trata de una librería con métodos que acceden a las funciones de Spotify. La versión actual, 1.0.0-beta8 fue lanzada el 12 de febrero de 2015. Con el lanzamiento de la versión 6 se eliminó la restricción de uso comercial de las aplicaciones creadas y se permitió el acceso de estas a las tiendas de aplicaciones de los dispositivos móviles.

Además, Spotify activó el envío de notificaciones de eventos a través de `BroadCastReceivers` dentro de su aplicación nativa, de forma que estos pudieran ser capturados por una aplicación desarrollada.

Antes del desarrollo de aplicaciones basadas en Spotify, es necesario registrar éstas en el sitio My Apps dentro de la web de Spotify para desarrolladores (véase [3]) Dentro de esta página es necesario dar un nombre a la aplicación, realizar una descripción breve e introducir una URI de redirección, normalmente con el formato app-name://callback, que deberá utilizarse dentro del código de la aplicación y del archivo AndroidManifest.xml. Spotify proporciona un Client ID a cada aplicación para su uso.

La versión mínima de Android necesaria para el funcionamiento de las aplicaciones es 4.0, *Ice Cream Sandwich*.

Utilizar el SDK no necesita de una cuenta *Premium*, sin embargo, Spotify recomienda darse de alta como desarrollador, para lo cual sí es necesaria este tipo de cuenta.

El SDK de Android cuenta con tres paquetes: `com.spotify.sdk.android`, que contiene la clase principal Spotify, `com.spotify.sdk.android.autentication`, para el proceso de autenticación, y `com.spotify.sdk.android.playback`, para la reproducción musical. Cabe destacar que las aplicaciones móviles creadas con este SDK necesitan de una cuenta *Premium* de Spotify para ser utilizadas.

La autenticación se realiza mediante los métodos dentro de las clases que componen el paquete de autenticación, y está basada en el protocolo OAuth. Para ello, antes de la creación del objeto Spotify dentro de la aplicación ha de llamarse al método `openAuthWindow()` de Spotify, que abre una ventana en el explorador de Internet del dispositivo en la cual se piden las credenciales de usuario y los permisos necesarios. La mejor opción es llamar a este método dentro del método `OnCreate()` de la aplicación programada. El proceso de autenticación del Android SDK es el mismo que el de Spotify Web API. La autenticación a través de Facebook también está disponible en esta ventana.

Una vez el usuario ha introducido sus credenciales, a través de la dirección *callback* antes creada nos devolverá a la aplicación, se verifica que son correctas en el método de Android `OnNewIntent()` y se crea un nuevo objeto Spotify sobre el que realizar las acciones de reproducción. Para la creación de este objeto se extrae el *token* de autorización que provee el *callback*, y se le pasa a un objeto Config.

La versión actual del entorno de desarrollo no proporciona información del usuario, como pueden ser sus playlists o artistas seguidos. Tampoco hay acceso al catálogo musical, por lo que ha de predefinirse la lista de canciones que sonarán. Además, una vez ha iniciado la sesión no se puede comprobar la identidad del usuario, sin embargo, sí es posible acceder al *token* de sesión.

El reproductor creado funciona en un hilo separado al hilo principal de la aplicación, así que las llamadas son asíncronas. Por ello, es seguro utilizar los métodos del reproductor desde el hilo de la interfaz de usuario. Estas llamadas son no-bloqueantes y se reenvían al hilo correcto.

Una de las novedades de la versión 1.0.0.beta6 fue la posibilidad de que el reproductor iniciase una canción con datos precargados, como la posición de la canción o el índice que tiene dentro de la lista de reproducción. Para ello, se utilizan objetos PlayConfig, que después se pasan al reproductor como argumento en su creación.

Además de funciones de reproducción podemos encontrar otras dentro del API que proporcionan datos sobre el estado de la reproducción (posición, URI (Identificador de Recursos Uniforme) de la canción o duración en milisegundos entre otros) o la conectividad del dispositivo.

Una vez se ha finalizado la actividad, es importante destruir el objeto Spotify para evitar la fuga de recursos.

2.1.1.3 Sockets

2.1.1.3.1 Introducción

Un *socket* (a veces, también conocidos como *"Internet Socket"*) es un termino usado para identificar un punto de conexión que cualquier *software* puede utilizar para transmitir datos a través de Internet. A través de un *socket*, cualquier dato puede viajar desde un programa (o, en el caso de Android, aplicación) ejecutándose en un dispositivo hasta otro. Usualmente se utiliza una arquitectura de cliente y servidor, y pueden usar tanto el protocolo TCP como UDP.

2.1.1.3.2 Funcionamiento

El funcionamiento de un *socket* se divide en dos partes: el lado del servidor y el lado del cliente.

El servidor únicamente espera, escuchando a través del *socket* a que el cliente haga una petición.

El cliente ha de conocer el nombre de *host* o la dirección IP de la máquina en la cual el servidor se está ejecutando, junto al número de puerto al que está conectado. Una vez conoce estos datos puede realizar la petición de conexión. Si todo va bien, el servidor acepta la conexión y el cliente puede comenzar a comunicarse con él. En el lado del cliente no se utiliza el número de puerto usado para realizar la petición al servidor, sino que se asigna un número de puerto local a la máquina en la que está siendo ejecutado. Tras realizar la transmisión de datos, el *socket* debe cerrarse en ambas máquinas.

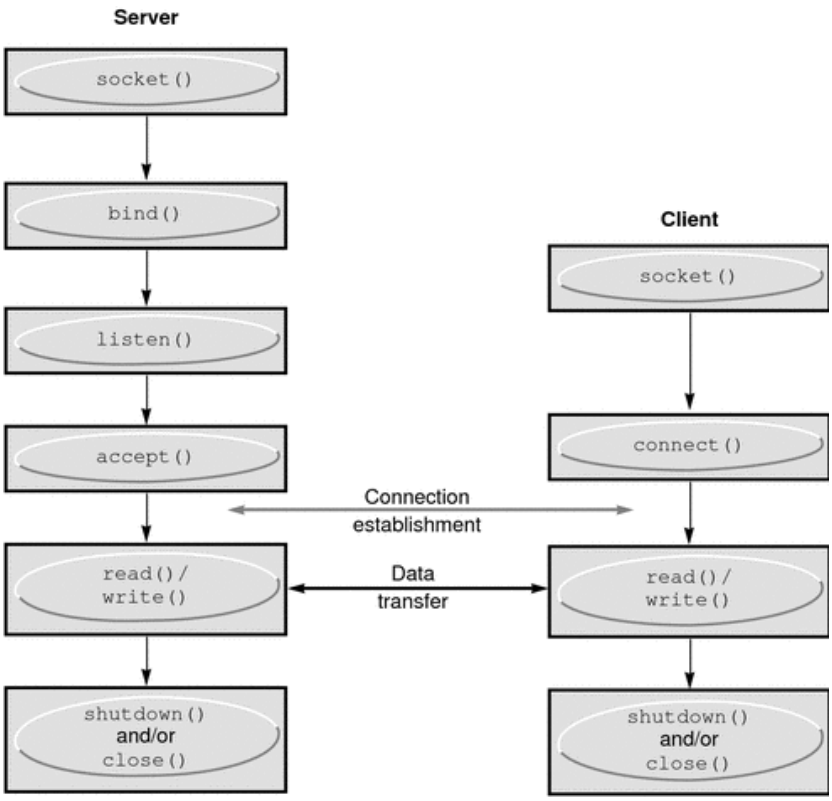

Figura 8: Diagrama de flujo de un socket

Fuente: Oracle

2.1.1.3.3 Sockets en Java y Android

La plataforma Java proporciona una clase Socket, que implementa una de las partes de la comunicación bidireccional entre un programa Java y otro programa en la red. Además, incluye la clase `ServerSocket`, que implementa un *socket* que los servidores pueden utilizar para escuchar y aceptar peticiones de conexiones de clientes. El hecho de que las aplicaciones de Android se programen en Java permite utilizar ambas clases y, por lo tanto, utilizar *sockets* para la comunicación entre distintos dispositivos.

El modelo más simple es el siguiente:

- El servidor establece un puerto y espera a que el cliente establezca la conexión. Cuando el cliente solicite la conexión, el servidor la abrirá con el método `accept()`.
- El cliente establece una conexión con la máquina *host* a través del puerto que se designe.
- Cliente y servidor se comunican mediante manejadores `InputStream` y `OutputStream`.

Desde el cliente, para crear el socket, son necesarios dos parámetros: la dirección IP o nombre de la máquina donde intentamos abrir la conexión y el número de puerto que utiliza el servidor. Ha de tenerse en cuenta que este número de puerto ha de ser siempre mayor a 1023, pues el rango 0-1023 está reservado. Debe tenerse en cuenta que la creación de un *socket* puede crear excepciones, por lo que es una buena práctica su captura.

En el lado del servidor, debemos crear un objeto `ServerSocket`. Para ello necesitamos únicamente el puerto elegido para la comunicación. Una vez se ha creado el *socket*, utilizamos el método `accept()` para quedar a la espera de una conexión.

Cuando la conexión se ha establecido por las dos partes, utilizamos las clases `DataInputStream` y `DataOuputStream` para la lectura y escritura de líneas de texto y tipos de datos primitivos de Java. Es importante conocer qué tipo de datos estamos manejando para realizar una correcta lectura.

Una vez se ha finalizado la transmisión de datos, deben cerrarse los canales de entrada y salida, al igual que los *sockets* creados. El orden de cierre es relevante: primero han de cerrarse los flujos relacionados con el *socket* y después el propio *socket*.

Si la comunicación se va a establecer entre dos dispositivos Android y forma parte del uso de la aplicación pero no es su principal cometido, es altamente recomendable que los sockets se creen en un hilo dentro de la actividad principal, de esta forma la aplicación no quedará bloqueada para el usuario por estar a la espera de la recepción o emisión de datos.

2.1.2 Movilidad de Sesión

La movilidad de sesión puede realizarse a nivel de enlace, red, transporte o aplicación. Dado que la solución implementada en este proyecto realiza esta movilidad a nivel de aplicación, la investigación del estado del arte se ha centrado en ella.

En general, la mayoría de los trabajos proponen la misma regla: Cuando el usuario abandona, su sesión es suspendida. Una vez ha cambiado al nuevo dispositivo, la sesión es reanudada en el nuevo entorno de ejecución. Sobre el momento en el que migrar la sesión se proponen tres alternativas: Migrar después de que el usuario esté en el nuevo entorno, migrar mientras el usuario está realizando el cambio de entorno o migrar inmediatamente cuando el usuario está abandonando el entorno.

Actualmente existen pocos trabajos a nivel industrial sobre el tema, y la gran mayoría de ellos son propios de un fabricante que proporciona movilidad de sesión entre dispositivos de su misma marca o en aplicaciones determinadas.

A nivel académico, gran parte de los estudios se han basado en la movilidad de sesiones SIP o acceso web. La mayoría proponen modificaciones de los protocolos propuestos por [10], basado en SIP, y [11], basado en Web.

Existen dos proyectos académicos que han destacado especialmente y han servido como base al gran parte de los trabajos relacionados: Aura [12] y GAIA [13] ambos centrados en la gestión del contexto.

El primero de ellos, Aura, se propone como objetivos el aprovechamiento de recursos y el mínimo esfuerzo por parte del usuario para movilizar la sesión, creando un entorno inteligente que se adapta a las necesidades del usuario y el contexto en el que se encuentre. Propone la creación de una capa de tareas para representar las intenciones del usuario, disponible para el resto del sistema como base a la adaptación o anticipo de las acciones del usuario.

La movilidad de sesión se consigue usando dos aplicaciones: Proveedores y Conectores. La primera provee la abstracción de datos y la segunda las interfaces de esos datos. Cuando el usuario cambia a una localización diferente, un módulo llamado "observador de contexto" avisa de este evento a un gestor de tareas (bautizado como *Prism*). Este gestor hace una petición a un gestor de entorno para obtener la asignación de un nuevo proveedor de aplicaciones basado en la información de la tarea y el contexto actual. Este nuevo proveedor permitirá al usuario continuar su tarea en el nuevo entorno.

Por su parte, Gaia se enfoca en el problema de la migración de aplicaciones entre diferentes entornos. Su *framework* habilita a una aplicación para adaptar su estructura descomponiéndola en pequeños componentes (modelo, presentación, controladores…). Durante el proceso de migración se toma un *snapshot* del estado de la aplicación y se almacena en un fichero junto a la descripción de la estructura de la aplicación. Cuando el usuario cambia su localización puede reanudar sus tareas seleccionando de forma manual el archivo a cargar. Tras la migración, los componentes de control y presentación pueden ser reemplazados si no están disponibles en el nuevo dispositivo por otros similares.

Al contrario que SuSSo, que es independiente de la infraestructura dedicada, estos dos proyectos se basan en espacios con una infraestructura fija embebida en el entorno que realizan el control por el usuario.

La modificación de estos dos proyectos ha dado pie a varias propuestas que buscan complementarlo, como [14], que centra su investigación en el rendimiento y la adaptación al contexto. Otras, como *ENME* [15] proponen la utilización de RFID (Radio Frequency Identification) para tomar la información del contexto y basan el traspaso de la sesión en SIP.

Centrado en Spotify, la propia empresa lanzó el servicio Spotify Connect [16] en noviembre de 2014. Este servicio permite al usuario utilizar su dispositivo móvil como mando a distancia, dando así una impresión de movilidad de sesión. Sin embargo, su funcionamiento es bastante diferente:

La utilización de Spotify Connect se basa en la elección de una canción, lista o álbum que se quiera enviar a otro dispositivo. Para ello, ha de abrirse la aplicación en un dispositivo móvil, un PC o un dispositivo de sonido especial (varias marcas han lanzado altavoces que lo incluyen). Cuando se inicia la reproducción, el dispositivo móvil nos permite elegir la fuente de reproducción.

La función es gratuita para usuarios Premium y su uso no transfiere la sesión como tal, dado que el usuario debe iniciar sesión en ambos dispositivos. Una vez ha iniciado, el dispositivo móvil le permite elegir dónde continuar la escucha musical, lo que se realiza de forma inmediata, proporcionando así la impresión de transmisión de sesión.

En general, existe poca homogenización entre las soluciones de movilidad de sesión a nivel de aplicación, y la gran mayoría de implementaciones actuales requieren

que cada aplicación implemente directamente su propia función de movilidad. SuSSo, en contraposición, propone los fundamentos para construir un MD-SSO universal para todo tipo de aplicaciones y servicios y permite la movilidad entre dispositivos heterogéneos.

Las investigaciones parecen estar centrándose en la gestión del contexto y la adaptación del modelo de datos al nuevo entorno, especialmente para aquellas aplicaciones de contenido multimedia [17, 14].

Los trabajos que proponen el almacenamiento de datos en la nube también acusan el problema de la privacidad de datos del usuario. Precisamente SuSSO hace hincapié en esta parte, enfocándose en mantener la seguridad de la sesión tras el cambio de dispositivo.

Dada la centralización de las investigaciones en el control del contexto, la mayoría de ellas proponen el traspaso automático de la sesión, sin embargo, para la parte de reanudado, casi todas proponen que sea el usuario quien vuelva a realizar la autenticación en el nuevo dispositivo. SuSSo en cambio propone una única autenticación.

Por último, para finalizar el estudio, en 2013 se publico una comparativa de las soluciones existentes hasta el momento en materia de movilidad de sesión [18]. La siguiente figura, extraída directamente del estudio, muestra la comparativa de las soluciones estudiadas:

Table 5
Comparison of existing proposals (II).

Proposal	Seamlessness	Ubiquity	Smartness	Heterogeneity	Network resilience	Network load	Solution generality	Implementation complexity
RDP	High	High	Low	Low	Low	Low	High	Low
VNC	High	High	Low	High	Low	Low	High	Low
ISR	High	High	Low	Medium	High	High	High	High
VMotion	High	High	Low	Medium	High	High	High	High
SoulPads	High	High	Low	High	High	None	High	High
Aura	Low	Medium	Medium	High	High	Low	Low	Medium
Gaia	Medium	Medium	Medium	High	Medium	Medium	Low	Medium
One.world	High	Medium	Low	Medium	High	Medium	Low	Medium
MDAgent	High	Medium	Medium	High	High	Medium	Low	Medium
Roam	Medium	Medium	Low	High	High	Low	Low	Medium
FollowMe	Low	Medium	Medium	High	High	Low	Low	Medium
Francis' framework	High	Medium	Low	High	High	Medium	Low	Medium
A2M	Medium	Medium	Low	High	High	Low	Low	Medium
ScudOSGi	Low	Medium	Medium	High	High	Low	Low	Medium
iCloud	High	High	Medium	Medium	High	Low	Medium	Medium

Figura 9: Comparativa de soluciones de movilidad de sesión

Fuente: [18]

Como puede observarse, la mayoría consigue una alta transparencia, sin embargo, en la parte de generalidad de la solución existen grandes diferencias entre las propuestas.

2.2 Restricciones y Marco Regulador

En esta sección se describen las restricciones existentes para la creación de esta aplicación y el marco regulador en el que se encuentra.

2.2.1 Restricciones

Las restricciones de la aplicación desarrollada, entendiendo como aplicación a las dos partes (servidor y cliente) vienen dadas por el uso de las tecnologías empleadas.

El sistema operativo Android es reconocido por enmarcarse dentro del *software* conocido como *software* libre, lo que implica que no presenta restricciones. Además es gratuito, por lo que no requiere del pago de licencias.

Por su parte, Spotify depende del SDK de Android para la posible implementación de funciones. En diciembre de 2014 se levantó la restricción de venta de aplicaciones basadas en Spotify, por lo que ya se puede comercializar con estas siempre y cuando cumplan los términos de uso, disponibles en su página web (Véase [2]). A día de hoy no hay requisitos para programar aplicaciones basadas en Spotify, sin embargo, antes del lanzamiento del SDK de Android era necesario registrarse como desarrollador para la programación, lo que implicaba contar con una cuenta *Premium*. No obstante, dado que las funciones de reproducción musical sólo pueden utilizarse con una cuenta *Premium*, es indispensable contar con ella si se quieren realizar pruebas de la aplicación. El coste de este tipo de cuenta es de 9,99€ mensuales.

2.2.2 Marco Regulador

La aplicación necesita de la autenticación del usuario en Spotify, lo cual se puede realizar con su cuenta propia de la aplicación o con su cuenta de Facebook. Esto implica que la aplicación podría servir para identificar personas en aplicaciones comerciales, por ello, se le pide permiso al usuario y se le informa de qué datos utilizará la aplicación en el momento de su instalación.

La autenticación se realiza directamente en el sitio web de Spotify, que cumple con la legislación vigente sobre la Ley Orgánica 15/1999 del 13 de diciembre de Protección de Datos de Carácter Personal (LOPD) que conforma el marco regulador del presente proyecto.

Capítulo 3
Análisis y Diseño de la Solución Técnica

Contenido

En este tercer capítulo se describirá el diseño de la solución técnica elegida. Para ello, se comentarán los requisitos de usuario que debe cumplir, se presentará la arquitectura elegida para su desarrollo y se expondrán dos casos de uso que muestran ejemplos de uso de la aplicación.

3.1 Arquitectura

El artículo en el que se ha basado este proyecto propone cuatro módulos: Gestor de Contexto, Gestor de Estado, Gestor de Comunicaciones y Gestor de Configuración. La adaptación a la aplicación ha seguido esta arquitectura, si bien hay partes de las propuestas en el artículo que no han sido implementadas completamente. Estas partes se dejan como línea de investigación y desarrollo futuro (ver Capítulo 7). Además de esta arquitectura existen elementos externos, como el archivo XML en el cual se guarda la sesión (en el lado del cliente) o se lee para su restauración (en el lado del servidor). La arquitectura puede verse en la siguiente figura:

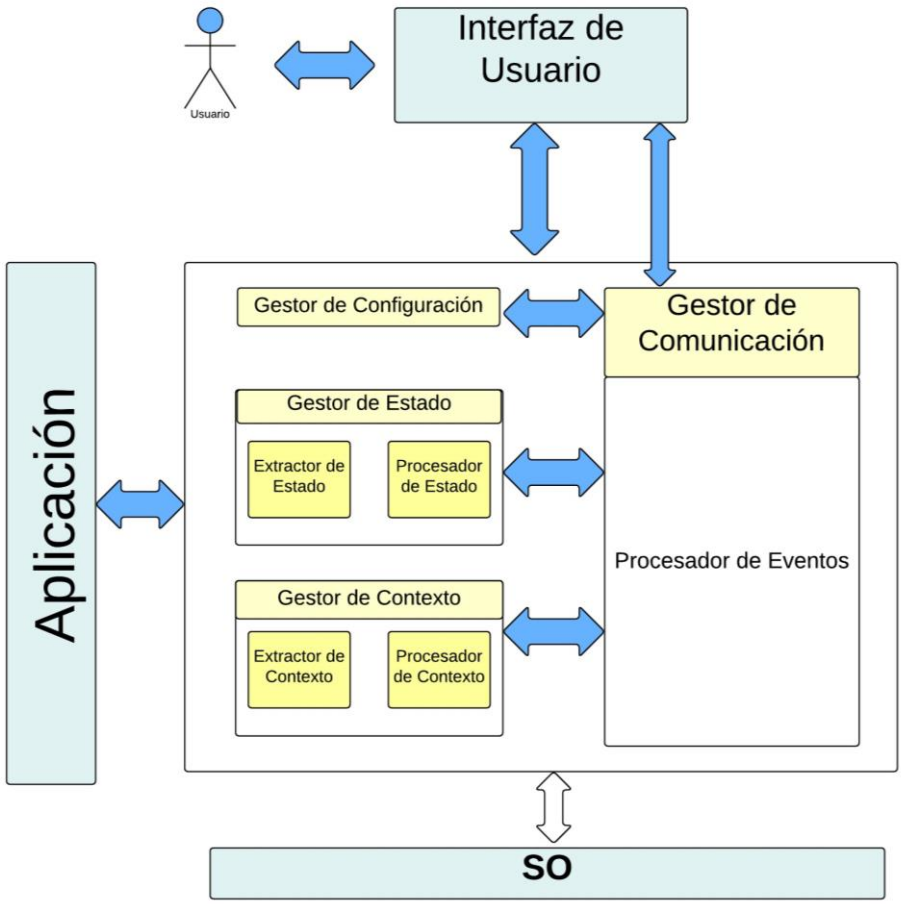

Figura 10: Arquitectura de la aplicación

El artículo en el que se ha basado este proyecto propone dos bloques claros en la implementación: el *framework* SuSSO, que funciona para todo servicio o aplicación, y el *plugin* desarrollado para mover sesiones de una aplicación concreta sobre SuSSo.

El objetivo de implementación era este, sin embargo, debido a las limitaciones de la librería de Spotify utilizada, la implementación real no es tan desacoplada. Por ello, se han creado dos aplicaciones.

La primera de ellas, la aplicación cliente, se encarga de la reproducción musical: Conecta al usuario con su cuenta de Spotify y le permite escuchar música. Cuenta además con un botón de tipo *switch* para tener en cuenta su preferencia de si enviar la sesión o no. También dispone de un botón para cargar la IP del dispositivo al que se va a enviar la sesión. Una vez el usuario pausa la reproducción pulsando el botón *stop*, si cumple ciertas condiciones (envío de sesión activo, disponibilidad de una tarjeta SD de

almacenamiento y conexión a una red WiFi) se genera el archivo con el estado actual de la sesión y se envía al dispositivo destino cuya IP se ha indicado.

La segunda es simplemente un pequeño servidor que está a la espera de la recepción del archivo de sesión. Una vez se recibe este archivo, se lee y se abre la aplicación pertinente para poder restaurar la sesión, que en el caso del que trata este proyecto sería la aplicación anteriormente descrita.

Podríamos identificar la aplicación servidor con el *framework* SuSSo, pues su funcionamiento es independiente de la aplicación cuya sesión se desea restaurar, mientras que la aplicación cliente se podría identificar con el *plugin*, aunque en el artículo original este sólo vendría usado por el gestor de estado, lo cual no ocurre en la implementación real.

Entre los objetivos de [5], se encuentra el de que un usuario pueda guardar varias sesiones, por ello la existencia de esta aplicación servidor, cuyo único cometido es la recepción y apertura de la aplicación necesaria, de forma que el protocolo pueda seguir desarrollándose tomando esto como base. Para cumplir con esta flexibilidad se ha desarrollado un algoritmo de lectura del fichero “universal”: La aplicación servidor lee el archivo en busca de aplicaciones cuya sesión pueda restaurar. Una vez dentro de estas aplicaciones, cada una realiza la lectura exclusivamente de los datos que le corresponden, y una vez guarda los datos necesarios de forma local, los borra del fichero.

[5] propone el restaurado de sesiones múltiples utilizando un fichero por aplicación, sin embargo, para este prototipo se ha considerado también la posibilidad de que un mismo fichero pudiera albergar datos de sesión de varias aplicaciones.

Con respecto a la arquitectura, las funciones de los cuatro módulos, a grandes rasgos, son las siguientes:

Gestor de configuración

Es el módulo responsable de configurar la aplicación y las preferencias del usuario. Engloba la autenticación de usuario, la elección del dispositivo destino (a través de su IP) y la elección por parte del usuario de si enviar la información de la sesión o no.

Está dentro de la aplicación cliente.

Este módulo es el que provee la información al resto dentro de la arquitectura para aplicar las preferencias del usuario.

La autenticación del usuario es obligatoria y debe realizarse con una cuenta *Premium* de Spotify, mientras que la decisión de enviar la sesión o no la realiza el usuario. También el usuario se encarga de introducir la IP del dispositivo de destino.

Gestor de Estado

Su función principal es la de extraer la información del estado de la sesión en un momento determinado. Se divide en dos sub-módulos: Extractor de estado y Procesador de estado.

El **extractor de estado** funciona en el lado del cliente, y es totalmente transparente para él. Una vez ha decidido, a través de la interfaz gráfica, que desea enviar la información de la sesión, cuando termina la reproducción se genera un evento que es capturado por el sistema y se extraen la canción actual (a través de su URI de Spotify), la posición de esta canción dentro de la lista de reproducción, la posición de la reproducción en milisegundos y el resto de la lista de reproducción (guardando cada una de las URIs de las canciones que conforman la lista) como datos de reproducción, y el *token* de sesión para que el usuario no tenga que volver a autenticarse. Estos datos

se encapsulan dentro de un archivo XML, que será enviado al servidor a través del gestor de comunicación previo cifrado..

El **procesador de estado**, al igual que el extractor, funciona en el lado del cliente, y también es transparente para él. El servidor queda a la espera de recibir información de una sesión, y, una vez que esto ha ocurrido, lee la línea del archivo que indica la aplicación de cuya sesión tenemos datos, y abre esta para la continuación de la misma. Una vez se ha abierto, el procesador de estado hace que esta aplicación descifre el fichero y cargue los datos para continuar la reproducción desde el momento en el que se terminó en el dispositivo inicial. Dado que no sólo se guarda la información de la reproducción del momento en el que se terminó, sino que se incluye su posición dentro de la lista y el resto de la lista, el usuario podrá continuar con la misma lista. El guardado del *token* de sesión permite que el usuario no tenga que volver a autenticarse.

Este es el módulo más importante de los cuatro, pues es en el que se basa la aplicación.

Gestor de Contexto

El gestor de contexto es el encargado de la extracción de la información de contexto. El artículo en el que se basa este proyecto proponía extraer el estado de la batería, dispositivos cercanos, localización y tiempo, sin embargo, se ha optado por simplificar esta parte y tan sólo se averigua si el dispositivo está conectado a una red WiFi y el estado de su batería (sólo en el lado del servidor). Si el porcentaje de batería es inferior al umbral adecuado para la recepción de sesión, establecido en el 10% (en el dispositivo con la aplicación servidor), y el dispositivo no está conectado a una red WiFi (en ambos dispositivos), la recepción y restauración de una sesión no serán posibles.

Está estrechamente relacionado con el gestor de comunicaciones, pues esta información es determinante para la comunicación entre dispositivos.

Gestor de comunicaciones

Es la parte de la arquitectura responsable de la comunicación con otros dispositivos. Recibe notificaciones del gestor de contexto para determinar si el dispositivo puede recibir la sesión en la parte del servidor, mientras que en el lado del cliente se comunica con el gestor de estado para elegir el momento en el que realizar la transmisión y con el gestor de configuración para conocer si el usuario desea que esta tenga lugar.

Queda claro pues que los módulos más importantes son el gestor de comunicaciones y el gestor de estado, pues forman la base de la funcionalidad de la aplicación.

Una de las líneas futuras de investigación (ver capítulo 7) es la implementación de un descubridor de servicios, de forma que el propio dispositivo esté dotado de una inteligencia artificial y determine a qué otros dispositivos puede enviar la información de la sesión. De esta manera se evitaría que el usuario introdujera de forma manual la IP del destino.

Para mayor facilidad del usuario, dentro de la interfaz gráfica se muestra la IP del dispositivo, de forma que el usuario solo ha de fijarse en dicho campo en su dispositivo destino para conocer su IP.

3.2 Requisitos

El diseño de la solución se ha basado en el artículo *"SuSSO: Seamless and Ubiquitous Single Sign-on for Cloud Service Continuity across devices"*, de P.A. Cabarcos et al. (véase [5]), adaptándose a las posibilidades que el API de Spotify para Android presentaba. A continuación se detallan los requisitos funcionales (aquellos que definen la función del sistema) y no funcionales (aquellos que especifican criterios para juzgar la operación de un sistema), obtenidos a partir de las especificaciones propuestas para el desarrollo de la aplicación, de las restricciones (capítulo 2, apartado 2.2.1) y de las limitaciones del API disponibles para el servicio de reproducción musical.

3.2.1 Requisitos Funcionales

El principal objetivo de la aplicación es la continuidad de la sesión a través de múltiples dispositivos. Para ello, se han establecido los siguientes requisitos funcionales:

Código	Nombre	Descripción
RF1	Autenticación	El usuario debe realizar su autenticación cuando utiliza la aplicación si no es una sesión restaurada. Si la sesión se restaura, el usuario no se debe volver a autenticar.
RF2	Recepción de eventos	El sistema debe ser capaz de recibir eventos relacionados con la reproducción (inicio de canción, parada, paso a la siguiente canción, etc), clasificarlos y actuar en consecuencia al evento recibido
RF3	Gestión del estado	En el dispositivo origen, deben obtenerse los datos de la sesión antes de su transmisión. Estos datos incluyen, como mínimo, el *token* de sesión y el estado de la reproducción.
RF4	Guardado de datos y generación de fichero	El sistema ha de ser capaz de generar un archivo XML con el estado de la sesión en el formato definido
RF5	Procesado de fichero	El sistema debe poder procesar ficheros XML con datos de la sesión para su posterior restaurado
RF6	Conexión entre dispositivos	Los dispositivos origen y destino deben conectarse entre sí para proceder al envío de datos de sesión. Para ello, ambos deben estar conectados a la misma red WiFi.
RF7	Restaurado automático de sesión	Tras el procesado de fichero, el sistema ha de ser capaz de restaurar la sesión con los datos obtenidos sin interacción por parte del usuario

RF8	Reproducción musical	El sistema debe ser capaz de realizar funciones propias de la reproducción musical. Estas funciones han de generar eventos que se tendrán en cuenta para el estado de la sesión
RF9	Configuración	El sistema debe proveer al usuario de opciones de configuración que habrán de respetarse

Tabla 2: Requisitos Funcionales

3.2.2 Requisitos de restricción

Los requisitos de restricción son aquellos que, de no cumplirse, no permitirían el correcto funcionamiento de ciertas partes de la aplicación

Código	Nombre	Descripción
RR1	Gestión adecuada de acuerdo al nivel de batería	El dispositivo destino debe tener un nivel de batería superior a cierto umbral para poder realizar la restauración de la sesión
RR2	Gestión adecuada de recursos de memoria	Los dispositivos origen y destino deben contar con un sistema de almacenamiento capaz de albergar el fichero XML generado o recibido
RR3	Gestión de conexión a Internet	Los dispositivos origen y destino deben estar conectados a la misma red WiFi para realizar el traspaso de sesión
RR4	Cuenta de usuario	El usuario debe contar con una cuenta *Premium* de Spotify para el uso de la aplicación

Tabla 3: Requisitos de Restricción

Estos requisitos se han establecido por varios motivos:

- Dado que se busca que el usuario cambie su sesión a un dispositivo con mejores condiciones dependiendo del contexto, se ha considerado que no tiene sentido que se restaure la sesión en un dispositivo cuya batería está a punto de agotarse. Por ello, la recepción y continuación de la sesión debe realizarse en un dispositivo cuyo nivel de batería sea mayor a un umbral establecido.
- Teniendo en cuenta que la aplicación hace uso de ficheros, ya sea por la generación de estos en el dispositivo origen o por su recepción en el caso del dispositivo destino, es necesario contar con el espacio suficiente para realizar el guardado de este archivo. Las pruebas de la aplicación muestran que el tamaño del fichero es bastante pequeño (menor a 1Kb), por lo que la restricción es meramente anecdótica. No obstante, el tamaño del fichero aumenta en función del cifrado del mismo y de la cantidad de datos que alberga, por lo que podría llegar a tener un tamaño considerable si se desean restaurar sesiones de múltiples aplicaciones o es necesario un cifrado de alta complejidad.
- Spotify impone que las aplicaciones programadas con los APIs disponibles sólo pueden funcionar con cuentas de usuario de pago, por lo que este requisito es propio de la librería utilizada.

En cuanto al tercer requisito, se ha impuesto por varios motivos:

- El primero de ellos es que la conexión de datos aportada por el operador no permite la utilización del dispositivo como un servidor de datos, por lo que el envío desde una red móvil (ya sea 2G, 3G o 4G) no es posible.
- El segundo de ellos es el ahorro de batería que supone la conexión a una red WiFi frente a una red móvil. Con esta aplicación se busca que el usuario aproveche al máximo la batería restante de su dispositivo, enviando la sesión a un segundo dispositivo que pudiera tener un mayor nivel de batería.
- La tercera restricción es propia de la programación en sockets: para poder realizarse la conexión ambos dispositivos deben estar conectados a la misma red.

3.2.3 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no son estrictamente necesarios para el correcto uso de la aplicación, pero aportan mejoras, ya sea a nivel de facilidad de uso y comodidad para el usuario, seguridad, rendimiento o fluidez.

Código	Nombre	Descripción
RNF1	Centralización en el usuario	El sistema debe ser fácil de utilizar para el usuario, controlado por él y teniendo en cuenta sus preferencias
RNF2	Rendimiento	El rendimiento del sistema en términos de tiempos de respuesta debe ser razonable para asegurar una buena experiencia de usuario
RNF3	Seguridad	La comunicación y manejo de datos del sistema deben tener como prioridad la seguridad
RNF4	Flexibilidad	El sistema debe permitir la interoperabilidad entre plataformas y protocolos heterogéneos y ser capaz de acomodarse a diferentes aplicaciones y servicios
RNF5	Visualización de datos	El usuario debe ser capaz de visualizar datos de la sesión en curso, como la canción en reproducción o notificaciones de traspaso
RNF6	Interfaz Gráfica	El sistema debe contar con una interfaz gráfica sencilla, que responda a acciones, adaptada al dispositivo y de fácil uso para el usuario
RNF7	Apertura Automática de Aplicación	La aplicación servidor debe abrir automáticamente aquella aplicación cliente de cuyos datos dispone en el archivo

Tabla 4: Requisitos no funcionales

3.3 Alternativas de Diseño

Tras la fase de investigación de las funciones necesarias para llevar a cabo este diseño, el siguiente paso fue buscar el diseño más adecuado para la aplicación. Aparte del diseño elegido, se propusieron otras dos alternativas, que se desecharon a favor de la que finalmente se ha llevado a cabo. Estas dos alternativas consistían en la utilización de la aplicación nativa de Spotify y en la creación de una única aplicación que funcionase como cliente y servidor a la vez.

3.3.1 Alternativa 1: Aplicación nativa de Spotify

La primera de las alternativas, y la que más se adapta al *framework* de referencia, era la de utilizar la aplicación nativa de Spotify en lugar de crear una aplicación que reprodujera música.

En esta alternativa se utilizarían los `BroadCastReceivers` que la aplicación envía para capturar el estado de la sesión cuando el usuario pausase la reproducción. Se programaría una aplicación que recibiese estos `BroadcastReceivers` y guardase el estado de la sesión en un archivo XML, justo como se hace en la aplicación finalmente programada y lo enviaría a otro dispositivo. A su vez, o bien esta misma aplicación que guarda el estado podría ser capaz de realizar la recepción y restauración, o bien se programaría otra aplicación exclusiva para ello.

Este diseño no se llevó a cabo porque, pese a que sí era posible realizar el guardado de datos procedentes de la aplicación nativa, el API actual de Spotify no permite la restauración de sesión como se desea, desde el punto en el que se acabó la sesión. Lo máximo que se hubiera podido realizar es restaurar la última canción escuchada (cuya restauración, además, se haría a modo de hipervínculo, por lo que quizás el usuario debería actuar para realizar esta restauración, lo cual no cumple el requisito funcional de restaurado automático de sesión) y que esta empezase desde el principio. Además, el usuario debería volver a realizar su autenticación.

A medida que el API de Spotify vaya creciendo es posible que esta alternativa sea viable.

3.3.2 Alternativa 2: Aplicación cliente y servidor todo en uno

La segunda alternativa era la de crear una aplicación que funcionase a la vez como cliente y servidor. Esta aplicación sería capaz de reproducir música, guardar el estado de la sesión, enviar el archivo, recibirlo y restaurarlo. Para ello, cuando la aplicación se iniciase se crearía un hilo receptor de archivo. Si se recibe el archivo restaura la sesión, si no, el usuario puede elegir utilizar la aplicación como reproductor y enviar la sesión a otro dispositivo una vez finalice. Es decir, esta arquitectura unificaría el *framework* y el *plugin* propuestos por el artículo en un solo bloque.

Esta alternativa sí era viable en su totalidad (de hecho, se creó un pequeño prototipo que ayudó a la comprensión del funcionamiento de los *sockets* en Android y sobre el cual se realizaron algunas de las pruebas más importantes para comprobar el funcionamiento), sin embargo, no cumplía con el requisito no funcional 4: la flexibilidad. Lo que se busca con este trabajo es adaptarse al *framework* propuesto, creando un

concepto universal que funcione entre dispositivos multiplataforma (en este caso, Android) agnóstico al servicio o aplicación que lo utiliza, lo cual queda cubierto con la creación de la aplicación servidor programada, que recibe el archivo de sesión y abre todas las aplicaciones de las que se tienen datos. Con esta alternativa sólo se podrían restaurar sesiones de esta aplicación creada y basada en Spotify, mientras que el hecho de que la aplicación servidor sea capaz de abrir cualquier otra aplicación basándose en los datos contenidos en el fichero de sesión hace que cualquier restauración sea posible, por lo que se desechó la idea de continuar por este camino en favor de la arquitectura que finalmente se utilizó.

3.4 Casos de Uso

En este apartado se utilizarán dos casos de uso que cubren la funcionalidad de la aplicación por parte del usuario. El primero de ellos es el de un usuario transmitiendo una sesión, mientras que el segundo se basa en la recepción y restauración.

3.4.1 Transmisión de sesión

En este caso de uso, el usuario abre la aplicación para reproducir su lista musical y enviar la sesión al siguiente dispositivo:

CU1 Transmisión de sesión	
Actores	Usuario
Descripción	Envío de la información de sesión a un dispositivo destino en el cual se continuará con la reproducción
Flujo	1. El usuario abre la aplicación cliente y comienza la reproducción musical. 2. El usuario activa la posibilidad de enviar la sesión a otro dispositivo 3. El usuario introduce la IP del dispositivo destino. 4. El usuario pulsa el botón *stop*. Inmediatamente, se extrae la información del estado de la sesión, se encapsula y guarda en un fichero que se envía al dispositivo destino. 5. El usuario es avisado del correcto envío de sesión o la ocurrencia de un error.

Tabla 5: Caso de Uso 1: Transmisión de sesión

3.4.2 Recepción y restauración de sesión

El segundo de los casos de uso consiste en la recepción y restauración de la sesión en el dispositivo destino.

CU2 Recepción y restauración de sesión	
Actores	Usuario
Descripción	El dispositivo destino continúa la sesión donde el usuario finalizó en el dispositivo origen
Flujo	1. El usuario abre la aplicación servidor en el dispositivo destino 2. Si se cumplen las condiciones, el dispositivo destino queda a la espera de la recepción de una sesión. 3. Una vez se ha recibido el archivo con los datos de la sesión, la aplicación servidor lee de qué aplicaciones tiene datos y realiza su apertura. 4. La aplicación realiza el des-encapsulado de datos y los guarda de forma local. 5. A partir de estos datos, la reproducción comienza de forma automática.

Tabla 6: Caso de uso 2: Recepción y restauración de sesión

Capítulo 4
Implementación del sistema

Contenido

A lo largo de este capítulo se mostrará como se ha implementado la solución técnica descrita en el capítulo anterior. Para ello, se hará la división de implementación por aplicaciones y, dentro de cada aplicación, cada uno de los módulos propuestos en la arquitectura. Previo a estas explicaciones se muestra un diagrama de flujo de la aplicación para su mejor comprensión.

4.1 Aplicación Cliente

La aplicación cliente tiene varias funciones: La reproducción musical, la creación del archivo con los datos de la sesión y el envío de esta. Además, debe ser capaz de la restauración de la sesión. A lo largo de este apartado se explicará cómo se ha implementado cada una de estas funciones. Previamente, se muestra un diagrama de flujo para una mejor comprensión.

4.1.1 Diagrama de flujo

En la siguiente página se muestra el diagrama de flujo de la aplicación cliente. Cabe destacar que, dado que es una aplicación que depende de las acciones del usuario, el diagrama de flujo no puede ser lineal. Por ejemplo, el usuario podría cambiar de canción, activar o desactivar el *switch* de envío de sesión o introducir la IP del servidor en cualquier momento. El diagrama muestra el uso más básico de la aplicación, sin tener en cuenta este tipo de acciones.

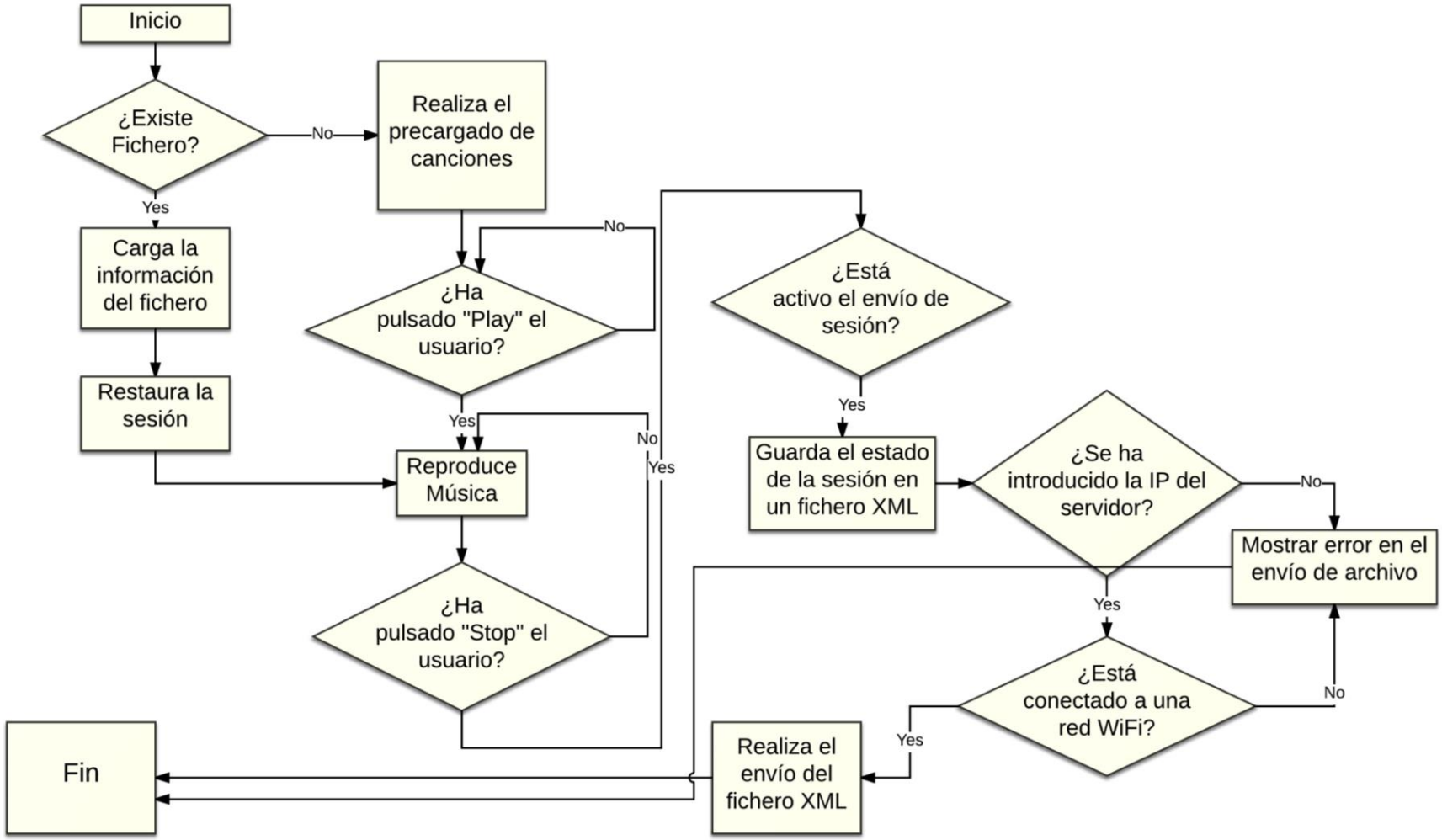

Figura 11: Diagrama de flujo de la aplicación cliente

4.1.2 Aspectos Generales

La aplicación cliente consta de tan sólo una actividad. Esta a su vez consta de dos partes: La interfaz gráfica de usuario y la lógica interna.

La aplicación se ha programado pensando siempre en la comodidad del usuario y la facilidad de uso para éste. Por ello, la interfaz de usuario se ha simplificado lo máximo posible, sin dejar de lado todas las funcionalidades que la aplicación presenta.

Pese a la simplicidad de la interfaz de usuario, la lógica interna es dónde reside el verdadero potencial de la aplicación. En esta lógica interna se han cubierto todos los módulos de los cuales consta la solución.

Cabe destacar que la aplicación servidor y la aplicación cliente han de relacionarse entre ellas para poder cubrir todas las funcionalidades de las que trata el protocolo.

4.1.3 Interfaz de Usuario

Esta aplicación depende estrechamente de las acciones que el usuario decida. Por ello, su interfaz de usuario debe ser capaz de darle a elegir todas las opciones que la aplicación es capaz de realizar. Esta interfaz es, en cierta manera, la que controla al resto de módulos dentro de la aplicación cliente.

La interfaz de usuario se ha diseñado gráficamente de forma que no sea complicada de utilizar para el usuario, que siempre se ha tenido como prioridad (como establece el requisito no funcional 1). Este diseño consta de un fondo de pantalla y varios botones de reproducción, así como un botón de tipo *switch* para que el usuario decida si quiere realizar el envío de sesión y un botón para introducir la IP del servidor al que se enviará la sesión.

Debido al uso de una imagen como fondo, se ha forzado a que la aplicación se ejecute sólo con el dispositivo en posición vertical.

Figura 12: Interfaz de Usuario de la aplicación cliente

Debido al gran número de dispositivos que funcionan con Android y la gran diferencia en sus tamaños y resoluciones, para adaptar esta interfaz se ha utilizado un método de escalado automático en función del tamaño de pantalla. Este método se ha adaptado de una publicación de la empresa Vanteon Electronic Design, cuya autoría pertenece a Aaron Sher [20]. El escalado de la interfaz se realiza únicamente la primera vez que la ventana está enfocada.

La interfaz muestra la IP del dispositivo para conocimiento del usuario. Cuando se está reproduciendo música, la interfaz también muestra qué canción es la que se está escuchando.

En los siguientes apartados se describirá cada uno de los módulos de la aplicación, sin embargo, ha de destacarse la relación que la interfaz gráfica tiene con cada uno de ellos: El *switch* de elección de envío de sesión permite el funcionamiento del extractor de estado (para el guardado del estado de la sesión) y del gestor de comunicación. A su vez, pese a que sin esta activación el extractor de estado no queda activo, es el botón de *Stop* el que hace que funcione y active el mecanismo de guardado y envío de sesión.

La interfaz de usuario se actualiza de forma constante, por lo que se han implementado varias funciones única y exclusivamente para ello:

- Cada vez que el usuario pulsa el botón *Next* o una canción termina se comprueba que no está reproduciendo la última canción de la lista. De ser así, este botón desaparece. Lo mismo ocurre con el botón *Previous* y la primera canción de la lista.
- Cuando se inicia la reproducción o la canción cambia, bien por su finalización o bien porque se han pulsado el botón *Next* o el botón *Previous*, se cambia el título de la canción para adaptarlo a la que está sonando en ese momento.

4.1.4 Lógica Interna

4.1.4.1 Reproducción musical

La base de la aplicación cliente es la reproducción musical. Debido a las limitaciones del API de Spotify, la lista de canciones elegidas debe estar precargada, es decir, el usuario no puede elegir qué canciones escuchar. Tampoco puede acceder a las listas de reproducción musical guardadas en su cuenta.

A nivel de reproducción se ha creado una lista de canciones y se ha dotado a la aplicación de las funciones de reproducción, parada, pasar a la siguiente canción y pasar a la canción anterior.

Para la utilización de estas funciones se han usado métodos propios del API de Spotify para Android. Estos son el método `play()`, `pause()`, `skipToNext()` y `skipToPrevious()`.

De estos métodos, el más interesante es el método `play()`, el cual puede reproducir la lista de reproducción de forma lineal o con ciertos parámetros, lo cual es especialmente útil para el restaurado de sesión. Para hacer esto se utiliza un objeto de tipo `PlayerConfig`, que permite indicar cuál será la primera canción que se reproducirá a través de su índice en la lista y su posición en milisegundos. Una vez se han cargado estos datos, se utiliza este configurador para la reproducción (utilizando una variante del método, `play(PlayerConfig pConfig)`).

Debe destacarse que en el inicio de la aplicación la primera función que se utiliza es la de cargar canciones. Para ello, primero se busca si existe un fichero con datos de sesión o no. En el caso de que exista, se cargarán los datos de este fichero (*token* de sesión incluido), mientras que en caso contrario se cargarán las canciones y se esperará a que el usuario inicie la reproducción.

4.1.4.2 Gestor de Configuración

El gestor de configuración es el módulo responsable de la configuración de la aplicación y controlar las preferencias del usuario.

La primera información que se encarga de obtener este gestor es la autenticación del usuario. Para ello, cuando la aplicación se inicia (sin restaurado), se redirige a una ventana del explorador de internet predeterminado del dispositivo donde el usuario puede introducir sus credenciales de Spotify. La autenticación puede realizarse bien con la cuenta de Spotify o bien a través de Facebook, siempre y cuando la cuenta de Spotify esté conectada a Facebook. Recalcar una vez más que el funcionamiento depende del tipo de cuenta: Sólo será correcto si la cuenta es *Premium*.

Dentro de la librería de Spotify para Android existe un paquete exclusivo para la autenticación, del cual se ha hecho uso. Concretamente, el método utilizado es `openAuthWindow()`, encargado de redirigir al usuario a la página de autenticación. Este método se utiliza dentro de la función `onCreate()` propia de Android, la que se ejecuta junto a la apertura de la aplicación.

Cuando la autenticación ha terminado, el servicio de cuentas de Spotify devolverá el foco a la aplicación a través de la URI de redirección que se indicó en el sitio *myApps* de Spotify, la cual ha de indicarse como parámetro de este método. El resultado se devuelve en un `Intent`, por lo que es necesario utilizar el método `onNewIntent()` de Android. Una vez se ha recibido y gestionado ese `Intent` se crea un objeto de tipo reproductor con el *token* de autenticación. Este *token*, en la versión de la librería 1.0.0-beta6 (la utilizada para la realización del proyecto), tiene una duración

de una hora. La versión 1.0.0-beta7 introdujo leves mejoras con respecto a la autenticación, entre ellas la mayor disponibilidad temporal. La versión 1.0.0-beta8 incluye una función de *login* que no necesita abrir el explorador de Internet.

Si la aplicación funciona en modo restauración, el usuario no deberá autenticarse, pues el sistema es capaz de extraer el *token* de sesión guardado en el fichero de datos de sesión. Para su carga se ha hecho uso del método `setAccessToken()` de Spotify.

La segunda información que recaba este gestor es la preferencia del usuario sobre el envío de sesión. Para ello, la interfaz gráfica cuenta con un botón de tipo *switch* en el cual el usuario decide si el envío se realizará o no. Este botón además determina si la sesión se guardará o no. La decisión de no guardar la sesión si el botón no estaba marcado se tomó por aprovechamiento de recursos.

Además del marcado del botón, para poder realizar el envío de sesión es necesario que el usuario esté conectado a una red WiFi (el gestor de contexto se encarga de esta comprobación) y que haya introducido la IP del servidor al cual se quiere enviar la sesión.

Si esta IP no está introducida en el momento de marcar el botón como activo (por defecto está inactivo), el usuario recibirá un aviso en forma de ventana emergente indicando que debe hacer esto. En el caso de que el usuario pulse el botón *Stop* y aún no haya introducido la información de la IP del servidor volverá a recibir un aviso indicando que el envío no se puede realizar por la falta de información.

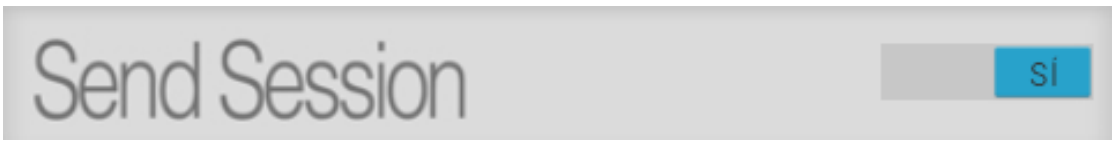

Figura 13: Switch de envío de sesión en la interfaz gráfica

Es obvio pues que es este gestor quien se encarga de conocer si se ha introducido la IP del servidor.

Para introducir esta IP, el usuario debe pulsar el botón pertinente en la interfaz de usuario. Una vez lo ha pulsado aparecerá una ventana emergente pidiendo que introduzca dicha IP. Si el usuario no conoce dicha IP basta con que abra una de las aplicaciones, servidor o cliente, en el dispositivo destino, pues en ambas se muestra la IP del dispositivo en la interfaz gráfica. Si esta información queda en blanco, la sesión no podrá ser transferida a ningún dispositivo.

Figura 14: Ventana emergente para introducción de la IP del servidor en interfaz gráfica

En resumen: las funciones del gestor de configuración son la autenticación del usuario, la comprobación de la elección de envío de sesión por parte del usuario y la comprobación de la introducción de la IP del servidor.

Se relaciona directamente con la interfaz gráfica debido a que es en esta donde el usuario marca sus preferencias ([5] propone tres medios para tener en cuenta las políticas de configuración: la elección del usuario a través de la interfaz gráfica, basarse en la historia de las transacciones y construirlas de forma dinámica o un híbrido de ambas).

También se relaciona con el gestor de estado, pues este necesita que el usuario haya elegido el envío de la sesión para poder realizar la extracción y guardado de datos, y con el gestor de comunicación, dado que si no se ha introducido la IP del servidor no se podrá realizar el envío.

A su vez, sin la autenticación ninguna de las funciones de la aplicación está disponible, incluyendo esto la reproducción musical.

4.1.4.3 Gestor de Estado

La principal función de este gestor es la obtención del estado de las aplicaciones que están funcionando en el dispositivo origen para su posterior traspaso al dispositivo destino, así como la lectura de los archivos con la información de la sesión para el restaurado de esta.

Teniendo en cuenta su función, es obvio que este gestor tendrá relación directa con el gestor de comunicación, tanto en la parte de envío del estado como en la de restauración de sesión.

Este módulo se divide en dos sub-módulos: El extractor de estado y el procesador de estado. El primero de ellos se encarga de la obtención de datos de sesión, mientras que el segundo se ocupa de la restauración.

A continuación se describirán por separado cada uno de estos sub-módulos.

4.1.4.3.1 Extractor de Estado

El extractor de estado cumple la función de obtener los datos de la sesión en curso en un momento determinado. Este submódulo se implementa en la aplicación cliente.

Para la obtención del estado se han programado varias funciones:

- Un gestor de eventos para decidir cuándo guardar estos datos.
- Una función encargada de la obtención de este estado cuando el gestor de eventos reciba un evento determinado
- Una función para el guardado de este estado en un String con formato de fichero XML.
- Una función de cifrado de los datos de sesión
- Una función para guardar este String cifrado en un fichero .xml que se almacenará en el dispositivo previo a su envío al dispositivo destino.

Para un mejor entendimiento de estas funciones, a continuación se describen una a una:

4.1.4.3.1.1 Gestor de Eventos

Spotify tiene varias formas de notificar los eventos en Android. La primera de ellas es desde su aplicación nativa a través de `BroadcastReceivers`. La segunda es para aplicaciones implementadas a través del Android SDK, para lo cual ha creado la interfaz `PlayerNotificationCallback`. Dado que esta aplicación está creada a partir del Android SDK, se ha hecho uso de los métodos de esta interfaz.

Concretamente, el método utilizado es `onPlaybackEvent (EventType evenType, PlayerState playerState)`, que es capaz de recibir los eventos que se producen en el reproductor pasado como parámetro. Previo a la implementación de este método ha de añadirse un gestor de eventos al objeto de reproducción, a través del método `addPlayerNotificationCallback()`.

Spotify es capaz de notificar los siguientes eventos:

`AUDIO_FLUSH` Request to flush the audio buffers
`BECAME_ACTIVE` The current device became the active player (related to Spotify Connect)
`BECAME_INACTIVE` The current device is no longer the active player (related to Spotify Connect)
`END_OF_CONTEXT` End of context was reached
`EVENT_UNKNOWN` Unknown event
`LOST_PERMISSION` Playback was started on another device
`PAUSE` Playback was paused
`PLAY` Playback has started
`REPEAT_DISABLED` Repeat was disabled
`REPEAT_ENABLED` Repeat was enabled
`SHUFFLE_DISABLED` Shuffle was disabled
`SHUFFLE_ENABLED` Shuffle was enabled
`SKIP_NEXT` Skip to the next track
`SKIP_PREV` Skip to the previous track
`TRACK_CHANGED` A track change event was received

`TRACK_END`	Song finished playing
`TRACK_START`	Song started playing

Tabla 7: Eventos de Spotify

Fuente: [3]

Para esta aplicación se ha hecho uso de los eventos `SKIP_NEXT`, `SKIP_PREV` y `TRACK_END`, los cuales se utilizan para control de la interfaz de usuario, y el evento `PAUSE`, que es el núcleo de este extractor de estado.

El estado de la sesión se guarda cuando el usuario pulsa el botón *stop*, lo cual hace que se reciba un evento de tipo `PAUSE`. En este momento, dentro del método `onPlaybackEvent()`, se guarda la canción actual, su posición en milisegundos y se busca su posición dentro de la playlist en la que se encuentra. Estos tres parámetros se guardan en tres variables, `currentTrack`, `position` e `index`.

Además del guardado de parámetros, es en este método donde se realizan las funciones más importantes de la aplicación: Una vez salvados y cifrados, se guardan los datos en el archivo XML y se procede al establecimiento de la comunicación. Para ello, primero ha de averiguarse si se ha cargado la IP del servidor, después, si estamos conectados a una red WiFi y, por último, proceder al envío.

4.1.4.3.1.2 Guardado del estado

Una vez se ha obtenido el estado de la forma que se ha explicado anteriormente, se procede a su salvado en un fichero con extensión .xml. Para ello se han utilizado cuatro funciones: Un método que guarda los parámetros en un *String*, otro para cifrarlos, otro que guarda este *String* en un fichero y uno auxiliar para centralizar a ambos.

El método encargado de guardar los datos en un *String* lleva por nombre `createSessionContent()`. Este método simplemente genera un *String* con el formato especificado por el artículo para el fichero XML con los datos de la sesión.

Este fichero se compone de dos etiquetas: `<AppName>` y `<AppState>`. La etiqueta `<AppName>` se utiliza para indicar el nombre de la aplicación cuyos datos se están salvando, mientras que los datos como tal aparecen dentro de `<AppState>`. El contenido de esta etiqueta dependerá de qué aplicación es la que está guardando datos y qué datos se necesitan guardar teniendo como objetivo la restauración.

Para esta aplicación se han guardado los siguientes datos:

1) Paquete al que pertenece la aplicación. Utilizado para la posterior restauración, pues permite abrir de forma automática la aplicación donde se restaurarán los datos.

2) *Token* de la sesión. Imprescindible para su restauración.

3) Número de canciones en la lista

4) Canción Actual. Guardada en formato URI de Spotify.

5) Posición actual. Guardada en milisegundos

6) Índice de la canción actual en la lista de reproducción

7) Canciones en la *playlist*. Lista con la URI de cada una de las canciones que componen la lista de reproducción.

```
<?xml version="1.0" encoding="utf-8"?>

<AppName='Spotify'>
<AppState>
<Package>
tfg.susso2
</Package>
<Token>
BQDXsSgnp4TqhtWeEIsyPEepXS5jPKnF4SCicfTxmPzdm-UiPKpK9XZNODwMadY0casr07-
    ilJhy61gJM0MLlBgVVVqAN00PRHV4v09gzC7H9aqtfEBgWNiXawcRluc4f6GJ18Qb4Wyx1g5zitaFLR4tbl7yJqFxdlQ
</Token>
<Songs number>
8
</Songs number>
<Current Track>
spotify:track:4VrWlk8IQxevMvERoX08iC
</Current Track>
<Current Position>
15534
</Current Position>
<Current Index>
3
</Current Index>
<Songs in the playlist>
spotify:track:7GBGXDmYOTHuBtPfRZc6Vu
spotify:track:7JeYK1pt9Kds0E62S7uiIU
spotify:track:7GOAPpHfg8yb2jGxMIOx2y
spotify:track:4VrWlk8IQxevMvERoX08iC
spotify:track:78zp5yTpZWdrBrC1LAEP3c
spotify:track:73JwUp9K4GYUNnqz5onncr
spotify:track:03m03OVQGPuzt8Rw2r0a7t
spotify:track:0PjkFSNBbpfYRYpAG4uuGb
</Songs in the playlist>
</AppState>
</AppName>

```

Figura 15: Archivo XML (no cifrado) con datos de sesión

Una vez se ha generado este *String*, se hace uso de una clase con métodos de cifrado para que su guardado no sea en claro (se ha utilizado cifrado DES con clave de 128 bits) y se utiliza el método `writeToFile(String fileName, String fileText)` para guardarlo a un fichero en el directorio local.

Este método primero crea el directorio donde se guardarán los ficheros en la tarjeta SD del dispositivo. A este directorio se le ha dado el nombre "*Session_Files*", y su ubicación dentro del dispositivo es `sd_card/Session_Files`.

Una vez creado el directorio, se crea un fichero con el nombre `Session.xml`, se escribe sobre él el contenido y se guarda en el directorio.

Con vistas a la utilización por parte de varias aplicaciones de este fichero, en lugar de escribir un fichero nuevo primero se comprueba si ya existe. En caso afirmativo, se añade a este fichero el contenido de la sesión de la aplicación. Por si ya existieran datos de la aplicación en curso, se buscan estos y se borran. Este borrado se realiza fácilmente gracias a la etiqueta `<AppName>`. Cuando el fichero queda libre de todo dato de sesiones anteriores de la aplicación se procede a añadir los datos de la sesión actual a los que ya existan de otras aplicaciones en el fichero.

4.1.4.3.2 Procesador de Estado

El procesador de estado se encarga de la restauración de la sesión a partir de los datos obtenidos en el fichero. Para ello, una vez se ha recibido un archivo, la aplicación servidor abre la aplicación cliente, que lee este archivo descifrándolo previamente, crea un reproductor con los datos obtenidos en él e inicia la reproducción a partir del punto indicado.

La aplicación cliente comienza realizando la carga de la lista de reproducción, para lo cual ha de averiguar si existe un fichero de sesión o no haciendo uso del método `thereIsFile()`, que devuelve un valor *booleano* sobre la existencia del archivo. En caso afirmativo, llama al método `chargeFile()`, encargado de obtener los datos de este fichero y, por lo tanto, parte central del procesador de estado. Para el caso negativo, simplemente se cargan las canciones y se espera a que el usuario inicie la reproducción.

Dentro del método de carga de fichero se abre el fichero recibido, se descifra y se busca la etiqueta `<AppName>`. Gracias a ella, el método sólo se detendrá a leer con detalle los datos contenidos dentro de la aplicación que nos concierne. Cuando encuentra esta etiqueta y el nombre coincide con el de la aplicación, lee línea a línea cada una de las cabeceras y guarda en los atributos de la clase los parámetros leídos. Como ya se indicó en el punto anterior, el fichero XML guarda los datos de número total de canciones en la lista, canción actual, posición en milisegundos, posición de la canción dentro de la lista de reproducción y canciones de la lista de reproducción junto al *token* de sesión. Una vez se han leído estos datos, el método los borra del fichero, impidiendo así que la sesión se restaure dos veces pero respetando la posible existencia de datos de sesión de otras aplicaciones.

Tras realizar el restaurado del *token* de sesión y la carga de canciones se crea un reproductor con la lista cargada, y se le indica, a través del índice de canción leído en el fichero y la posición guardada, en qué canción debe iniciar la reproducción y en qué instante. La reproducción comienza de forma inmediata, sin actuación por parte del usuario.

4.1.4.4 Gestor de Contexto

El gestor de contexto se encarga de la extracción de la información contextual del dispositivo. [5] propone, como información de contexto, datos como la batería restante del dispositivo origen, dispositivos próximos a él, localización actual o tiempo.

En este caso, para la aplicación cliente, sólo se ha tenido en cuenta el tipo de conexión del dispositivo. La sesión sólo podrá transferirse si el dispositivo está conectado a una red WiFi.

Para realizar esta comprobación simplemente se hace uso de un objeto de tipo `ConnectivityManager` que comprueba la información del contexto del dispositivo.

Junto a esto, el resto de información de contexto que se ha extraído es la dirección IP del dispositivo (que se muestra en la interfaz gráfica), pero tan sólo se utiliza de forma informativa. Además, ha de comprobarse que la IP del servidor se ha introducido previo al envío de la sesión.

4.1.4.5 Gestor de Comunicación

Junto al gestor de estado, el gestor de comunicación es el otro gran bloque importante de la aplicación.

Este gestor controla la comunicación con otros dispositivos. Recibe información del gestor de contexto cuando una transferencia de sesión es posible y se encarga de su realización.

Para su inicialización es indispensable que se cumplan todas las condiciones que se han ido mencionando a lo largo de este documento: Conexión a una red WiFi, *switch* de envío de sesión activo e IP del servidor introducida en el campo correspondiente. Si todo esto se cumple, el envío será posible.

La realización del envío ocurre cuando se ha generado el archivo xml con los datos de la sesión. Tras las comprobaciones de los datos anteriormente mencionados, se inicia un hilo distinto al de la actividad principal que realizará la comunicación.

El puerto elegido para la comunicación es el 7266, que cumple la condición para la comunicación entre sockets de ser mayor que 1024. Cuando el hilo se inicia, se crea un nuevo *socket* en este puerto y, si el servidor está activo y esperando, aceptará la conexión y se iniciará la transferencia. El fichero a enviar se indica por parámetro en la creación del *socket*.

Finalizada la transferencia del fichero, se cerrará el *socket* y se mostrará un aviso en pantalla del envío correcto del fichero, informado al usuario de que ya puede cerrar la aplicación si así lo desea. En el caso de que el envío no sea correcto también se avisará al usuario, sin embargo, no se exponen los motivos de este fallo.

Este gestor forma parte de las dos aplicaciones, cliente y servidor. En el próximo apartado de este mismo capítulo se explicará cual es el funcionamiento en el lado del servidor, encargado de la recepción del archivo.

La comunicación es totalmente transparente para el usuario una vez ha pulsado el botón stop. La única acción que debe realizar el usuario para que este módulo pueda funcionar de forma correcta es la introducción de la IP del servidor al que desea enviar el fichero.

4.2 Aplicación Servidor

La aplicación servidor es la encargada de la recepción del archivo con los datos de la sesión de la aplicación cliente. Su implementación es mucho más simple que la de esta, sin embargo, su función es indispensable para el buen funcionamiento del sistema completo.

De los cuatro módulos antes expuestos, la aplicación servidor sólo cuenta con dos: el gestor de comunicación y el gestor de contexto. A su vez, la interfaz gráfica es mucho más sencilla y no necesita de las acciones del usuario.

Esta aplicación podría estar en segundo plano y su funcionalidad sería completa de la misma manera.

A lo largo de este apartado se describirán cada una de las partes de las que está compuesta y cómo se han implementado.

4.2.1 Diagrama de flujo

En este apartado se muestra el diagrama de flujo que clarifica el funcionamiento de la aplicación servidor.

Al contrario que en la aplicación cliente, la aplicación servidor tiene un funcionamiento mucho más simple y lineal que no depende de las acciones realizadas por el usuario, por lo que su diagrama de flujo es mucho más simple.

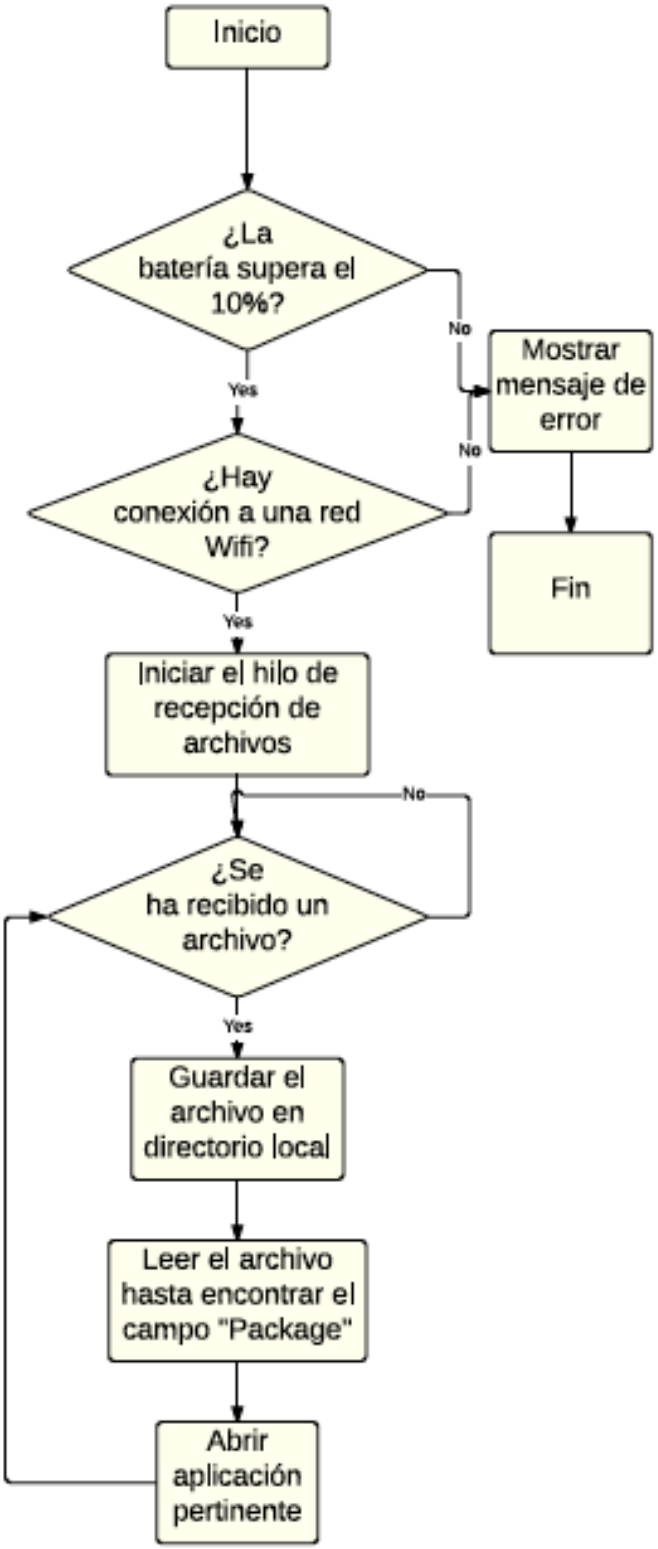

Figura 16: Diagrama de flujo de la aplicación servidor

4.2.2 Aspectos Generales

La aplicación servidor, al igual que la aplicación cliente, consta de tan sólo una actividad. Una vez más, está compuesta por dos partes: La interfaz gráfica de usuario y la lógica interna.

Es en la lógica interna donde la aplicación tiene su verdadero potencial. Esta aplicación constituye gran parte del gestor de comunicación.

La aplicación se relaciona directamente con la aplicación cliente, encargada del envío del archivo de sesión que esta recibe. Una vez recibido, lee la cabecera y abre la aplicación cliente.

La lectura de cabecera se realiza debido a la universalidad del protocolo. [5] propone la creación de una tabla de mapeado de aplicaciones cuya sesión es capaz de restaurar, sin embargo, para este prototipo, se ha realizado la apertura de aplicación utilizando el nombre del paquete al que pertenece.

Al contrario que la aplicación cliente, que es muy específica para Spotify, esta aplicación servidor podría ser válida para cualquier aplicación en la cual se quiera implementar el *framework*.

4.2.3 Interfaz de Usuario

La interfaz de usuario de esta aplicación es mucho más simple que la de la aplicación cliente. Dado que no necesita de la interacción del usuario para su funcionamiento, la interfaz de usuario sólo consta de una imagen de fondo en la cual se muestra la IP del dispositivo. Esta IP se ha indicado aquí para facilidad del usuario en el momento que tenga que introducir la IP en la aplicación cliente para el envío de sesión.

Dado que la interfaz está basada en una imagen, se ha forzado a que la utilización de la aplicación sólo sea posible con el dispositivo en posición vertical. Esto también se realizó en la aplicación cliente por el mismo motivo.

La siguiente figura muestra el fondo utilizado para la aplicación. Justo en el centro de la imagen, el motor de la aplicación extrae la IP del dispositivo y la muestra en pantalla. Si la sesión no se pudiera restaurar por no cumplirse las condiciones de contexto, se muestra un mensaje de error en color rojo justo debajo de la IP, advirtiendo al usuario de que la restauración no será posible.

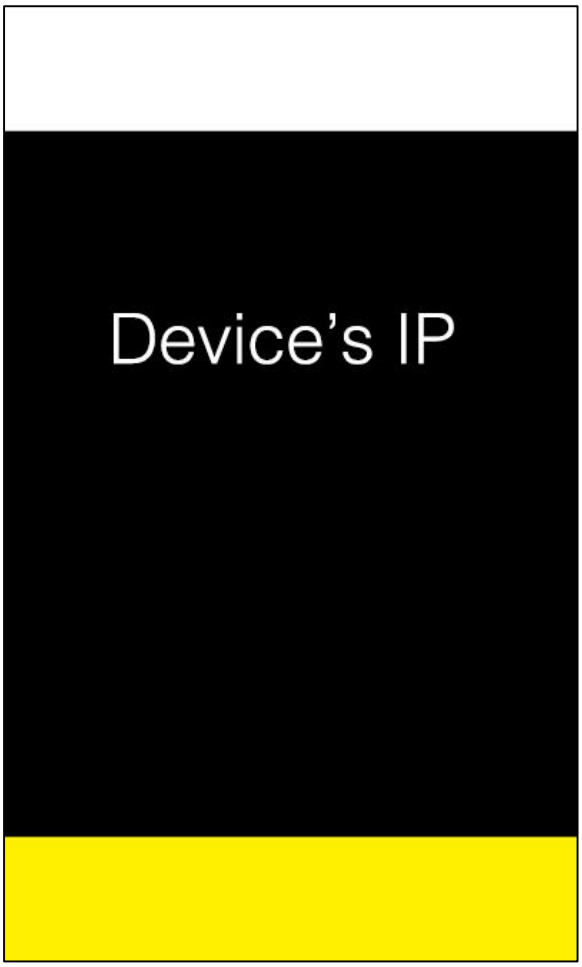

Figura 17: Interfaz de usuario de la aplicación servidor

Al igual que en la aplicación cliente, esta interfaz de usuario utiliza un método de re-escalado para adaptarse a todas las pantallas y resoluciones que presentan los múltiples dispositivos Android.

4.2.4 Lógica Interna

La lógica interna de esta aplicación realiza la recepción del archivo y la apertura de la aplicación que será capaz de des-encapsular los datos que en él aparezcan para el restaurado de sesión. De los cuatro módulos de la arquitectura, la aplicación está compuesta por dos: El gestor de contexto y el gestor de comunicación. Es este último el realmente importante de los dos.

A continuación se describirá cómo se ha realizado la implementación de estos dos módulos y cuales son las funciones más importantes dentro de ellos.

4.2.4.1 Gestor de Contexto

La restauración de la sesión tiene dos requisitos: que el nivel de batería sea superior al 10% y que el dispositivo esté conectado a una red WiFi. Esta información es aportada por el gestor de contexto.

Se ha considerado que la batería debe ser superior al umbral indicado porque se ha considerado que si el usuario está cambiando la sesión de dispositivo es porque desea continuar con ella. Por ello, el dispositivo destino debería contar con las condiciones óptimas para un largo uso.

En el caso de que el nivel de batería no pueda ser extraído, se considera que el dispositivo está totalmente cargado.

La implementación de este gestor es bastante simple: basta con un objeto `ConnectivityManager` para averiguar si el dispositivo está conectado a una red WiFi y un filtro para capturar el nivel de batería.

4.2.4.2 Gestor de Comunicación

La parte del gestor de comunicación de la aplicación servidor consiste en un hilo que queda a la espera de que se establezca la comunicación desde la aplicación cliente. El puerto elegido para esta comunicación es el 7266. Este hilo se ejecuta infinitamente, de forma que se pueden recibir archivos de sesión sin tener que abrir y cerrar la aplicación cada vez que una comunicación ha sido establecida.

Este hilo sólo comienza su actividad si el nivel de batería es superior al 10% en el dispositivo y si este está conectado a una red WiFi. Si se cumplen estas condiciones, el hilo comienza su actividad y queda a la espera de una conexión.

Una vez se ha establecido la comunicación, el hilo guarda el archivo en un directorio local de la tarjeta SD. Para ello, utiliza el directorio sd_card/Session_Files/Received_Files. El nombre del archivo se ha mantenido desde la aplicación cliente, Session.xml

Tras el salvado en el dispositivo, la aplicación lee el archivo buscando la línea `Package`. Dado que este archivo está cifrado, la aplicación debe descifrarlo para poder leerlo. Para ello se ha utilizado la misma librería auxiliar de la aplicación cliente.

Una vez ha encontrado la línea buscada, abre la aplicación correspondiente, que se encargará del restaurado de sesión en su lógica interna. Cuando la aplicación cliente se abre esta aplicación servidor queda en segundo plano, sin embargo, gracias a que el código del hilo receptor se ejecuta de forma infinita, podría abrir una nueva sesión si recibiese otro archivo.

La apertura de la aplicación se realiza mediante un `Intent` de Android. Dado que pueden abrirse varias aplicaciones, los nombres de paquetes leídos en el fichero se guardan en un *array* de *Strings* y se llama a un `Intent` por cada paquete existente. Se ha restringido a 10 el máximo de aplicaciones que pueden abrirse de forma simultanea.

Una vez se ha abierto la aplicación cliente correspondiente, se cierran tanto el fichero como el *socket*, volviéndose a abrir una nueva instancia de este último e iniciando todo el proceso de nuevo de forma inmediata.

Capítulo 5
Evaluación y Resultados

Contenido

En este capítulo se comprueba el correcto funcionamiento de cada una de las aplicaciones desarrolladas a través de una serie de pruebas y se muestran los resultados obtenidos.

Las pruebas se distribuyen en diferentes apartados en función de cada uno de los módulos y funcionalidades a probar.

5.1 Entorno de Pruebas

Las pruebas de la aplicación se han realizado en dos entornos: el emulador AVD (*Android Virtual Device*) y varios dispositivos físicos.

Para las pruebas realizadas en el entorno de simulación se han utilizado distintas versiones de Android, siempre superiores a la 4.0.

Los dispositivos físicos utilizados para las pruebas han sido los siguientes:

- Teléfono HTC Desire
- Teléfono HTC Wildfire S
- Teléfono Sony Xperia E3
- *Tablet* Wolder miTab POP

Los dos dispositivos más utilizados han sido el teléfono HTC Desire y la *tablet* Wolder miTab POP. El primero contaba con la versión 4.0.4 de Android (con una ROM propia de Cyanogenmod), mientras que la *tablet* contaba con Android 4.1.1.

Las pruebas en simulador no permitían realizar la conexión entre dispositivos, por lo que se su utilización ha sido muy limitada. Por ello, no se hará distinción sobre en qué entorno se han realizado las pruebas.

5.2 Tablas de pruebas

Las diferentes pruebas realizadas quedan agrupadas, mayormente, en función del módulo del sistema al que pertenecen. Algunas de ellas, en cualquier caso, dependen de varios módulos. Se hará también distinción entre la aplicación servidor y la aplicación cliente.

Estas comprobaciones se presentan en tablas con la letra “P” seguida del identificador de la prueba, su nombre, una descripción y el resultado obtenido. Los códigos utilizados para las pruebas difieren del módulo/funcionalidad que prueben, y siempre irán acompañados del número de prueba. En cualquier caso, se explicará la notación para cada uno de los apartados. A su vez, también se indicará en cada una de las tablas si la prueba se realiza en la aplicación cliente, servidor o involucra a ambas, y a qué requisitos hacen referencia si corresponde.

5.2.1 Pruebas de interfaz gráfica

Estas pruebas determinan el correcto o incorrecto funcionamiento de la interfaz gráfica de cada una de las aplicaciones. El código utilizado es “I”.

PI-01	Escalado
Descripción	Comprobación del correcto escalado de la imagen de la interfaz dependiendo del tamaño y resolución de pantalla
Objetivo	Aplicación Cliente, Aplicación Servidor
Dependencia	RNF6
Resultado	Correcto

Tabla 8: Prueba de Interfaz Gráfica 1

PI-02	Botones
Descripción	Comprobación del correcto funcionamiento de cada uno de los botones
Objetivo	Aplicación Cliente
Dependencia	RNF6
Resultado	Correcto

Tabla 9: Prueba de Interfaz Gráfica 2

PI-03	Campos editables
Descripción	Recogen la información proporcionada por el usuario.
Objetivo	Aplicación Cliente
Dependencia	RNF6
Resultado	Correcto

Tabla 10: Prueba de Interfaz Gráfica 3

PI-04	Actualización de objetos
Descripción	Los botones de reproducción se muestran y ocultan según la situación. El título y artista de la canción son correctos.
Objetivo	Aplicación Cliente
Dependencia	RNF5
Resultado	Correcto

Tabla 11: Prueba de Interfaz Gráfica 4

PI-05	IP Mostrada
Descripción	La IP del dispositivo mostrada en la interfaz de usuario es correcta
Objetivo	Aplicación Cliente, Aplicación Servidor
Dependencia	RNF5
Resultado	Correcto

Tabla 12: Prueba de Interfaz Gráfica 5

PI-06	Ventanas emergentes
Descripción	Las ventanas emergentes se muestran en la situación que deben. Permiten al usuario pulsar un botón para su desaparición.
Objetivo	Aplicación Cliente, Aplicación Servidor
Dependencia	RNF5, RNF6
Resultado	Correcto

Tabla 13: Prueba de Interfaz Gráfica 6

5.2.2 Pruebas de funciones de reproducción

Estas pruebas determinan el correcto o incorrecto funcionamiento de las funciones de reproducción de la aplicación cliente. El código utilizado para estas pruebas es “R”.

PR-01	Lista de reproducción
Descripción	La lista de reproducción se reproduce en orden correcto, tanto al restaurar sesión como al comenzar desde cero.
Objetivo	Aplicación Cliente
Dependencia	RF8
Resultado	Correcto

Tabla 14: Prueba de Reproducción 1

PR-02	Paso de canciones
Descripción	El paso de canciones en la lista de reproducción, hacia delante o hacia detrás, se realiza de forma correcta.
Objetivo	Aplicación Cliente
Dependencia	RF8
Resultado	Correcto

Tabla 15: Prueba de Reproducción 2

PR-03	Pausa y reanudación
Descripción	El botón de Stop pausa la reproducción. Pulsar Play después hace que se reanude
Objetivo	Aplicación Cliente
Dependencia	RF8
Resultado	Correcto

Tabla 16: Prueba de Reproducción 3

5.2.3 Pruebas del Gestor de Configuración

Estas pruebas determinan el correcto funcionamiento del gestor de configuración. El código utilizado para estas pruebas es “CF”

PCF-01	Autenticación
Descripción	Al abrir la aplicación por primera vez aparece la ventana emergente de autenticación en el navegador predeterminado. El usuario es capaz de autenticarse con sus credenciales y volver a la aplicación.
Objetivo	Aplicación Cliente
Dependencia	RF1
Resultado	Correcto

Tabla 17: Prueba de Gestor de Configuración 1

PCF-02	Elección destino
Descripción	El usuario es capaz de elegir el dispositivo destino a través de su IP.
Objetivo	Aplicación Cliente
Dependencia	RNF1
Resultado	Correcto
Notas	Esta prueba implica el correcto funcionamiento de la interfaz gráfica (campos editables, ventanas emergentes, IP mostrada). A su vez, está íntimamente relacionada con el gestor de comunicación.

Tabla 18: Prueba de Gestor de Configuración 2

PCF-03	Envío de sesión
Descripción	El *switch* de envío de sesión desencadena las acciones pertinentes si está activo.
Objetivo	Aplicación Cliente
Dependencia	RNF1, RF9
Resultado	Correcto
Notas	Esta prueba implica el correcto funcionamiento de botones. A su vez, su resultado sólo es visible con el uso de funciones de otros gestores.

Tabla 19: Prueba de Gestor de Configuración 3

5.2.4 Pruebas del Gestor de Estado

Estas pruebas determinan el correcto funcionamiento del gestor de estado. El código utilizado para estas pruebas es “E”

PE-01	Recepción de eventos
Descripción	El sistema recibe eventos relacionados con la reproducción y es capaz de capturarlos correctamente, extrayendo los datos necesarios
Objetivo	Aplicación Cliente
Dependencia	RF2
Resultado	Correcto

Tabla 20: Prueba de Gestor de Estado 1

PE-02	Creación y guardado de fichero
Descripción	El fichero con los datos de la sesión tiene el formato correcto y se guarda en el directorio especificado
Objetivo	Aplicación Cliente
Dependencia	RF4
Resultado	Correcto

Tabla 21: Prueba de Gestor de Estado 2

PE-03	Procesado de fichero
Descripción	El fichero se procesa correctamente y se guardan los parámetros en él indicados para la restauración de sesión. Los datos restaurados se borran del fichero.
Objetivo	Aplicación Cliente
Dependencia	RF5
Resultado	Correcto

Tabla 22: Prueba de Gestor de Estado 3

PE-04	Restaurado de sesión
Descripción	La sesión comienza en el punto donde quedó en el dispositivo origen. El usuario no realiza ninguna acción para que esto ocurra.
Objetivo	Aplicación Cliente
Dependencia	RF7
Resultado	Correcto

Tabla 23: Prueba de Gestor de Estado 4

5.2.5 Pruebas de Gestor de Comunicación

Estas pruebas determinan el correcto o incorrecto funcionamiento del gestor de comunicación. El código utilizado es “CM”.

PCM-01	Creación del hilo cliente
Descripción	El hilo cliente se inicia única y exclusivamente cuando el usuario pulsa stop y sí se cumplen las condiciones (*switch* de sesión activo, conexión a red wifi). Una vez transmite el fichero, finaliza.
Objetivo	Aplicación Cliente
Dependencia	
Resultado	Correcto

Tabla 24: Prueba de Gestor de Comunicación 1

PCM-02	Conexión entre dispositivos
Descripción	Ambos dispositivos son capaces de comunicarse mediante *sockets* y realizar la transmisión de datos.
Objetivo	Aplicación cliente, aplicación servidor
Dependencia	RF6
Resultado	Correcto

Tabla 25: Prueba de Gestor de Comunicación 1

PCM-03	Recepción y guardado de fichero
Descripción	El fichero con los datos de la sesión se guarda en el directorio correcto con el formato original.
Objetivo	Aplicación Servidor
Dependencia	RF4
Resultado	Correcto

Tabla 26: Prueba de Gestor de Comunicación 3

PCM-04	Apertura de aplicación cliente
Descripción	La aplicación servidor es capaz de abrir la aplicación cliente de forma automática para su restaurado de sesión
Objetivo	Aplicación Servidor
Dependencia	RNF7
Resultado	Correcto

Tabla 27: Prueba de Gestor de Comunicación 4

PCM-05	Imposibilidad de Restauración
Descripción	La aplicación servidor no puede restaurar la sesión si el nivel de batería es demasiado bajo o el dispositivo no está conectado a una red wifi. El usuario recibe el aviso de imposibilidad de recepción de fichero.
Objetivo	Aplicación Servidor
Dependencia	RR1, RR3,
Resultado	Correcto

Tabla 28: Prueba de Gestor de Comunicación 5

5.3 Rendimiento

Para evaluar la aplicación correctamente se han realizado pruebas de rendimiento a nivel de tiempo. Para ello, se han medido varias veces y en distintas condiciones varios valores de tiempo:

-Tiempo desde recepción de archivo hasta apertura de aplicación cliente en aplicación servidor (implica creación de conexión, recepción de fichero, descifrado y lectura de su cabecera y lanzado de aplicación cliente. Tiempo de flujo de aplicación servidor)

-Tiempo desde que el usuario pulsa *stop* hasta que el archivo se envía en aplicación cliente (implica captación de eventos, captura de datos de sesión, guardado de datos en fichero, cifrado y envío. Tiempo de flujo de aplicación cliente).

-Tiempo desde que la aplicación servidor lanza la aplicación cliente y ésta comienza la reproducción (tiempo de restaurado).

-Tiempo desde que el usuario pulsa *stop* hasta que la sesión está completamente restaurada en el dispositivo destino (tiempo total de restauración)

Se han variado condiciones como el número de aplicaciones abiertas en el dispositivo, la calidad de la red wifi o el nivel de batería. A su vez, las pruebas se han realizado alternando dispositivos, de forma que un mismo dispositivo a veces ejercía de cliente y otras de servidor.

Cabe destacar que la única condición que parece afectar a estos tiempos, y no lo hace de manera significativa, es el número de aplicaciones abiertas. La variación de tiempos se notó especialmente cuando el teléfono móvil HTC Desire utilizado para pruebas actuaba como cliente.

Los resultados obtenidos pueden observarse en la tabla de la siguiente página (todos ellos en milisegundos).

	Flujo Aplicación Servidor	Flujo Aplicación Cliente	Tiempo de restaurado	Tiempo total de restauración
	202	167	2429	2598
	211	125	2518	2645
	214	161	2491	2656
	276	234	2488	2724
	181	112	2461	2576
	279	381	2499	2882
	301	397	2510	2909
	278	314	2495	2810
	291	123	2461	2586
	254	461	2595	3058
	202	107	2409	2516
	211	125	2518	2644
	194	121	2491	2614
	302	374	2488	2864
	181	226	2291	2519
	239	381	2613	2998
	301	417	2810	3230
	198	294	2391	2687
	191	213	2461	2680
Máx	**302**	**461**	**2810**	**3230**
Min	**181**	**107**	**2291**	**2516**
Media	**233,2001723**	**225,7476734**	**2491,609744**	**2744,918723**

Tabla 29: Tiempos de respuesta

Como puede observarse, el usuario podría pasar su sesión de un dispositivo a otro en aproximadamente 3 segundos. Cabe destacar que los desarrolladores de Spotify recomiendan introducir un pequeño retardo entre la carga del reproductor y la reproducción musical cuando se hace con parámetros (es decir, en el caso de la restauración). Este retardo recomendado es de 2 segundos. Se realizaron pruebas con diferentes retardos y con valores menores había ciertos problemas en ocasiones. Si este retardo no existiera, el tiempo de restauración sería menor.

El cifrado del archivo hace que la aplicación sea más lenta. Para realizar este cifrado se ha utilizado una libería libre (véase [19]) que implementa un cifrado DES con clave de 128 bits. Esta clase utiliza funciones de la clase `PBEKeySpec` de Android, que a su vez se basa en la especificación PKCS #5 ([21]), que propone una función de derivación de clave basada en iteraciones. El número de estas iteraciones, por defecto en la librería establecido en 65536, es el que provoca mayores variaciones en el rendimiento de la aplicación, que se vuelve considerablemente lenta para este valor, mientras que para un valor de 1000 (el mínimo recomendado por la RFC) el rendimiento es similar al de la aplicación sin el cifrado de información. Los valores mostrados en la tabla superior se corresponden a 1000 iteraciones.

El valor máximo medido para el tiempo total de restauración coincide con la situación en la que el teléfono HTC Desire actuaba como dispositivo cliente y tenía abiertas un total de 6 aplicaciones además de esta aplicación.

El valor mínimo tuvo lugar utilizando el dispositivo Sony Xperia E3 como cliente y la Tablet wolder miTabPop como servidor, ambos sin ninguna aplicación más abierta. Cabe destacar que en este caso ambos dispositivos se encontraban con un nivel de batería del 15% en el caso del móvil y un 11% en el caso de la Tablet, valor mínimo para la restauración.

Capítulo 6
Gestión del Proyecto

Contenido

En este capítulo se expondrán las etapas llevadas a cabo para la realización de este trabajo. Cada una de las etapas se explicará de forma detallada y se medirán en duración y nivel de esfuerzo. Se adjunta un gráfico de Gantt para una mejor visualización de las fases por las que ha pasado el proyecto y cómo se han organizado. También se medirán en porcentaje del total del tiempo utilizado para la realización.

Además, se mostrará el cálculo del presupuesto del proyecto y su coste total, medido en personal necesario y material utilizado.

6.1 Etapas del proyecto

Este proyecto se ha dividido en tres etapas: Definición de objetivos y planteamiento inicial, implementación del sistema y documentación. A lo largo de este apartado se describirá cada una de ellas, comentando las tareas llevadas a cabo, problemas encontrados y resultados.

6.1.1 Definición de objetivos y planteamiento inicial

Este proyecto comenzó con la idea de ser un prototipo del *framework* propuesto en el artículo tomado como base para continuar sesiones de la aplicación Spotify.

Para ello, tras la lectura del artículo y el establecimiento de objetivos, se inició una fase de investigación del API disponible de Spotify en ese momento y si el proyecto sería posible. La primera concepción del proyecto proponía el traspaso de sesiones entre un PC y un dispositivo Android, sin embargo, las funciones de la librería para PC no lo permitían, por lo que se optó por realizar la transmisión de sesión entre dispositivos Android.

Inicialmente se propuso utilizar la aplicación propia de Spotify para captar los parámetros y realizar el restaurado de sesión sobre ella, sin embargo, tras un estudio, se llegó a la conclusión de que no era factible y se comenzaron a buscar alternativas.

Tras una fase de búsqueda de funciones que pudieran ayudar a su implementación y su hallazgo, se llegó a la conclusión de que el proyecto sí era viable si se sacrificaban ciertos aspectos de los propuestos en el artículo. Por ello, se optó por crear un reproductor que utilizara las credenciales de Spotify y se buscaron las herramientas necesarias para la implementación de la aplicación.

Cabe destacar que esta etapa pasó por varias fases de cambios debido al SDK de Spotify para Android, aún en versión beta, pues cada una de las nuevas versiones afinaba un poco más el objetivo inicial. La primera versión de la librería de Spotify para la que este proyecto fue totalmente realizable fue la 1.0.0-beta4, en la cual aun así faltaban algunas de las funciones buscadas como, por ejemplo, la de iniciar la canción en un punto determinado de forma automática (el usuario debía pulsar un botón para que esto ocurriera). En las versiones anteriores el proyecto era viable, sin embargo, las funciones disponibles hacían que la variación con respecto a lo propuesto por el artículo fuesen demasiado grandes. El proyecto comenzó a implementarse para esta versión, y a medida que llegaban las nuevas se cambiaron los aspectos que mejoraba.

La versión utilizada, 1.0.0-beta6, fue la primera en la cual la reproducción se podía iniciar de forma automática sin que el usuario hiciese nada. Esta versión dejaba fuera aspectos de autenticación del usuario, como la obtención del *token* de sesión una vez se había iniciado o la carga del mismo si el usuario aún no se había autenticado, sin embargo, con el lanzamiento de la versión 7 se introdujeron también para esta, por lo que se optó por no adaptarse a ella, pues cuando se realizó su lanzamiento el proyecto ya estaba programado y era funcional. Pese a ello, con la fase de desarrollo casi terminada, se incluyeron estas nuevas opciones de autenticación.

A nivel de arquitectura siempre se tuvo claro que se quería respetar la propuesta por el artículo, con una división en cuatro módulos que gestionaban las diferentes funcionalidades de la aplicación. También se respetaron los requisitos establecidos.

Debido a la falta de conocimientos del sistema operativo Android, en esta fase se realizó un aprendizaje básico de la programación para este sistema operativo, que se llevó a cabo en paralelo con la búsqueda de funciones de Spotify. Esta etapa de aprendizaje alargó esta fase.

Con los objetivos claros y el problema planteado se pasó a la fase de implementación del sistema.

6.1.2 Implementación del sistema

En primer lugar se desarrolló el gestor de comunicaciones, programando una pequeña aplicación capaz de comunicar dos dispositivos. Este desarrollo permitió además poner en práctica el aprendizaje de la programación para Android, lo que hizo que la programación del resto de la aplicación fuese más fluida. En la versión inicial de esta aplicación los dispositivos transmitían mensajes entre ellos, pasando luego a transmitir ficheros. El código programado se utilizó en la versión final.

Finalizado este gestor, comenzó el desarrollo de la aplicación cliente. Se optó por comenzar con las funciones de autenticación y reproducción musical junto a una interfaz gráfica muy básica, que permitieron pasar a la captura de eventos y creación del fichero con los datos de la sesión.

Tras diversas pruebas de la creación correcta de este fichero, se incluyó el gestor de comunicación anteriormente programado. En una primera fase, la aplicación cliente contenía tanto la parte cliente como la parte servidor para facilitar así la realización de pruebas. Se instaló la misma aplicación en dos dispositivos, y se dotó a la aplicación de la opción de funcionar como cliente o servidor, de forma que la misma aplicación era capaz de iniciar la sesión, enviarla y restaurarla.

Una vez se comprobó el correcto funcionamiento de la restauración de sesión, se mejoró la interfaz gráfica y se eliminaron las funciones de servidor, creando en ese momento la aplicación servidor.

Los problemas asociados a esta etapa son los típicos de programación:

- Se necesitó de la búsqueda de funciones de re-escalado de la interfaz gráfica para su adaptación a todo tipo de pantallas.

- Debido a que se busca la continuidad del protocolo con otras aplicaciones y la posibilidad de restaurar varias sesiones a la vez (como se comentará en el capítulo siguiente), era necesario que la aplicación sólo tomase los datos necesarios de la aplicación en curso si un mismo fichero contuviese datos de más de una sesión. Esto se solucionó con un algoritmo de escritura y lectura de ficheros que buscaba las etiquetas correspondientes a esta aplicación, como se ha comentado previamente en el capítulo 4, “Implementación del Sistema”.

A medida que se iba implementando el sistema, se iban realizando pruebas (ya comentadas en el capítulo anterior). La falta de dispositivos reales sobre los que probar las aplicaciones hizo que esta fase se extendiese en el tiempo un poco más de lo esperado, sin embargo, el emulador de dispositivos Android, AVD, palió en parte esta falta.

6.1.3 Documentación

La última etapa consiste en la recogida en esta memoria de todo el trabajo llevado a cabo para la realización de este proyecto.

6.1.4 Organización temporal

Para terminar, se expone una visión de la duración de las etapas del proyecto y su planificación temporal. Este proyecto se llevó a cabo entre los meses de junio de 2014 y febrero de 2015.

En primer lugar, la duración de las etapas del proyecto fue la presentada en la siguiente tabla:

Etapa	Descripción	Duración (meses)
Etapa I	Definición de objetivos y planteamiento inicial	3
Etapa II	Implementación del sistema	3
Etapa III	Documentación	2

Tabla 30: Duración de cada etapa del proyecto

La cantidad de tiempo utilizada en cada parte tiene relación con el esfuerzo necesario para llevarla a cabo. Así, pese a que la etapa inicial debería haber sido más reducida en el tiempo debido a que el artículo guiaba gran parte del trabajo, fue el planteamiento lo que hizo que se alargara más de lo esperado debido a los cambios constantes en este por las nuevas funcionalidades encontradas.

A nivel porcentual, el reparto de tareas ha sido bastante equitativo:

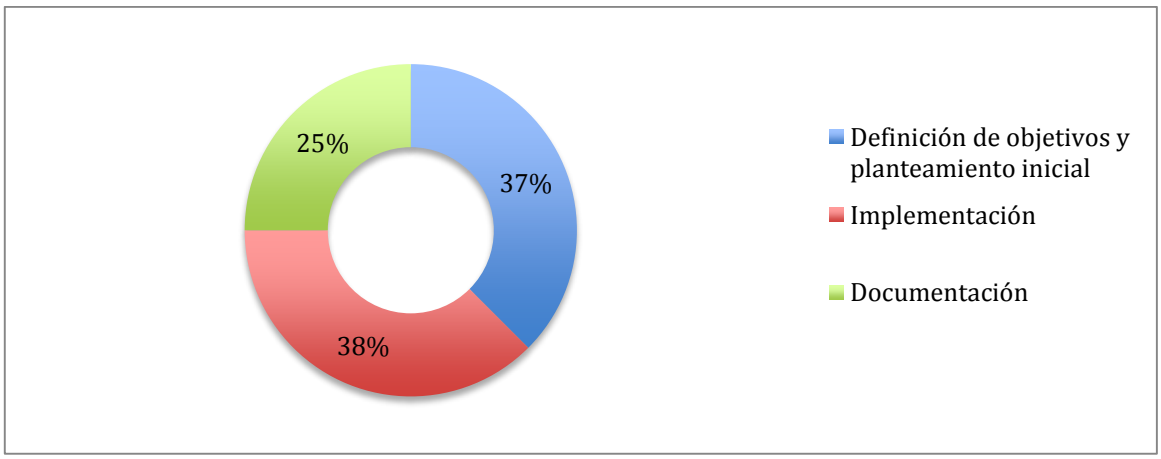

Figura 18: Porcentaje de cada etapa del proyecto

Para hacer un desglose más ajustado de cada una de las etapas, la siguiente página muestra el gráfico de Gantt con la duración de las etapas y las relaciones entre ellas.

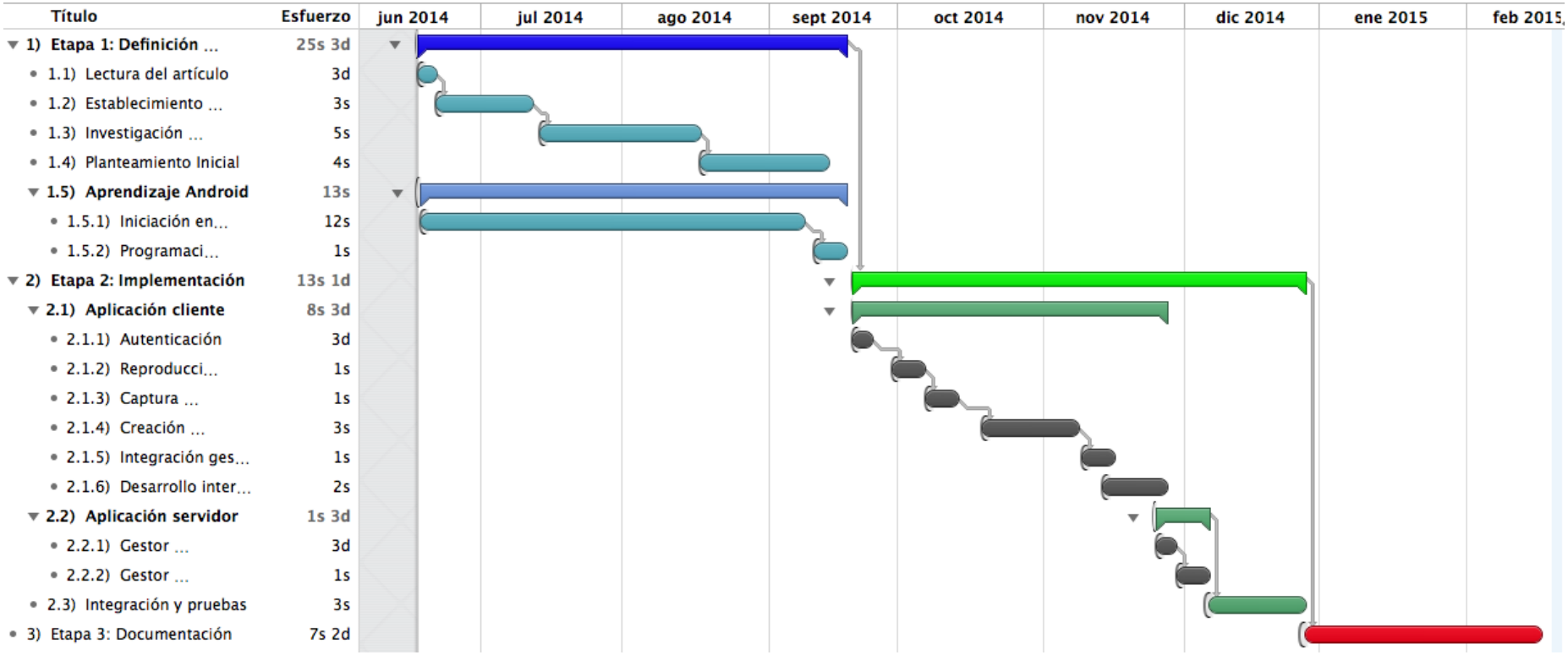

Figura 19: Diagrama de Gantt

6.2 Presupuesto

El cálculo del presupuesto se divide en dos sub-cálculos: Los costes de personal y el coste de material e infraestructura. A continuación se detallan los cálculos de cada uno de ellos.

6.2.1 Costes de personal

Para realizar el cálculo del coste de personal se ha tenido en cuenta el salario medio de un Ingeniero Técnico de Telecomunicación con menos de 5 años de experiencia en los últimos meses, establecido en 27000€ anuales. Teniendo en cuenta 1680 horas de trabajo anuales, el coste por hora de ingeniero asciende a 16,07€/hora. Dada la reciente incorporación de graduados en ingeniería al mundo laboral, no existen datos concretos para este perfil, pues las empresas siguen contratando a estos titulados como ingenieros técnicos.

A este proyecto se le han dedicado un total de 8 meses con una media de trabajo de 4 horas diarias. Esto implica 768 horas dedicadas al proyecto, por lo que basta con multiplicar para obtener el coste dedicado al personal:

Presupuesto en personal			
Concepto	**Dedicación (h)**	**Salario (€/hora)**	**Coste (€)**
Ingeniero Técnico de Telecomunicación Junior	768	16,07	12.341, 76
Total		**12.341,76 €**	

Tabla 31: Presupuesto en personal

6.2.2 Costes de material e infraestructura

Los costes de material e infraestructura implican al material *hardware* y *software* utilizados para la realización de este proyecto.

El material *hardware* empleado ha sido el siguiente:

Dispositivo	Coste
Ordenador portátil MacBook Pro	1200€
Tablet Wolder miTab POP	27€
Teléfono móvil Sony Xperia E3	140€
Teléfono móvil HTC Desire	119€
Teléfono móvil HTC Wildfire S	93,72€

Tabla 32: Coste de material utilizado

A nivel *software* no se han tenido costes por la utilización de herramientas de libre de distribución. No obstante, se tendrá en cuenta el coste de la suscripción al servicio *Premium* de Spotify, de 9,99€/mes.

En cuanto a la infraestructura necesaria, se ha requerido de una conexión fija a Internet ADSL a 6 MB de velocidad, con un coste de 48.8€/mes.

Así, los costes imputables se calculan utilizando la fórmula A/BxCxD, donde

A = nº de meses desde la fecha de facturación en el que el equipo es utilizado

B = periodo de depreciación (en meses)

C = Coste del equipo

D = Porcentaje del uso que se dedica al proyecto (habitualmente 100%)

Descripción	Coste (€)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable (€)
Ordenador portátil MacBook Pro	1.200,00	100	8	60	160,00
Teléfono móvil HTC Desire	119,00	100	2	60	3,97
Tablet Wolder miTab POP	27,00	100	2	60	0,90
Teléfono móvil Sony Xperia E3	140,00	100	2	60	4,67
Teléfono móvil HTC Wildfire S	93,72	100	2	60	3,12
Suscripción a Spotify	9,99		5		49,95
Conexión fija a Internet	48,80		8		390,40
Total					613,01

Tabla 33: Costes de material e infraestructuras

6.2.3 Coste Total

Conocidos ambos costes, podemos sumar ambos para obtener el coste total del proyecto, obteniendo un resultado final que asciende a **12954,77€**

Presupuesto Final (€)	
Costes de personal	12.341,76
Costes de material e infraestructura	613,01
Total	**12954,77€**

Tabla 34: Presupuesto final

Capítulo 7
Conclusiones y Líneas Futuras

Contenido

Este último capítulo se divide en dos partes: Conclusiones obtenidas del proyecto y líneas futuras de investigación.

Las líneas futuras de desarrollo e investigación están fuertemente condicionadas por el estado del API de Spotify para Android. A su vez, se tratará de extrapolar el resultado obtenido a diferentes aplicaciones multimedia que utilicen esta infraestructura, sirviendo este proyecto como prototipo y ejemplo.

7.1 Conclusiones

Los dispositivos móviles inteligentes o *smartphones* han conseguido convertirse en poco tiempo en elementos indispensables en el día a día. No sólo se utilizan con los fines de comunicación típicos de los teléfonos móviles clásicos, al contrario, su utilización para el ocio aumenta cada día más.

Android es el sistema operativo móvil más utilizado en la actualidad en teléfonos móviles y el segundo en *tablets*. La tienda de aplicaciones propia del sistema operativo crece por días, y los usuarios cada vez son más exigentes. A su vez, el servicio de música en *streaming* Spotify ha logrado alcanzar cifras inesperadas para sus creadores en un comienzo, creciendo tanto en su número de usuarios como en su catálogo.

Este proyecto aprovecha este catálogo y la posibilidad de tomar datos de Spotify para mejorar la experiencia de usuario en cuanto a la reproducción musical en distintos dispositivos, primando sobretodo su comodidad.

Por esta comodidad se tienen en cuenta en todo momento las preferencias del usuario en cuanto al uso de la aplicación, pudiendo decidir él en todo momento si desea continuar su sesión en otro dispositivo o no y eligiendo dicho dispositivo.

Además, este proyecto sirve como prototipo del *framework* propuesto por [5], solución para la movilidad de sesión de diferentes aplicaciones multimedia. Continuando con la investigación y probando el desarrollo para diferentes aplicaciones (sus autores ya lo probaron en un navegador web) podría universalizarse y permitir la continuidad de sesión de multitud de aplicaciones.

Pese a las diferencias entre el resultado final y la concepción inicial, basada en la aplicación nativa de Spotify, puede considerarse que la implementación utilizada muestra que la continuidad de sesión del servicio de música es posible, tan sólo hay que esperar a las mejoras en el desarrollo del API de Spotify para Android para poder acercarse cada vez más a la propuesta inicial.

El sistema desarrollado ha tenido en cuenta las posibilidades que presentan los dispositivos móviles a día de hoy, y las dos aplicaciones creadas son ligeras y utilizan la tarjeta SD del dispositivo para no ocupar memoria interna. Las pruebas realizadas en diferentes dispositivos, con diferentes características tanto hardware como software, muestran que apenas se necesitan recursos.

Por último, todas las herramientas utilizadas para el desarrollo del proyecto han sido de software libre, lo que hace que su modificación e introducción de mejoras se pueda realizar de forma sencilla y asequible.

7.2 Líneas de Investigación y Desarrollo Futuras

Para finalizar, se exponen algunas de las posibles vías futuras para la ampliación de este proyecto, de forma que aumenten sus funcionalidades.

- **Descubridor de Servicios**

La mejora más interesante pasa por evitar que el usuario introduzca la IP del dispositivo destino al que se quiere enviar la sesión. Así, se propone la

programación de un descubridor de servicios que sea capaz de reconocer dispositivos cercanos a los que pueda enviar la sesión y pregunte al usuario si desea enviarla a dicho dispositivo.

En el caso de haber varios dispositivos capaces de realizar el restaurado de sesión, podrían ofrecérsele todos ellos al usuario o simplemente elegir el más adecuado en función de la aplicación a restaurar y el contexto. Por ejemplo, si la sesión se lleva a cabo en un navegador web se propondría el PC en lugar de una *tablet*, mientras que si es reproducción de vídeo podría proponerse la televisión.

- **Mejoras de seguridad**

Dado que la aplicación envía datos de la sesión del usuario, aumentar la seguridad siempre es una buena práctica.

Se propone utilizar diferentes tipos de cifrado y hacer un estudio de cuál de qué algoritmos son más apropiados desde el punto de vista del balance entre seguridad y coste computacional.

Estas mejoras de seguridad podrían variar en función de la aplicación cuya sesión se restaurará, pues la información puede ser más o menos sensible. Quizás sólo se requiera cifrar partes de estos datos y no todo el conjunto, o el cifrado podría variar dependiendo del contexto (por ejemplo, si se está utilizando una red WiFi pública la confidencialidad necesaria sería mayor).

Por otra parte, deberíamos estudiar la autenticación continua del usuario para acceder a los distintos dispositivos y mantener la sesión activa. Por un lado, si cambiamos de dispositivo, el acceso al dispositivo destino puede estar protegido por algún tipo de mecanismo de autenticación. SuSSo debería proporcionar los medios para autenticar automáticamente al usuario en el dispositivo destino. Por otro lado, el dispositivo origen mantiene un estado de autenticación con el servicio de la aplicación externa que se está utilizando y dicho *token* debe ser reutilizado o re-obtenido por el dispositivo destino para el restablecimiento de la sesión de forma transparente. Una línea futura de seguridad pasaría por determinar los mecanismos para conseguir esta funcionalidad.

Por último, aunque la confidencialidad y la autenticación son dos pilares fundamentales para la seguridad de SuSSo, convendría hacer análisis adicionales específicos para cada aplicación que se utilice sobre la infraestructura, ya que los perfiles de riesgo de las mismas pueden ser muy diferentes (por ejemplo, uso personal para contenidos de entretenimiento en contra de uso por parte de un trabajador de la salud para restaurar datos de una aplicación).

- **Restauración de varias sesiones a la vez**

Pese a que la aplicación servidor se ha programado con miras a este tipo de utilización y el código de la aplicación cliente respeta el resto de posibles sesiones que puedan existir en el archivo de sesión, aún no se ha probado la funcionalidad de restaurar varias sesiones a la vez.

Habría que distinguir entre el tipo de sesión y tomar la decisión de cuál sería la principal a restaurar en un dispositivo, evitando así que se restauren simultáneamente dos sesiones incompatibles (por ejemplo, dos sesiones musicales en un mismo dispositivo, o una sesión musical y una de vídeo).

Resultaría interesante también estudiar el caso de si hay más de un dispositivo disponible y sesiones incompatibles. En esta situación, podrían dividirse las sesione entre los distintos dispositivos disponibles.

- **Uso de la aplicación nativa de Spotify**

En el caso de que el API de Spotify se siga desarrollando y permita el restaurado de sesión, lo ideal sería utilizar su aplicación nativa. La aplicación servidor podría re-utilizarse en este caso, pues es totalmente genérica, mientras que la aplicación cliente podría quedar en desuso (aunque, dado que SuSSo está diseñado para que no haya dependencia de la aplicación cuya sesión se mueve, no habría problemas en su utilización).

Dado que no se necesitarían funciones de reproducción, ambas aplicaciones podrían fusionarse en una sola que funcione como un *daemon* en el dispositivo y capte los datos de la sesión de forma transparente para el usuario, así como la restauración de la sesión. Esta nueva aplicación funcionaría a la vez de servidor y cliente.

Si el uso de la aplicación nativa no es posible, sería interesante observar la evolución del API comentado, pues podría implementar funciones de búsqueda dentro del catálogo y/o permitir el acceso a las *playlist* propias del usuario, lo cual permitiría crear una pseudoversión de la aplicación nativa.

- **Restauración en PC u otros sistemas operativos**

Actualmente existe una librería de Spotify para crear aplicaciones para PC. Podría investigarse esta librería y realizarse este mismo proyecto de forma que sea funcional tanto en PC como en dispositivos Android. Así, podría transferirse la sesión desde un dispositivo móvil a un PC o viceversa, a la vez que la transferencia entre dispositivos móviles seguiría siendo posible.

También existe una librería para iOS, de forma que también podría realizarse una versión para el sistema operativo de Apple.

Cuantas más versiones se creen para diferentes sistemas operativos, más universal será el protocolo siempre y cuando se garantice la compatibilidad entre todas.

- **Mejoras de los módulos de configuración y contexto**

La idea de esta mejora es automatizar y tomar decisiones inteligentes, dotando al módulo de configuración de un grado más alto de automatización. Podría dotarse este módulo de un componente de predicción que aprende del comportamiento del usuario y se anticipa a sus decisiones en la transferencia de sesiones.

Así, el dispositivo sería capaz de adaptarse al entorno en el que se encuentra de forma dinámica, moviendo la sesión hacia el dispositivo que el usuario haya elegido con anterioridad bajo las mismas condiciones de contexto.

Esta mejora es especialmente interesante para aplicaciones multimedia, donde debería tenerse en cuenta los recursos disponibles del dispositivo.

Como puede intuirse, es una mejora muy relacionada con la de descubrimiento de servicios.

Anexos

Anexo A: Glosario

A

ADSL Asymmetric Digital Suscriber Line

API Application Programming Interface

AVD Android Virtual Device

D

DES Data Encryption Standar

G

GPS Global Positioning System

I

IP Internet Protocol

J

JDK Java Development Kit

JSON JavaScript Object Notation

L

LOPD Ley Orgánica de Protección de Datos

M

MD-SSO Multi Device-Single Sign-On

N

NDK Native Development Kit

O

OAuth Open Authorization Protocol

OHA Open Handset Alliance

P

PC Personal Computer

PKCS	Public-Key Cryptography Standar

R

RFID	Radio Frequency Identification

S

SD	Secure Digital
SDK	Software Development Kit
SIP	Session Initiation Protocol
SuSSo	Seamless and Ubiquitous Single Sign-on for Cloud Service Continuity Across Devices

T

TCP	Transmission Control Protocol

U

UDP	User Datagram Protocol
URI	Uniform Resource Identifier

X

XML	eXtensible Markup Language

Bibliografía

[1] Página web oficial de Android, www.android.com, Jun. 2014

[2] Página web oficial de Spotify, www.spotify.com, Jun. 2014

[3] Página web oficial de Spotify Developer, https://developer.spotify.com, Jun. 2014

[4] Deloitte, "*Consumo Móvil en España 2014: Revolución y Evolución*", http://www2.deloitte.com/content/dam/Deloitte/es/Documents/tecnologia-media-telecomunicaciones/Deloitte_ES_TMT_Consumo-movil-espana-2014-def.pdf, Dic. 2014

[5] Arias Cabarcos, P., Almenares, F., Sánchez-Guerrero, R., Marín López, A., & Díaz-Sánchez, D. (2012). *SuSSo: seamless and ubiquitous single sign-on for cloud service continuity across devices*.

[6] IDC, "Smartphone OS Market Share, Q3 2014", http://www.idc.com/prodserv/smartphone-os-market-share.jsp, Dic. 2014

[7] TabTimes, "The State of the Tablet Market", http://tabtimes.com/resources/the-state-of-the-tablet-market, Dic. 2014

[8] Poblador i Garcia, David & Fredriksson, Emil. "*Data Center & BackEnd Buildout*",http://es.slideshare.net/davidpoblador/spotify-bcn2013slideshare, Dic. 2014

[9] Gustavsson, Niklas, "*Pressing Play*",

http://es.slideshare.net/protocol7/spotify-architecture-pressing-play,Dic. 2014

[10] Ji-Young Kwak, "Ubiquitous Services System Based on SIP," IEEE Transactions on Consumer Electronics, vol.53, no.3, pp.938-944

[11] Ming-Deng Hsieh, Tsan-Pin Wang, Ching-Sung Tsai and Chien-Chao Tseng, "Stateful session handoff for mobile WWW," Information

Sciences, Volume 176, Issue 9, pp. 1241-1265

[12] Sousa, J. P., & Garlan, D. (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Software Architecture* (pp. 29-43). Springer US.

[13] Roman, M., Ho, H., & Campbell, R. H. (2002, December). Application mobility in active spaces. In *1st international conference on mobile and ubiquitous multimedia, Oulu, Finland*.

[14] Lee, Y. J. (2014). Adaptive delivery and handoff management framework for multimedia session mobility. *Information Sciences*, *274*, 143-155.

[15] Østhus, E. C., Osland, P. O., & Kristiansen, L. (2005, January). ENME: an enriched media application utilizing context for session mobility; technical and human issues. In *Embedded and Ubiquitous Computing–EUC 2005 Workshops* (pp. 316-325). Springer Berlin Heidelberg.

[16] Spotify Connect, https://www.spotify.com/es/connect/ , Dic.2014

[17] Chen, M. X., & Lin, B. Y. (2014). Location service and session mobility for streaming media applications in home networks. *Computer Standards & Interfaces*, *36*(2), 368-379.

[18] Yu, P., Ma, X., Cao, J., & Lu, J. (2013). Application mobility in pervasive computing: A survey. *Pervasive and Mobile Computing*, *9*(1), 2-17.

[19] Encryption Library, https://github.com/simbiose/Encryption, Ene. 2015

[20] Aaron Sher, Vanteon Corporation, “Automatically Scaling Android Apps for Múltiple Screens”, http://www.vanteon.com/papersandpublications.html, Nov. 2014

[21] RFC PKCS #5, http://www.rfc-base.org/rfc-2898.html, Ene. 2015