Aplicación end-user para gestionar la adaptación de las interacciones en dispositivos móviles Android.

Máster Universitario en Ingeniería del Software, Métodos Formales y Sistemas de Información

Javier Motos González

Miriam Gil Pascual
Vicente Pelechano Ferragud

Septiembre de 2013

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Agradecimientos

Este trabajo fin de master ha sido un trabajo muy costoso, a la vez que enriquecedor tanto para mi vida académica, como para mi vida profesional. El desarrollo del proyecto no hubiera sido posible sin la ayuda de muchas personas que directa o indirectamente me han ayudado a poder superar una etapa más de mi vida.

En primer lugar y como no quería agradecer a la persona más importante de mi vida, mi madre. Ella siempre ha luchado por mí, consiguiendo que hoy en día sea una persona completa en muchos aspectos de la vida. Gracias por toda tu insistencia y por la confianza que siempre has mostrado.

Un agradecimiento muy especial a Clara Benedicto, que es la que indirectamente más perjudicada acaba con todos los proyectos que empiezo.

Muchas gracias por enseñarme a querer, a crecer, a pensar y sobre todo a confiar más en mí.

Para continuar quiero darle las gracias a un gran amigo, Francisco Moreno, siempre atento a cualquier duda que pueda surgirme, ayudándome a encontrar una solución correcta y actual a los problemas que van apareciendo. No quería darte las gracias solo por la ayuda en la elección de tecnologías para este trabajo, o en cómo solucionar algunas dudas, si no mucho más, por hacer que mi interés por ser un mejor profesional incremente cada día y sobre todo y por lo más especial, por enseñarme a leer y entender el código escrito por los demás.

A continuación me gustaría agradecer a Javier Rodríguez todo lo que me ha enseñado en estos meses de trabajo común. Hemos pasado muchas horas juntos, escuchando nuestros progresos, dándonos ideas mutuamente para conseguir que esta tesina haya salido adelante.

Como no, darle las gracias por la oportunidad ofrecida a mis dos tutores de proyecto, Miriam Gil y Vicente Pelechano además de a Nacho Mansanet por la gran ayuda ofrecida a la hora de resolver problemas técnicos.

He conocido mucha gente a lo largo de mi vida, tanto en Valencia, como en Requena, en la Universidad o en mi estancia en Polonia. No puedo nombrar a todos pero este trabajo también ha sido posible gracias a todos vosotros. Cada

vez que estamos hacéis que mi mente se centre en otras cosas y luego retome el proyecto más fresco y centrado en él.

Este agradecimiento es un poco especial. Aunque la mayoría de las personas no pueden entenderlo quiero darle las gracias a todo el mundo ciclista. Encima de una bicicleta se pasan muchas, muchas horas llevando al cuerpo a niveles que nadie puede pensar que se podrían alcanzar.

He tenido la suerte de poder aprender encima de una bicicleta una cosa muy importante en esta vida, y es aprender a competir. No solo se aprende a competir, se aprende mucho más que ello, se aprende a ayudar a tus compañeros cuando son más fuertes que tú, se aprende a ayudar a gente que no conoces que ha tenido una avería o una caída.

Acompañado de una bicicleta siempre hay una gran persona capaz de ayudarte en cualquier momento sin pedirte nada a cambio y gracias a estos me hacen pensar que aún queda gente buena y sobre aprender que aunque creas que no puedas más, “Querer es Poder”.

Para acabar me gustaría agradecer el apoyo recibido por parte de toda mi familia, mis compañeros de trabajo, toda la gente que ha escrito artículos que me han ayudado a aprender, en definitiva todas aquellas personas que habéis participado en el proyecto bien directa o indirectamente

¡Gracias!

Javier Motos González

Tabla de contenido

Índice de Tablas

Tabla de Figuras

Índice de Código

1 Introducción

1.1 Descripción

En la actualidad hay un hecho innegable, y es que la tecnología avanza a pasos agigantados.

La gran mayoría de personas utilizamos teléfonos “inteligentes” a diario, pero ¿Son realmente inteligentes los teléfonos o es simplemente un elemento de marketing para que las grandes marcas alcen sus ventas?

Si buscamos en el sentido más profundo de la palabra podemos afirmar que no son inteligentes ya que no tienen capacidades aprender, si un humano no le proporciona información el dispositivo se comportará como esté configurado en ese momento, no tiene capacidad de auto-configurarse y mucho menos de aprender.

Por otro lado si miramos el término con una visión menos purista podemos entender que un “smartphone”, que es como se llama a los teléfonos de este tipo, es simplemente un teléfono que cumple una serie de prestaciones técnicas como son, disponer de pantalla táctil, capacidad para gestionar el correo electrónico, instalar programas adicionales. También se puede ver otras características como que sean multitarea, conectividad a internet, funciones multimedia (cámara de fotos, videos), acelerómetros, GPS y capaces de leer diferentes formatos de información como PDF u Office.

Es cierto que la tendencia clara de los aparatos electrónicos es a ser cada día más inteligentes, esto se puede observar haciendo un recorrido temporal desde el año 1990 que fue cuando empezó la venta de teléfonos inteligentes de una forma más considerable. A partir de 2007 se expresa que los teléfonos adquieren la característica de ser inteligentes.

Gracias a la ayuda de varios condicionantes se podría crear inteligencia real en los teléfonos móviles como por ejemplo:

- La ubicación.
- La hora del día.
- El ruido Ambiente.

- El estado físico de la persona.
- La actividad que se esté realizando en ese momento.

Con estos condicionantes podríamos lograr que nuestros teléfonos sufrieran una aproximación muy cercana a la verdadera inteligencia.

1.2 Motivación

¿Cuántas veces el móvil ha sonado cuando estábamos reunidos y nos ha puesto en un aprieto? O por el contrario, ¿Cuántas veces no nos hemos enterado de una llamada o una notificación importante porque se nos había olvidado activar el sonido?

Hay ocasiones en las que el móvil no se comporta de una forma adecuada al contexto en el que estamos y esto puede llegar a ser un problema, por ejemplo estamos viendo una película y recibimos dos notificaciones, una visual y una sonora. En el caso de la primera no será apercibida ya que si no se está prestando una atención especial al teléfono no se detectará. En caso de la segunda será molesta, intrusiva y no únicamente con el usuario del teléfono, si no con todas las personas que haya en ese momento. En este segundo ejemplo caso sería mucho más correcto enviar una notificación mediante una vibración o una iluminación del led. Es cierto que las notificaciones son configurables, pero ¿qué pasa si acaba la película y nos dirigimos a la calle?, y ¿si después de la calle nos metemos en un lugar con mucho ruido? Es muy molesto necesitar cambiar cada poco tiempo el volumen de sonido, el nivel de vibración, las notificaciones emergentes…

Hay grandes compañías que ofrecen sistemas como calendarios virtuales, estos calendarios son los encargados de recordarnos que tenemos que hacer o donde vamos a estar. Con estos calendarios podríamos confeccionar una pequeña aproximación de cuál va a ser nuestra posición a una determinada hora y desarrollar una primera versión de cómo debería comportarse el móvil. Con esta aproximación se dejan muchos factores al azar, no siempre se va a hacer lo que marca el calendario, por ello se van a utilizar factores más precisos como pueden ser, sensores RFID, sensores de presencia, GPS, etc.

Haciendo uso de dispositivos con sistema operativo Android, el contexto, y diversos factores explicados anteriormente se ha realizado una aplicación capaz de gestionar las auto-adaptaciones de las configuraciones previas para cada servicio permitiendo al usuario final el cometido de restaurar el sistema a estados de adaptaciones anteriores.

Hemos de tener en cuenta que para ello se necesitan dispositivos capaces de adaptarse a la situación sin necesidad de intervención humana, es decir, necesitamos sistemas auto-adaptables (De Lemos, 2011).

1.3 Planteamiento del problema

El proyecto presentado es un potente sistema de gestión para las adaptaciones de la configuración para cada servicio en dispositivos Android. A pesar de la envergadura del proyecto no requiere de un gran tiempo de aprendizaje puesto que una de las premisas más importantes es la amigabilidad del sistema.

Hoy en día resulta bastante difícil poder gestionar los estados o adaptaciones de la configuración de nuestros dispositivos de una forma exacta y natural. Pero no solo eso, resulta casi imposible que un dispositivo consiga aprender de elecciones anteriores siendo capaz de conseguir gestionarlas de una forma automática con un porcentaje bastante bajo de error llegando incluso en largos periodos a conseguir que las auto-adaptaciones sean perfectas.

En este trabajo se presenta una aplicación móvil para que los usuarios finales sean capaces de gestionar las auto-adaptaciones de la configuración para cada servicio pudiendo hacer rol back a adaptaciones previas consiguiendo así que los servicios se comporten de una forma menos intrusiva y se adapten a las preferencias de los usuarios.

La visualización de los estados del dispositivo móvil se pueden dividir en dos grupos: el primer grupo está formado por todos los estados previos ordenados de forma descendente (historial de adaptaciones) y en el segundo grupo se muestran las adaptaciones actuales de un determinado servicio.

Gracias a esta aplicación, la administración de las auto-adaptaciones se realizará de una forma user-friendly, permitiendo que la aplicación pueda ser utilizada por un gran número de usuarios.

1.4 Contexto de la Tesina de Máster

Hay que comentar que este proyecto se deriva de la tesis doctoral presentada por Miriam Gil. (Gil Pascual, 2013) En esta tesis se introducen capacidades de auto-adaptación en los dispositivos móviles para proporcionar interacciones que no resulten molestas al usuario.

La investigación ha sido desarrollada por el Centro de Investigación en Métodos de Producción Software (PROS) (www.pros.upv.es), que es un Centro

Propio de investigación de la Universidad Politécnica de Valencia. Los objetivos de este grupo de investigación son estudiar nuevos procesos, métodos y estrategias para enfocar el proceso de producción de software desde una perspectiva rigurosa, fiable y de gran aplicabilidad además de transferir esta actividad al tejido empresarial, tanto a nivel nacional como internacional. La misión del grupo de investigación es "Mejorar los métodos de desarrollo de Software tradicionales, proporcionando métodos y técnicas dirigidas por modelos para desarrollar de forma sistemática y productiva software de calidad".

Para entender un poco mejor en que ha consistido la tesina podemos observar la Figura 1.1, en la cual vemos de forma gráfica en que parte de la tesis doctoral de Miriam Gil se ha centrado nuestra tesina:

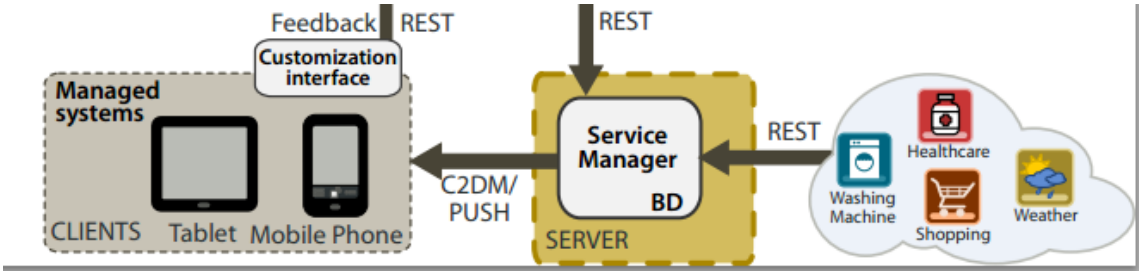


Figura 1.1 Estructura desarrollada

En concreto, la aplicación Android desarrollada se centra en la parte de la búsqueda de datos y almacenamiento de las modificaciones, acceso al servidor y gestión de las auto-adaptaciones realizadas por el usuario, es decir la parte cliente del proyecto.

Para enmarcar mejor nuestra contribución vamos a dar una breve descripción de la tesis doctoral desarrollada por Miriam Gil (Gil Pascual, 2013) de la que deriva el presente trabajo.

1.4.1 Adapting Interaction Obtrusiveness

Partiendo de las bases de la Ingeniería Dirigida por Modelos (MDE) (Schmidt, 2006) y de los principios de la Computación Ubicua, el trabajo se centra en diseñar y desarrollar servicios que sean capaces de adaptar sus interacciones de acuerdo a la atención del usuario en cada momento. El principal objetivo es introducir capacidades de adaptación considerada en los servicios ubicuos para proporcionar interacciones que no perturben al usuario. Esto se consigue mediante un proceso de desarrollo, que cubre desde el diseño de los servicios hasta su implementación, centrándose en los requisitos de la interacción particulares para cada usuario.

Además, como las necesidades y preferencias del de los usuarios pueden cambiar con el tiempo, la aproximación utiliza la estrategia del aprendizaje por refuerzo para ajustar los modelos de diseño iniciales de forma que maximicemos la experiencia del usuario. El diseño inicial de la interacción basado en el nivel de molestia nos asegura un comportamiento inicial consistente con las necesidades de los usuarios en ese momento. Luego este diseño se va refinando de acuerdo al comportamiento y preferencia de cada usuario por medio de su retroalimentación a través de la experiencia de uso.

Además, también proporcionamos una interfaz móvil que permite a los usuarios finales personalizarse de forma manual los modelos en base a sus propias preferencias. La estructura completa del proyecto desarrollado podemos verla en la **¡Error! No se encuentra el origen de la referencia..**

Figura 1.2 Arquitectura para la adaptación del nivel de molestia de las interacciones

1.5 Objetivos Globales

La tesina desarrollada es una plataforma software compuesta por varios elementos y con la cual se puede hacer una gestión total del comportamiento de las adaptaciones de los dispositivos orientada a todos los usuarios Android. La aplicación debe ser consciente del contexto (Nadav Savio, 2007) , y las

interfaces deberían ser capaces de adaptarse al usuario (Calvary G, 2003), consiguiendo de esta forma que la aplicación sea lo menos molesta para el usuario (M. Gil, 2012), permitiendo además la gestión de las adaptaciones y restablecer un estado anterior si lo considera oportuno.

El sistema debe cumplir los siguientes objetivos:

- **Portabilidad**: Es vital que el sistema funcione en cualquier dispositivo electrónico que pueda sernos útil en nuestro día a día. No podemos restringirnos solo a que sea utilizado en teléfonos móviles, también es importante que se pueda en tablets, o incluso en las futuras Google glass.

- **Utilizar formatos estándares y abiertos**: Se usarán estos para facilitar así el tratamiento y futuro uso de la información por aplicaciones independientes. Se usarán tecnologías libres y ampliamente extendidas como Silex (PHP), MySQL, Java, etc., con lo que conseguiremos la interoperabilidad de aplicaciones entre usuarios y la facilidad para los futuros desarrolladores que trabajen con el sistema.

- **Escalabilidad**: Este proyecto puede ser usado por millones de personas en el mundo y por lo tanto debe ser totalmente escalable para todas las opciones que se puedan imaginar.

- **Robustez, eficiencia, estabilidad:** El margen de error debe ser pequeño, un fallo en los avisos puede suponer un problema. Debe ser eficiente para que los cambios de alertas sean rápidos.

- **Gestionar alertas**: No es suficiente que el teléfono informe de forma adecuada según el contexto, también tiene que ser capaz de gestionar todo el sistema de alertas y de una forma correcta. El sistema debe diferenciar entre los distintos niveles de alertas e interactuar con el usuario y el contexto.

- **Facilidad y Simplicidad**: El sistema debe ser lo más intuitivo posible, así como súper simple de poder volver a configuraciones anteriores, cambiar algunas preferencias de las alertas o seleccionar nosotros todos los niveles de alerta.

- **Almacén de Datos**: Los datos son almacenados y consultados en un servidor donde se guardan configuraciones anteriores, y como esta en cada momento los avisos.

1.6 Objetivo de la Tesis

El principal objetivo es desarrollar un software capaz de permitir a los usuarios ver las adaptaciones anteriores por las que el dispositivo ha pasado, siendo capaz éste de restituir la configuración que más convenga en cada momento al usuario final, según las circunstancias medioambientales, de localización u otros parámetros.

En esta tesina se han desarrollado dos partes claramente diferenciadas de las cuales vamos a explicar brevemente cada una de ellas:

- **Herramienta de visualización de adaptaciones end-user**: debido a diferentes eventos enviados a los dispositivos móviles las preferencias de usuario pueden cambiar. Por tanto las adaptaciones diseñadas podrían no ser del total agrado del usuario y por ello se deben poder deshacer las adaptaciones a estados anteriores, adaptando de esta manera el dispositivo a las nuevas preferencias del usuario.

- **Acceso al servidor de almacenamiento de adaptaciones**: para poder mostrar la información precisa al usuario y restaurar adaptaciones anteriores necesitamos la información relacionada. Por ello se ha creado un servidor capaz de proporcionarnos los datos precisos en todo momento consiguiendo así la correcta gestión de las adaptaciones.

1.7 Estructura del documento

La estructura que va a seguir el documento va a ser la siguiente:

Capítulo 2 Aplicaciones Relacionadas

En este capítulo se presentan varias aplicaciones similares a la desarrollada realizando una comparativa entre estas.

Capítulo ¡Error! No se encuentra el origen de la referencia. **Contexto ecnológico**

En este punto se presentan las tecnologías utilizadas en el desarrollo de nuestra parte del proyecto haciendo una especial explicación de aquellas más importantes.

Capítulo 4 Desarrollo de la Propuesta

En este capítulo se explican los requisitos además de todo el proceso de diseño, implantación y pruebas de las tareas realizadas describiendo en profundidad todo lo realizado.

Capítulo 5 Manual de Usuario

En este capítulo se presenta la aplicación desarrollada, explicando cómo utilizarla en cada momento.

Capítulo 6 Casos de Uso

En este capítulo se expone un caso de estudio sobre el que se ha aplicado la propuesta de la tesis para demostrar su viabilidad práctica.

Capítulo 7 Conclusiones y trabajo futuro

Se plantean las conclusiones extraídas y trabajos de ampliación de la tesina desarrollada.

2 Aplicaciones Relacionadas

En este segundo capítulo de la tesina se desarrolla un estudio de las aplicaciones punteras en este ámbito, viendo las similitudes y las diferencias entre estas y la aplicación que nosotros hemos implementado.

Para comenzar vamos a ver el Time Machine de Apple.

2.1 Apple Time Machine

Time Machine es un software de backup desarrollada por Apple Inc., para hacer copias de seguridad. Viene incluido con el sistema operativo Mac OS X y fue introducido con el lanzamiento de la versión 10.5.

Como muchas utilidades de backup, Time Machine crea copias de seguridad incrementales de archivos que pueden ser restaurados en una fecha posterior. Permite al usuario restaurar todo el sistema, múltiples archivos o si el usuario lo desea un único archivo. Trabaja al interior de iWork, iLife y otros programas compatibles, haciendo posible la restauración de objetos individuales sin salir de la aplicación. Time Machine captura el estado más reciente de la información de un disco. Cuando estas capturas sean más antiguas, se priorizaran progresivamente a las más recientes.

En Figura 2.1 se observan capturas guardadas de los ficheros. Fijando la vista al lado derecho de la figura se ve una barra temporal donde podemos colocarnos en el día exacto que queramos seleccionar el fichero a restaurar. Apple ha creado un efecto llamado Space Gallery, que las imágenes van desapareciendo incrementalmente según avanzamos en la línea temporal dando paso a las más antiguas hasta que lleguemos a encontrar el estado necesario para indicar que vamos a iniciar su restauración.

Figura 2.1 Interfaz Apple time machine

2.1.1 Como funciona Time Machine

Una de las cosas más sorprendentes a la hora de utilizar Time Machine es la sencillez que presenta puesto que hace uso de tecnologías tan antiguas como los hard links para funcionar.

El funcionamiento de una copia de Time Machine es el siguiente:

- La primera vez que se realiza una copia, el sistema crea en un HDD externo una copia idéntica a nuestra estructura, y completa, a la que tenemos en nuestro ordenador. Esto hace que posteriormente se pueda navegar por las carpetas de forma manual como si se accediera a un disco interno.

- Una vez realizada la primera copia completa del sistema, Time Machine se dedicará a una tarea secundaria, comprobar los cambios en los archivos. Es decir, el sistema se encargará de realizar una comprobación constante de todos los archivos que son modificados por nosotros.

- Cada vez que un archivo modifica su fecha de creación, asociada a la modificación del mismo, el sistema detecta que hay que volver a realizar

una copia de dicho archivo y almacenarla en el HDD externo. Así con cada archivo que se modifique.

- Los hard links son utilizados para dar esa “apariencia” de que se realizan copias completas cada vez que realizamos un backup del sistema, aunque en realidad gracias a esos enlaces estamos accediendo a archivos que se copiaron hace bastante tiempo en el sistema.

Una vez el Disco Duro este lleno el sistema se encargará de borrar las copas de archivos más antiguos, con previo aviso al usuario.

2.1.2 Problemas de Time Machine

La única prueba que realiza Time Machine a la hora de realizar una copia de seguridad de un archivo es que este se haya modificado recientemente, el problema de utilizar este método se produce con grandes archivos o bases de datos.

Cualquier pequeña modificación en este tipo de archivos repercute en una modificación de su fecha de creación y por lo tanto unos cuantos kb modificados se pueden convertir en una copia de muchos GB extras a la hora de guardar una copia de dichos archivos.

Apple es plenamente consciente del fallo y por eso algunos programas se modificaron para no verse afectados. Es el caso de programas como Mail, iPhoto e iTunes. Programas que hacen uso de librerías sobre las cuales Time Machine puede actuar y detectar cambios internos.

Cuando se realiza una copia en iPhoto por ejemplo, no se copia toda la biblioteca, si no la imagen que se ha añadido o modificado.

2.2 Llama- Location Profiles

Llama es una aplicación Android que se encarga de cambiar automáticamente perfiles de usuario según el posicionamiento geográfico.

2.2.1 ¿Cómo funciona Llama Location Profiles?

El funcionamiento de Llama es bastante sencillo, y como se ha explicado anteriormente se basa en activar o desactivar perfiles según la ubicación del usuario final, aunque también es capaz de utilizar otras condiciones como patrones.

Para comenzar diremos que tenemos 4 pestañas en la pantalla principal, áreas, eventos, perfiles y configuraciones recientes.

Para empezar, vamos a ver los Perfiles (Figura 2.2). En estos tenemos varios predefinidos, y en ellos podemos modificar todo lo relacionado con el volumen, todos de llamada, vibración, sonido de notificaciones, alarmas, etc. Estos perfiles no son únicos, podemos crear tantos perfiles como necesitemos.

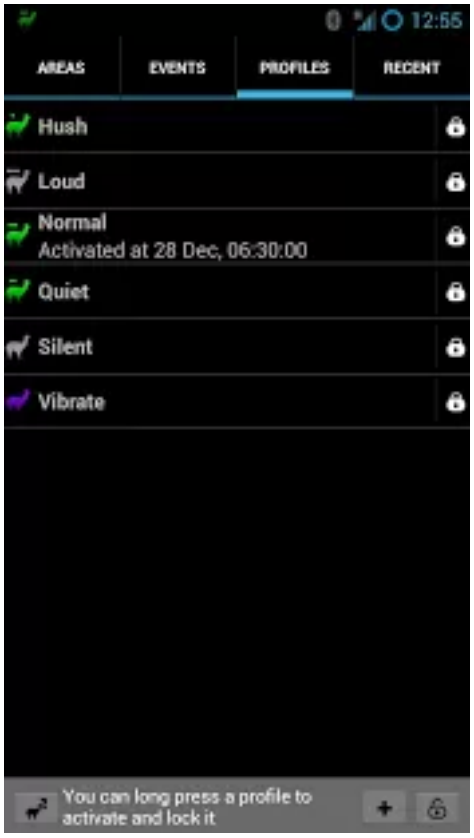


Figura 2.2 Perfiles Llama

En la pestaña Área (Figura 2.3Figura 2.3 Áreas Llama) se determina y nombran las zonas que se quieran asignar, pero se hace de una forma un poco rara. Para poder añadir una zona el usuario se debe encontrarnos en ella y decirle que esa zona se va a pasar a llamar X, simulando así un sistema de aprendizaje. Además se debe indicar cuanto tiempo vamos a encontrarnos en esa posición para de esa manera asignarle un radio determinado.

Figura 2.3 Áreas Llama

En la pestaña de Eventos (Figura 2.4), es el lugar donde más tiempo se va a dedicar a la configuración de la aplicación. En esta pestaña se indican las condiciones para que se active o se desactive un determinado perfil. Hay que remarcar que hay bastantes eventos como pueden ser, el porcentaje de la batería, si el teléfono está conectado al cargador, si están conectados los auriculares, etc. Se pueden relacionar los eventos con las áreas consiguiendo de esta forma evento con una mayor precisión.

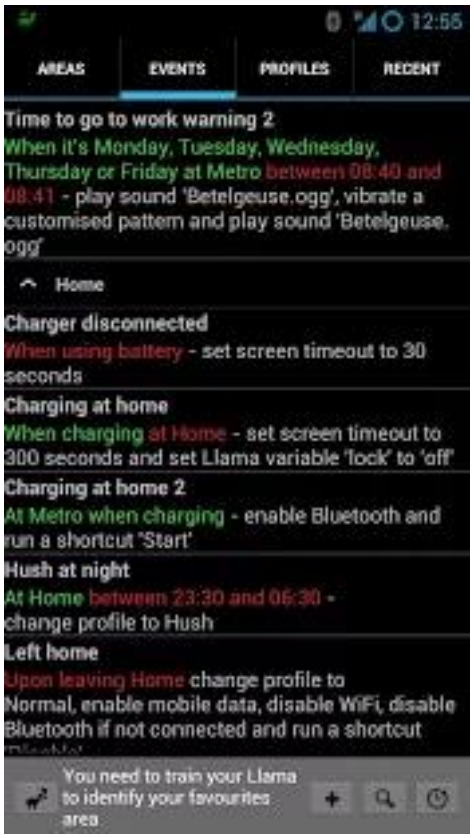


Figura 2.4 Eventos Llama

En la pestaña Recent se encuentra la configuración que se está utilizando en este momento, indicando las preferencias. Esto se puede comparar a nuestra pestaña de estado actual.

2.2.2 Problemas de Llama Location Profiles

Como principales problemas se indica que para indicar áreas se debe estar en ellas, esto es un poco diferente a las demás aplicaciones ya que las áreas se suelen definir sobre un mapa. También hay se indica que el proyecto está en un nivel de madurez bajo y esto puede dar muchos errores, siendo esto un poco molesto.

También añadir que no dispone de sistema de aprendizaje y toda la configuración ha de ser introducida por el usuario.

Como último problema no dispone de un historial con los perfiles utilizados anteriormente para poder seleccionar uno de estos sí el usuario considera que se adapta mejor a la situación actual.

2.3 Setting Profiles

Esta aplicación permite poder seleccionar el comportamiento que va a tener un dispositivo móvil según una serie de perfiles definidos anteriormente de una forma similar a los antiguos móviles de Nokia donde existían perfiles del tipo reunión, coche, muy alto, etc.

2.3.1 ¿Cómo funciona Setting Profiles?

Para poder utilizar Setting Profiles se debe ir a la pestaña de Perfiles (Figura 2.5) y crear un primer perfil según una serie de parámetros definidos por la aplicación.

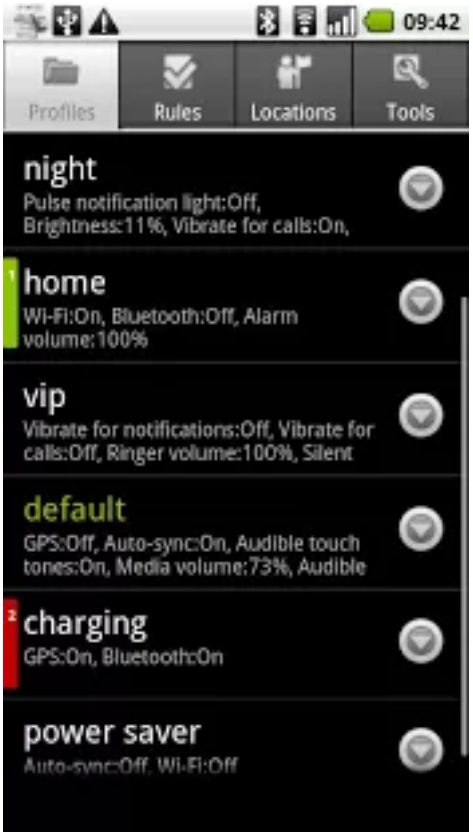


Figura 2.5 Perfiles Setting Profiles

Una vez se han creado tantos perfiles como el usuario desee se debe seleccionar el widget de esta aplicación el cual permitirá cambiar de perfil mediante una pulsación en una de estas opciones.

Figura 2.6 Selección Perfiles Setting Profiles

La versión de pago ofrece más funcionalidades como la creación de condiciones o localizaciones para que se active un perfil determinado según estas condiciones.

Para terminar con la funcionalidad de Setting Profiles hay que destacar que se permite realizar una copia de seguridad de los perfiles por si se cambia de móvil.

2.3.2 Problemas de Setting Profiles

Como toda aplicación dispone de una serie de problemas que vamos a comentar a continuación. En primer lugar no se nos ofrece ningún perfil predeterminado o de ayuda que nos permita tener un primer perfil.

También destacar que no dispone de aprendizaje, se base en localizaciones y configuraciones del usuario pudiendo esto crear demasiados fallos en las configuraciones.

Finalmente decir que dispone de una gestión de perfiles por prioridades, una idea magnifica si el funcionamiento fuera el correcto, cosa que no ocurre haciendo que una de las opciones más potentes sea un problema a solucionar.

2.4 Comparación entre aplicaciones

En este último apartado se realiza una serie de comparaciones entre las aplicaciones presentadas. En la comparación se ven las principales características ofrecidas y como se presentan.

Como primer parámetro de comparación estudiamos las plataformas en las cuales se ejecutan estas aplicaciones. En el time machine de Apple es un software diseñado para ordenadores, aunque en un futuro se podría utilizar también para dispositivos móviles. Por otro lado las otras aplicaciones son válidas para dispositivos móviles con un sistema operativo Android.

El segundo punto de comparación es la interfaz gráfica de usuario. En este punto se observa la gran similitud entre las aplicaciones móviles, mostrando una lista, bien sea de servicios o bien sea de escenarios. En cambio la aplicación de Apple ofrece una vista mucho más potente pero poco útil para un dispositivo móvil.

Únicamente la interfaz desarrollada en el proyecto es capaz de mostrar las preferencias para cada uno de los servicios y no las preferencias para un escenario completo. Esto permite lograr unas configuraciones mucho más completas y personalizables.

También hay que destacar que solamente la aplicación desarrollada es capaz de ofrecer un historial por servicios de todas las disponibles para móviles. Gracias a esta funcionalidad se crea una novedad en el mundo de las aplicaciones móviles.

Para continuar hay que fijarse en que solo nuestra aplicación y la desarrollada por Apple no requieren configuración, se va creando historiales y conocimiento mediante la utilización. En las aplicaciones vistas (Llama y Setting Profiles) se configura la aplicación desde un principio aunque la primera dispone de unas configuraciones creadas por defecto para poder utilizarlas si se adaptan a las preferencias del usuario.

Como último punto vamos a ver por qué se ha decidido utilizar un almacenamiento en la nube por contra de un almacenamiento en local.

Debido a los continuos cambios de situaciones experimentados durante el día, las configuraciones de las adaptaciones para cada servicio deben comportase de una forma correcta. Por ello, cada vez que el dispositivo realiza una auto-adaptación se debe almacenar tanto el estado actual, como el previo, para a través de esto poder realizar el historial que nos permitirá gestionar las configuraciones. Teniendo en cuenta que el trabajo ha sido desarrollado para aplicaciones Android, sistema operativo utilizado comúnmente en los

dispositivos móviles con pocas prestaciones. La memoria en estos dispositivos suele ser limitada, por lo que almacenar la información en local puede llegar a ocupar todo el espacio disponible. Completar el espacio disponible en memoria puede crear una mala experiencia de usuario puesto que para el dispositivo tendría una lentitud en la respuesta o no permitiría el almacenamiento de otras aplicaciones.

También debemos pensar que el sistema debe adaptarse a todos los dispositivos de un usuario, es decir al teléfono móvil, a la Tablet, etc. Por ello se necesita un sistema capaz de adaptarse en tiempo de ejecución mediante una serie de cálculos, y en la mayoría de los casos el dispositivo no dispone de potencia necesaria para gestionar estos cálculos.

Además de lo explicado anteriormente un grave problema para el correcto funcionamiento del software sería la perdida de información. Sí esto ocurriera se perderían todos los datos que permiten al dispositivo adaptarse de una forma correcta al contexto. La aplicación necesitaría un nuevo periodo de aprendizaje ya conseguido anteriormente. Esto además se relaciona directamente con los continuos cambios por antigüedad de los dispositivos móviles.

A continuación se va a realizar una tabla comparativa de las aplicaciones desarrolladas para dispositivos Android junto con la desarrollada en la tesina. Se ha decidido dejar fuera de la comparación a la aplicación Apple Time Machine puesto que actualmente no tiene versión para dispositivos móviles.

	Llama Location Profiles	**Setting Profiles**	**Gestión de la auto-adaptación**
S.O	>= Android 2.1	>= Android 1.6	>= Android 4.2
Preferencias por servicio	NO	NO	SI
Escenarios	SI	SI	SI
Facilidad de uso	4.7/5[1]	4.1/10[2]	Sin valoración
Necesidad de configuración inicial	SI	SI	NO
Almacenamiento de datos	Local	Local	Servidor
Roll-back	NO	NO	SI

Tabla 2.1 Tabla comparativa entre aplicaciones relacionadas

[1] Valoración extraída de la opinión de los usuarios en Google Play.
[2] Valoración extraída de la opinión de los usuarios en Google Play.

3 Contexto Tecnológico

3.1 Tecnologías Utilizadas

Durante el desarrollo se han encontrado varios problemas que hemos tenido que tomar decisiones importantes para solucionarlos. Para cada problema se han utilizado una serie de tecnologías y patrones más oportunos para la solución propuesta y que no solo deben servir ahora si no en un futuro.

Se han investigado y estudiado la evolución de las tecnologías antes de cada decisión para no caer en una tecnología en desuso o con un corto estado de vida futuro que en un corto periodo de tiempo puede ser un problema para el desarrollo temporal del proyecto.

A continuación vamos se explican las tecnologías utilizadas en cada momento para implementar una solución correcta y el por qué se han elegido.

3.1.1 REST

REST (REpresentational State Transfer) (Fielding, 2000), (Marqués, 2013) es una técnica de arquitectura software para sistemas hipermedia distribuidos. El término surgió en una tesis doctoral en el año 2000 por Roy Fielding (Fielding, 2000), uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser muy común por la comunidad de desarrollo.

Con esta tecnología se permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y usual que tecnologías usadas anteriormente como SOAP o XML-RPC.

REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a internet.

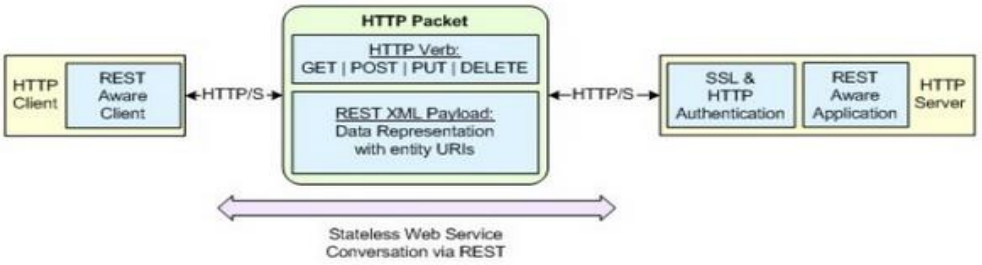


Figura 3.1 Estructura REST

Para crear la API REST se utiliza el modelo llamado Richardson Maturity Model, donde se dan una serie de patrones y buenas prácticas que hay que utilizar si se quiere crear un software de calidad (Fowler, 2010). Los niveles para ello son el uso correcto de URIs, uso correcto de HTTP e implementar Hypermedia.

3.1.1.1 Uso correcto de URIs

Cuando desarrollamos una aplicación web las URLs permiten acceder a cada una de las páginas, documentos, secciones etc. En REST a esto se le llama **recursos**. Podemos decir que el recurso es la información a la que queremos acceder.

Las URL son un tipo de URIs que además de permitir identificar de forma única el recurso, permiten localizarlo para poder acceder a él o compartir su ubicación.

Podemos ver un ejemplo de la estructura de una URL en ¡Error! No se ncuentra el origen de la referencia. :

```
{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}
```

Código 3.1 Estructura URL

Para crear las URIS de un recurso existen una serie de reglas:

- Los nombres no deben implicar una acción, por lo tanto debe evitarse el uso de verbos.
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independientes de formato.

- Deben mantener una jerarquía lógica.
- Los filtrados de información no se hacen en la URL.

3.1.1.1.1 Las URIs no deben implicar acciones y deben ser únicas

Este punto se va a explicar con un ejemplo ya que va ser una forma fácil de entender el concepto.

Imaginamos que se quiere editar el usuario con el identificador 5, se podría crear una URI de este tipo **/user/5/edit** pero sería incorrecta ya que aparece el verbo editar en la misma.

Para el usuario 5 la URI correcta seria **/user/5** y mediante el método de la llamada seriamos conscientes si es para editar, borrar o recuperar. Los métodos posibles los veremos más adelante.

3.1.1.1.2 Las URIs deben ser independientes de formato

Vamos a continuar con el ejemplo anterior, donde se quería recuperar el usuario con identificador 5, pero ahora se recupera en un formato XML, para ello se accede a la **URI /user/5.pdf** pero esto es incorrecto puesto que estamos indicando la extensión pdf en la misma.

Para recuperar el recurso se utiliza la URI **/user/5** en la petición se indica que queremos la respuesta en formato pdf.

3.1.1.1.3 Las URIs deben mantener una jerarquía lógica

Las URIs deben seguir una jerarquía lógica por ejemplo, la URI **/event/3/user/5** no sería correcta, ya que la jerarquía lógica es al revés **/user/5/event/3**.

3.1.1.1.4 Filtrados y otras operaciones

Para filtrar, ordenar, paginar o buscar información en un recurso, se debe hacer una consulta sobre la URI, utilizando parámetros HTTP en lugar de incluirlos en la misma.

Por ejemplo, la URI **/users/order/des** seria incorrecta ya que el recurso de listado de usuarios ordenados de forma descendente seria el mismo que para obtener la lista de todos los usuarios, utilizando parámetros para obtener los datos.

Un ejemplo de una URI correcta seria: **/user?order=DESC**

3.1.1.2 HTTP

HTTP (R. Fielding, 1999) es un protocolo de transferencia de hipertexto, es el protocolo utilizado en cada transacción de la WWW. Este protocolo fue desarrollado por el W3C[3] y IETF[4], aunque esta relación acabó en el año 1999 con la especificación RFC 2626 que especifica la versión 1.1 de HTTP. En este RFC se define la sintaxis y la semántica que utilizan los elementos web para comunicarse.

Es muy importante para crear un código de calidad conocer bien HTTP, para ello es importante leerse el RFC donde se explica la utilización del mismo.

Para desarrollar una API REST correcta es importante dominar los siguientes aspectos:

3.1.1.2.1 Métodos HTTP

Como se ha visto no se deben poner verbos que impliquen acciones para indicar una acción, para ello se utilizan los siguientes métodos HTTP definido cada uno de ellos para una acción:

- **GET:** Para Consultar y leer recursos.
- **POST:** Para crear recursos
- **PUT:** Para editar recursos.
- **DELETE:** Para eliminar recursos.
- **PATCH:** Para editar partes concretas de un recurso.

Vamos a ver los métodos en un ejemplo:

- **GET /user**: Nos devuelve los usuarios.
- **POST /user**: Nos permite crear un usuario nuevo.
- **GET /user/5**: Nos permite recuperar un el detalle del usuario cuyo id es el número 5.

[3] World Wide Web Consortium: consorcio que produce recomendaciones para la World Wide Web
[4] Internet Engineering Task Force: organización internacional de normalización.

- **PUT /user/5**: Nos permite editar el usuario con el id 5, sustituyendo la totalidad de la información anterior por la nueva.

- **DELETE /user/5**: Nos permite eliminar el usuario con id 5.

- **PATCH /user/5**: Nos permite editar cierta información del usuario con id 5, como por ejemplo el nombre sin necesidad de editar otro campo.

Durante los últimos años los desarrolladores web solo han utilizado los métodos que dan soporte los navegadores que son GET y POST, pero si se trabaja con REST, sería un error solo utilizar estos dos métodos ya que sería un uso incorrecto además que nos obligaría a poner verbos en las URLs y como hemos visto anteriormente no es un uso correcto.

3.1.1.2.2 Códigos de estado

Un error que se comete con mucha frecuencia cuando se construye una API es el de no utilizar las herramientas ya creadas para alguna funcionalidad, esto se puede ver con los códigos de estado, HTTP ya dispone de una serie de códigos de estado y es absurdo inventar códigos propios para saber cómo ha ido la operación.

En la Figura 3.2 se muestra un ejemplo donde se devuelve un código de estado 200 que en HTTP significa que la petición se ha realizado correctamente, sin embargo, estamos devolviendo en el cuerpo de la respuesta un error y no el recurso solicitado en la URL.

```
Petición
========
PUT /facturas/123

Respuesta
=========
Status Code 200
Content:
{
  success: false,
  code:    734,
  error:   "datos insuficientes"
}
```

Figura 3.2 Ejemplo código respuesta erróneo

Este problema es un inconveniente ya que tenemos varios aspectos inadecuados:

- No es REST ni estándar.

- El cliente que accede a la API debe conocer el funcionamiento especial y como tratar los errores de la misma, por lo que requiere un esfuerzo adicional importante para trabajar con ella.

- Tenemos que mantener los códigos y mensajes de error con todos los inconvenientes que eso supone.

Esto no es aconsejable ya que HTTP tiene un abanico muy amplio que cubre todas las posibles indicaciones que se deben añadir en las respuestas cuando las operaciones han ido bien o mal.

Es importantísimo conocer HTTP independientemente de la tecnología que utilices.

La Figura 3.3 muestra la respuesta correcta.

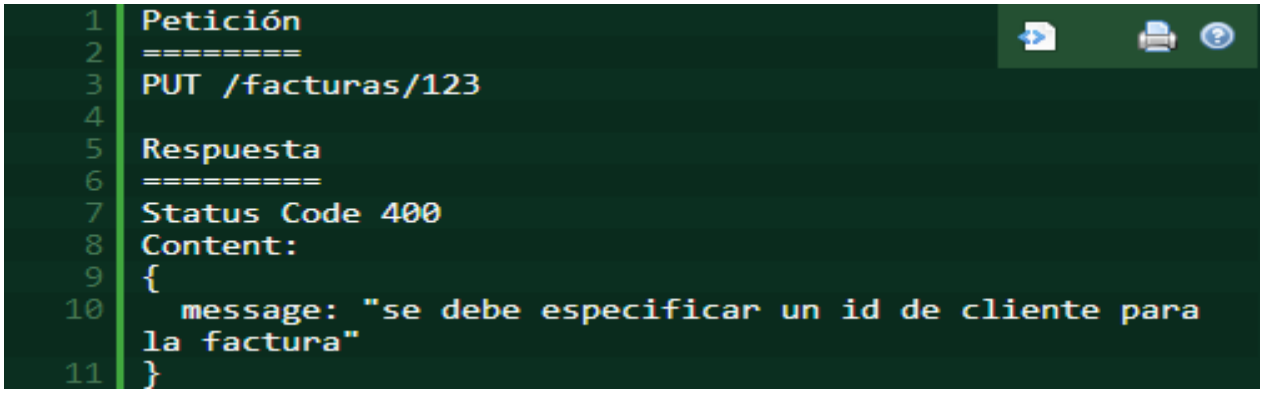


Figura 3.3 Ejemplo respuesta correcta

Los códigos de estado proporcionados por HTTP son los siguientes:

- **200**- Ok Standard response for successful HTTP request.
- **201**- Created
- **202**- Accepted
- **301**- Moved Permanently
- **400**- Bad Request
- **401** Unauthorised
- **402** Payment Required
- **403** Forbidden
- **404** Not Found
- **405** Method Not Allowed
- **500** Internal Server Error
- **501** Not Implemented

3.1.1.2.3 Aceptación de tipos de contenido

Cuando anteriormente hablamos sobre URLs, se mostró que no era correcto especificar el tipo de respuesta que se quería obtener. Para ello HTTP permite especificar en qué formato se quiere recibir el recurso, pudiendo indicar varios en orden de preferencia, para ello utilizamos el header **Accept**.

La API devolverá el recurso en el primer formato disponible, en caso de no mostrar el recurso en ninguno de los formatos indicados devolverá el código **HTTP 406**.

En la respuesta aparecerá con que formato se ha devuelto el recurso como podemos observar en la Figura 3.4:

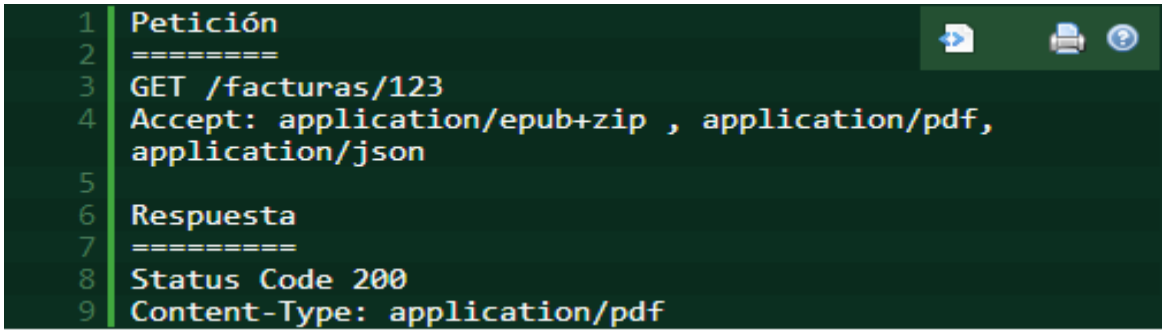


Figura 3.4 Content Type de la respuesta

El cliente pide el recurso en formato epub comprimido con zip, en pdf y en json, como podemos ver en la respuesta (**Figura 3.4**) el recurso es devuelto en formato pdf.

3.1.1.3 Hipermedia

A pesar de la sensación que pueda inducir el término hipermedia, el concepto y la finalidad es bastante sencilla: **conectar mediante vínculos las aplicaciones clientes con las APIs**, permitiendo a dichos clientes despreocuparse por conocer de antemano como acceder a los recursos.

Con hipermedia básicamente se añade información extra al recurso sobre su conexión a otros recursos relacionados con él.

En la Figura 3.5 se observa un ejemplo:

```
<pedido>
 <id>666</id>
 <estado>Procesado</estado>
 <links>
   <link rel="factura">

http://example.com/api/pedido/666/factura

   </link>
 </links>
</pedido>
```

Figura 3.5 Ejemplo Hipermedia

En el ejemplo Figura 3.5 se observa cómo indicar en un xml que representa un pedido, el enlace al recurso de la factura relacionada con el mismo.

Sin embargo, se necesita que el cliente que accede a la API entienda que esa información no es propia del recurso, sino que es información añadida que puede utilizar.

Esto se proporciona utilizando cabeceras Accept y Content-Type, para que tanto el cliente como la API sepan que se está hablando de hipermedia.

```
Petición
========
GET /pedido/666
Accept: application/nuestra_api+xml, text/xml

Respuesta
=========
Status Code: 200
Content-Type: application/nuestra_api+xml
Content:

<pedido>
 <id>666</id>
 <estado>Procesado</estado>
 <links>
   <link rel="factura">

http://example.com/api/pedido/666/factura

   </link>
 </links>
</pedido>
```

Figura 3.6 Ejemplo relación petición respuesta

Como vemos (Figura 3.6), el cliente solicita el formato application/nuestra_api+xml de forma preferente al formato text/xml. De esta forma, le indica al servicio web, que entiende su formato hipermedia y puede aprovecharlo.

El servicio web implementa hipermedia, este le devuelve la información del recurso y la información hipermedia que puede utilizar el cliente.

Hipermedia es útil para que el cliente no tenga que conocer las URLs de los recursos, evitando tener que hacer mantenimientos en cada uno de los mismos si en un futuro dichas URLs sufren modificación. También es muy útil para automatizar procesos entre APIs sin interacción humana.

3.1.1.4 ¿Por qué hemos utilizado REST?

Han sido varios los motivos por los cuales se ha decidido utilizar REST. Para comenzar vemos el rendimiento, utilizando REST la arquitectura se simplifica consiguiendo así un gran aumento a la hora del rendimiento.

Como segundo punto importante a destacar es el considerable aumento de velocidad debido a que las peticiones se simplifican.

No existe curva de aprendizaje con lo que se consiguen resultados óptimos y visualmente interpretables en poco tiempo.

Es muy sencillo escalar los proyectos que han sido relacionados con esta tecnología a pesar de toda la evolución que tengan los componentes.

Otro punto muy importante debido al tipo de trabajo que hemos desarrollado, una tesina de máster, es que otros compañeros pueden ampliar el trabajo. Este aumento de proyecto puede ser realizado mediante widgets o scripts, cosa que no penalizaría para nada ninguno de los puntos anteriores.

3.1.1.5 Otras tecnologías (SOAP)

SOAP (Joseph, 2002)**,** es un protocolo estándar que define como los objetos en diferentes procesos pueden comunicarse por medio de intercambio de ficheros XML.

En esta tecnología las operaciones son definidas como puertos WSDL, un puerto WSDL es un fichero XML donde se describe la forma de comunicación, es decir los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.

Este protocolo basado en XML consiste en tres partes: un sobre (envolope), el cual indica el mensaje y como procesarlo, un conjunto de reglas de codificación

para expresar instancias de tipos de datos y una tercera parte donde se representan llamadas a procedimientos y respuestas.

Podemos ver las tres partes en la Figura 3.7:

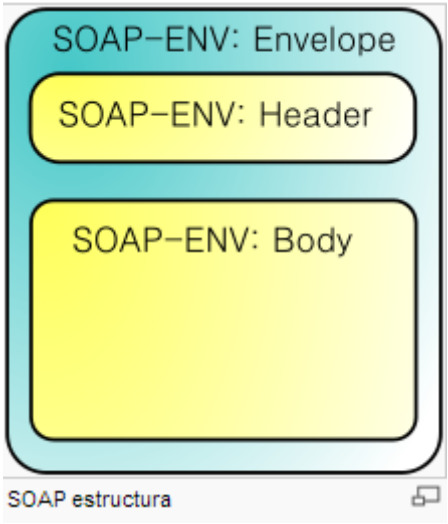


Figura 3.7 Estructura SOAP

SOAP dispone de tres características principales que son extensibilidad, neutralidad e independencia.

Otra tecnología que podíamos haber utilizado es RPC pero es muy antigua, y se ha descartado puesto que no se considera oportuno utilizar tecnologías antiguas.

3.1.1.5.1 SOAP vs REST

En este punto se realiza una comparación donde se explica la decisión de utilizar REST en contra de SOAP. (Pautaso, 2007)

SOAP es más útil utilizarlo cuando se establece un contrato formal donde se describen todas las funciones, el contrato formal se consigue como hemos visto anteriormente con el WSDL. También es necesario que la arquitectura aborde requisitos complejos no funcionales, y en nuestra tesina se resuelven recursos sencillos. Otro punto es la arquitectura, cuando esta maneja un proceso asíncrono debido al tiempo que necesita para realizar una parte de petición. Como podemos ver esta tecnología no conviene ya que no ayuda a solucionar el problema.

Por otro lado vemos cuándo es mejor utilizar REST, que es en los siguientes casos. El primer caso es cuando el servicio no necesita tener estado, cosa que cumplimos en nuestro trabajo. El siguiente caso es el más importante para

nosotros ya que REST es perfecto cuando el consumo se realiza desde un dispositivo móvil donde los recursos que tenemos son escasos. Además de esto tenemos otros factores explicados en otros puntos como la escalabilidad.

En la siguiente ilustración vamos a ver una gráfica donde podemos ver el estado de REST vs SOAP y otras tecnologías.

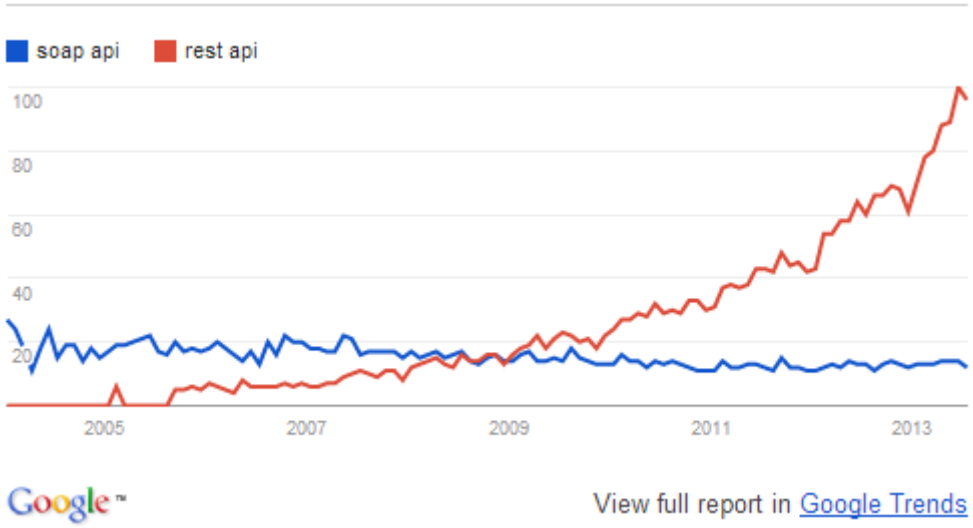


Figura 3.8 Crecimiento REST vs SOAP

En la Figura 3.8 se demuestra el gran aumento de uso de APIs desarrolladas en REST contra las SOAP estando seguros con esto que la tendencia es la correcta y hemos tomado una buena decisión a la hora de seleccionar la tecnología.

3.1.2 PHP

PHP es un lenguaje de programación de uso general, con la ejecución del código en el lado del servidor diseñado para el desarrollo web.

El código es interpretado por un servidor web con un módulo de procesamiento para PHP que es el encargado de generar la página web resultante.

Este lenguaje permite de una forma muy sencilla conectarnos a diferentes servidores de base de datos como MySQL, la que hemos elegido para nuestro proyecto. También tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos más comunes como son Linux, Windows, o Mac. Además de esto puede interactuar con los servidores web más populares ya que existe una versión CGI. También hay que tener en cuenta que es muy fácil instalar este lenguaje de programación en una máquina y no se requiere de una

configuración especial para poder correr las aplicaciones desarrolladas en PHP.

3.1.2.1 Características

Este lenguaje está desarrollado para el desarrollo de aplicaciones con acceso a base de datos. Es considerado un lenguaje sencillo de aprender, ya que en su desarrollo se simplifican distintas especificaciones como puede ser declarar variables primitivas. También es importante decir que el código PHP es invisible a los navegadores web y al cliente ya que es el servidor el que se encarga de ejecutarlo, creando así que la programación sea más segura.

Otro punto importante es la capacidad expandir su potencial de una forma muy sencilla, que es instalando extensiones. En cuanto a la documentación, dispone de una página web donde se explican todas las funciones explicitas con un ejemplo. Se permite la programación orientada a objetos y el manejo de excepciones.

PHP no obliga a una determinada metodología a la programación lo cual puede ser una ventaja o un inconveniente según la capacidad del programador. Por ellos existen Frameworks donde se facilitan las tareas comunes y permite crear una estructura MVC (Modelo, Vista, Controlador) de una forma más sencilla. Nosotros hemos utilizado SILEX, un Framework que explicaremos más adelante.

PHP también tiene una serie de inconvenientes, el primer inconveniente es que es un lenguaje interpretado y es más lento que un lenguaje compilado, aunque esto se puede mejorar con varios tipos de caching. Otro problema viene por la no tipificación de las variables dificulta a los IDEs asistencia a las variables, aunque IDEs como PHPStorm evitan esto creando un comentario en la definición de las variable.

3.1.2.2 ¿Por qué hemos elegido PHP?

Esta elección ha sido la más fácil, ya que disponemos de una experiencia con este lenguaje de programación. Además de esto nos proporciona todo lo necesario para crear una API de calidad, aunque como hemos dicho anteriormente no hemos utilizado PHP simple, hemos decidido utilizar Silex para que sea de gran ayuda a la hora de estructuración del código así como a evitar crear tareas básicas pudiendo centrar el desarrollo de la tesina en lo realmente importante, lo que da valor al trabajo.

3.1.3 Servicio Web

Un **Servicio Web** (David Booth, 2004) es una tecnología que un utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software pueden utilizar los servicios web para intercambiar información. La interoperabilidad se consigue mediante la adopción de entandares.

3.1.3.1 Ventajas de los servicios Web

Los servicios web proporcionan las siguientes ventajas:

- Proporcionar interoperabilidad entre aplicaciones de software independientemente de propiedades o plataformas.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan combinarse fácilmente para integrar servicios.

3.1.3.2 Inconvenientes de los servicios Web

- Su rendimiento es inferior si se compara con modelos de computación distribuida.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

3.1.3.3 ¿Por qué hemos creado servicios Web?

Existen varios motivos por los cuales hemos decidido trabajar con servicios web. (Milanovic, 2004) Como primer motivo podemos definir la separación de capas, con esto disminuimos el acoplamiento entre las piezas facilitando de

esta forma que puedan ser desarrolladas aplicaciones de terceros de una forma más sencilla, ya que solo es necesario que respeten la interfaz de los diferentes servicios web.

Otro motivo es que disponemos de una aplicación móvil, y los recursos son limitados. Los servicios Web no necesitan muchos recursos para ser creados ni para ser consumidos, así que podemos conectarnos con nuestro servidor y realizar las operaciones que necesitemos de una forma correcta.

3.1.4 JSON

JSON (JavaScript Object Notation) (Crockford, 2006) es un formato ligero de intercambio de datos. Es sencillo para las personas tanto leerlo y escribirlo, y para las maquinas es sencillo interpretarlo y generarlo. Está basado en un subconjunto del lenguaje de programación JavaScript. JSON es un formato de texto, que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de lenguajes como PHP, C#, Java, Perl, Python y otros muchos más. Estas propiedades hacen que JSON sea ideal para el intercambio de datos. Está constituido por dos estructuras:

- Una colección de pares nombre/valor. En otros lenguajes es conocido como una lista de claves, un arreglo asociativo etc.

- Una lista ordenada de valores. En la mayoría de lenguajes esto se implementa como arreglos, vectores, listas o secuencias.

Estas estructuras son soportadas por todos los lenguajes de programación. Como es independiente de lenguaje de datos se basa en unas estructuras. Las estructuras son las siguientes:

3.1.4.1 Representación de un JSON

Un **objeto JSON** (json org, 2013), es un conjunto desordenado de pares **nombre/valor.** Un objeto comienza por {(llave de apertura), y termina con } (llave de cierre). Cada nombre es seguido por: (dos puntos) y los pares nombre/valor están separados por una, (coma).

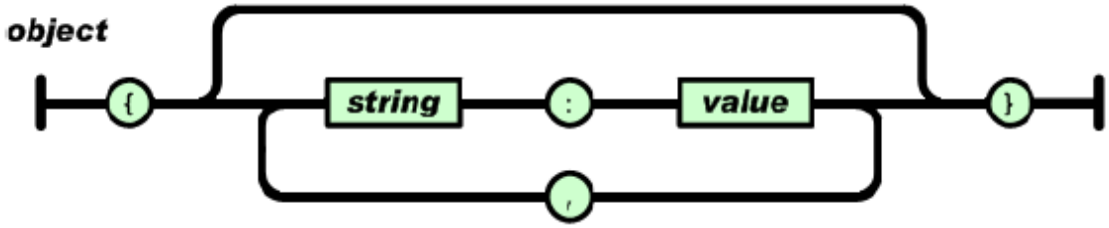


Figura 3.9 Objeto JSON

Un **Array JSON** es una colección de valores. Comienzan con [(corchete izquierdo) y termina con] (corchete derecho). Los valores se separan por, (coma).

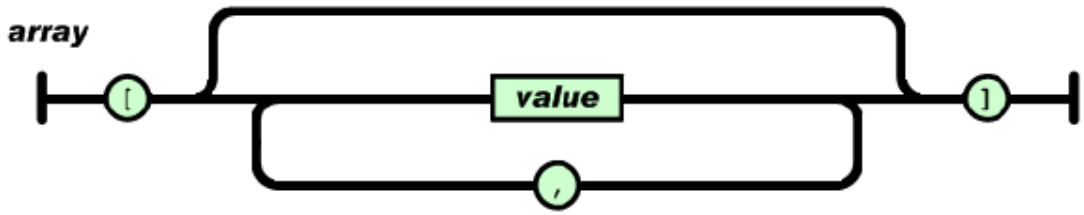


Figura 3.10 Array JSON

Un **valor JSON**, puede ser una cadena de caracteres con comillas dobles, un número, true, false o null, un objeto o un array. Estas estructuras pueden anisarse.

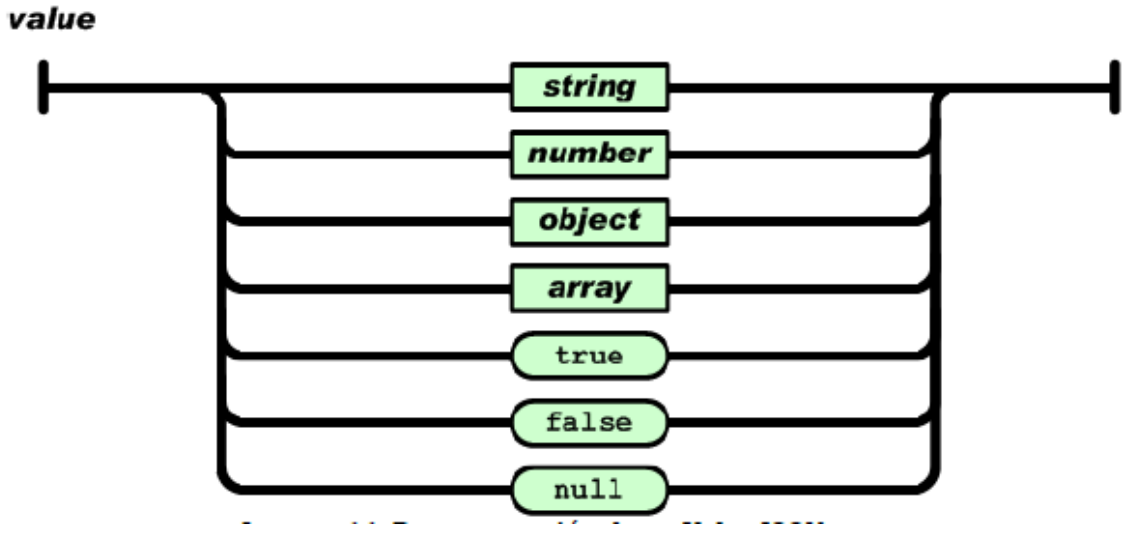


Figura 3.11 Valor JSON

Una **cadena de caracteres JSON**, es una colección de cero o más caracteres encerrados entre comillas dobles, usando barras divisorias invertidas como escape.

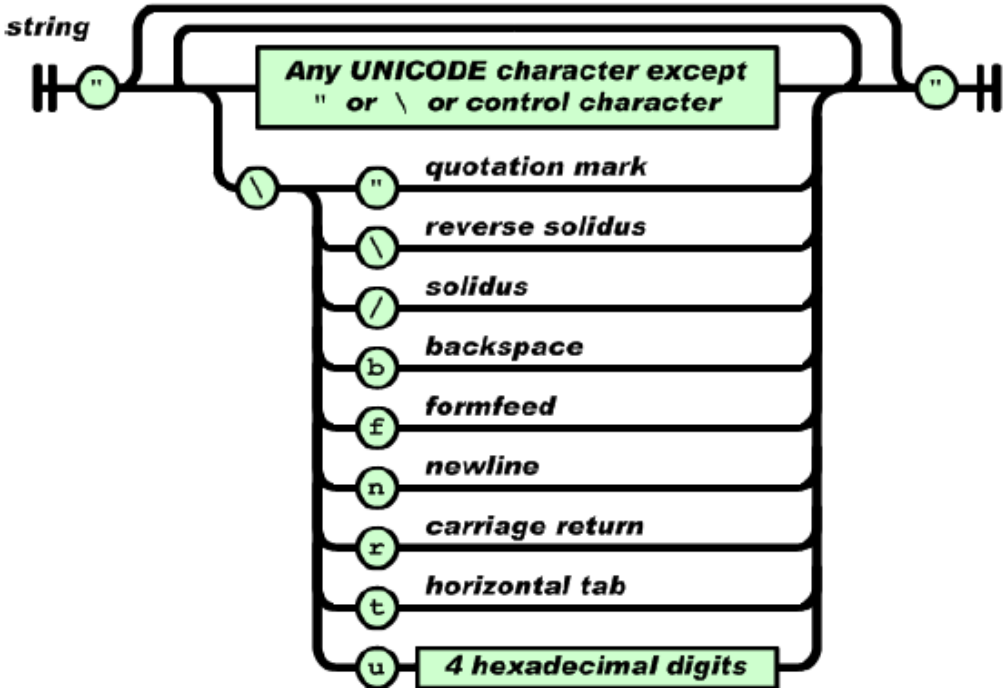


Figura 3.12 Cadena de valores JSON

La estructura de un **número JSON** es la siguiente:

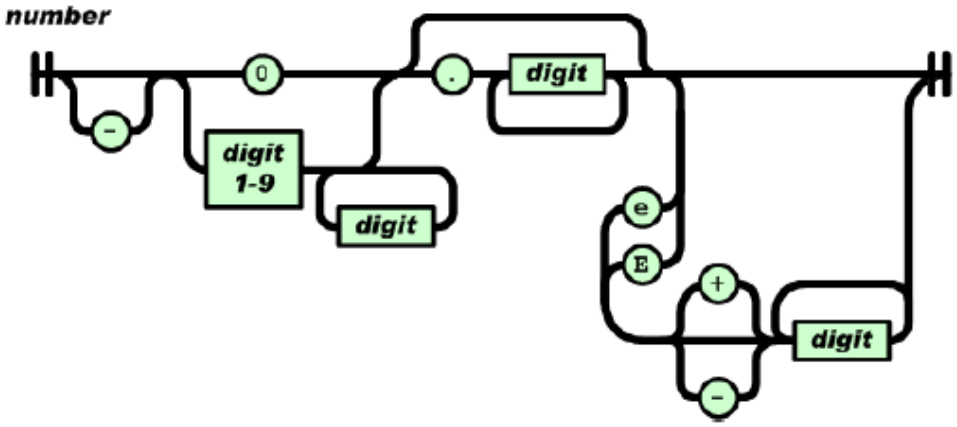


Figura 3.13 Número JSON

Código 3.2 y Código 3.3 son dos objetos JSON devueltos por la API cuando le pedimos una cierta información:

```
{
    "service_id":"s1",
    "obtrusivenessSpace_id":"OS2",
    "obtrusivenessLevel_id":"ol5",
    "device_os":"1",
    "sound":"1",
    "speech":"1",
    "vibration":"1",
    "light":"2",
    "statusBar":"1",
    "toast":"1",
    "dialog":"2",
    "desktopWidget":"1",
    "changeDateTime":"2012-02-09 11:24:00",
    "considerateSystemID":"CS1"
}
```

Código 3.2 Ejemplo JSON 1

```
{
    "service_id":"3",
    "obtrusivenessSpace_id":"idOS1",
    "obtrusivenessLevel_id":"idOL5",
    "device_os":"1",
    "sound":"1",
    "speech":"1",
    "vibration":"1",
    "light":"1",
    "statusBar":"1",
    "toast":"1",
    "dialog":"1",
    "desktopWidget":"3",
    "changeDateTime":"2012-01-20 00:00:00",
    "considerateSystemID":"CS1"
}
```

Código 3.3 Ejemplo JSON 2

3.1.5 Android

Android (Burnete, 2009) es un sistema operativo creado por Google orientado principalmente a dispositivos móviles aunque cada vez son más intensos los rumores que van a aparecer en otros dispositivos como ordenadores. El proyecto no fue creado por Google, sino por la compañía Android Inc, que posteriormente fue comprada por Google. Android fue presentado en el 2007

junto a la fundación Open Handset Alliance (OHA)[5]. El día 23 de Septiembre del año 2008 Google lanzó su primera versión del sistema operativo, basado en una versión del Kernel 2.6 de Linux y un mes más tarde se vendió el primer móvil con este sistema operativo. El sistema operativo está diseñado principalmente para dispositivos móviles con pantalla táctil como Smartphone o tablets.

La licencia del sistema operativo es software libre. Además de esto es open source con lo cual se permite acceder al código fuente y modificarlo creando así las ROMs personalizadas, utilizadas por usuarios expertos para personalizar sus dispositivos, o por las distintas compañías que trabajan con el creando una capa superior al núcleo para así intentar sacar el rendimiento máximo a los dispositivos.

El desarrollo del sistema no es sólo asunto de Google, también colabora OHA. Este consorcio está formado por multinacionales en el mundo de la tecnología como Samsung, LG, Nvidia o HTC. El objeto principal de este es desarrollar estándares del campo de la telefonía y Android es uno de ellos.

Para desarrollar para Android es necesario reunir todos los elementos de entorno y el control de las aplicaciones. Como lenguaje de desarrollo se utiliza Java y se proporciona un plugin para eclipse que contiene todas las herramientas necesarias. En este plugin se incluye un SDK que facilita al máximo el trabajo e intenta evitar las malas prácticas dentro de lo posible.

Una de las principales ventajas es que cualquier dispositivo Android puede ejecutar cualquier aplicación (siempre y cuando la versión instalada sea compatible) ya que es un sistema operativo multiplataforma.

3.1.5.1 Arquitectura Android

Para empezar a desarrollar aplicaciones debemos conocer como es la estructura de este sistema operativo. Para empezar diremos que está formado por varias capas o niveles, facilitando así la tarea al programador. Además, esta distribución permite acceder a las capas más bajas mediante librerías excluyendo así la necesidad de programar a bajo nivel las funcionalidades necesarias para utilizar las componentes hardware del dispositivo.

Cada capa utiliza elementos de la capa inferior para realizar funciones, este tipo de arquitectura es conocida como arquitectura pila, (Brahler, 2010).

[5] Consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles.

Figura 3.14 Arquitectura Android

A continuación se van a explicar las diferentes capas siguiendo una dirección ascendente:

1. **Kernel de Linux**: El núcleo actúa como capa de abstracción entre el hardware y el resto de las capas. El desarrollador no accede directamente a esta capa, sino se deben utilizar librerías disponibles en capas superiores. Uno de los motivos de utilizar estas librerías es la de evitarnos el hecho de conocer las características precisas de cada teléfono. Para cada elemento de hardware existe un driver (controlador) dentro del kernel que permite su utilización.

   El kernel también se encarga de gestionar los diferentes recursos del teléfono y del sistema operativo en sí.

2. **Bibliotecas**: Es la capa que siguiente al kernel. Esta capa está formada por las capas nativas de Android. Están escritas en c, c++ y compiladas para la arquitectura hardware especifica del teléfono. El objetivo de estas es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando que se tengan que codificar una y otra vez garantizando una mejor mantenibilidad y una mejor eficiencia.

   Entre las librerías se encuentra:

   - **OpenGL**: motor gráfico 3D basado en las APIs de OpenGL. Si el teléfono proporciona aceleración grafica es utilizada, en caso contrario se utiliza un motor software altamente optimizado.

- **Gestor de superficies (Surface Manager)**: se encarga de componer las imágenes que se muestran en la pantalla a partir de capas gráficas 2D, 3D. Cada vez que la aplicación pretende dibujar algo en la pantalla, la biblioteca no lo hace directamente sobre ella. En vez de eso, realiza los cambios en imágenes que almacena en memoria y que después combina para formar la imagen final que se envía a la pantalla.

- **Bibliotecas Multimedia**: basadas en OpenCore, permiten visualizar, reproducir e incluso grabar numerosos formatos de imagen, video y audio.

- **Webkit**: motor web utilizado por el navegador. Es el mismo motor que utiliza Google Chrome y Safari.

- **SSL (Secure Sockets Layer)**: proporciona la seguridad necesaria al acceder a Internet por medio de criptografía.

- **FreeType**: permite mostrar fuentes tipográficas, basadas tanto en mapas de bits como en vectores.
- **SQLite**: motor de BBDD relacional, disponible para todas las aplicaciones.

- **SGL**: desarrollada por Skia, utilizada tanto en Android como en Google Chrome, se encarga de representar elementos de dos dimensiones, Es el motor gráfico 2D de Android.

- **Biblioteca C de sistema (libc)**: está basada en la implementación de Berkeley Software Distribución (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

3. **Entorno de ejecución**: Como podemos ver en la ilustración el entorno de ejecución se encuentra dentro la capa de librerías, aunque nosotros hemos dicho que se iba a separar en 5 capas esta capa no es una capa en sí, aunque se puede considerar si lo consideras así. En este lugar se encuentran las librerías con las funcionalidades habituales de Java así como otras específicas.

   El componente principal del entorno es la máquina virtual **Dalvik**. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que la máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán

listas para distribuirse con la garantía de serán ejecutables en cualquier dispositivo que disponga de una versión Android compatible.

Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el bytecode Java. Java se usa como lenguaje de programación, y los ejecutables que genera el SDK de Android tienen la extensión .dex que es específico para Dalvik, por ello no se pueden ejecutar aplicaciones Java en Android ni viceversa.

4. **Framework de aplicaciones**: La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes son librerías Java que acceden a través de la máquina virtual.

   Siguiendo el diagrama encontramos:

   a. **Activity Manager**: Encargado de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.

   b. **Windows Manager**: Se encarga de organizar lo que se mostrará en pantalla que posteriormente pasarán a ser ocupadas por las actividades.

   c. **Content Provider**: Esta librería es la encargada de crear una capa de encapsulación que se compartirá entre aplicaciones para tener control sobre cómo se accede a la información.

   d. **Views**: Son las encargadas de construir las interfaces de usuarios, están formadas por elementos simples como botones, cuadros de texto, listas o elementos más complejos como navegadores web o visor de Google Maps.

   e. **Notification Manager**: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de notificaciones. Esta biblioteca permite utilizar sonidos, activar las vibraciones, o los LEDs de notificación si el dispositivo dispone de ellos.

   f. **Package Manager**: Esta biblioteca da la información sobre los paquetes instalados en el dispositivo además de gestionar la instalación de nuevos. Estos paquetes tienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.

g. **Telephony Manager**: Librería encargada de manejar todo lo relacionado con las llamadas, los SMS/ MMS, esta librería no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

h. **Resource Manager**: Con esta librería gestionamos todos los elementos que forman la aplicación pero están fuera del código, es decir, traducciones, imágenes, sonidos, layouts.

i. **Location Manager**: Permite determinar la posición geográfica del dispositivo mediante GPS, triangulación u otras técnicas para poder trabajar con mapas.

j. **Sensor Manager**: permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensores de luminosidad, campo magnético, presión, proximidad, temperatura….

k. **Cámara**: Con esta librería podemos hacer uso de todas las cámaras que tenga el dispositivo, para tomar instantáneas o grabar videos.

l. **Multimedia**: Permiten reproducir y visualizar audio, video, imágenes.

5. **Aplicaciones**: En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz como las que no, las nativas y las administrativas, las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha decidido instalar.

   En esta capa encontramos también la aplicación principal del sistema: el lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde podemos colocar accesos directos a aplicaciones o *widgets*.

   Como podemos observar esta separación por niveles proporciona un entorno sumamente poderoso para que podamos programar aplicaciones que hagan cualquier cosa. Nada de Android es inaccesible y podemos jugar siempre con las aplicaciones de nuestro teléfono para optimizar cualquier tarea.

   Uno de los potenciales de Android es que permite al usuario el control total de su dispositivo para poder crear un entorno a medida.

3.1.5.2 Componentes de una aplicación Android

Una aplicación Android está construida por varios bloques esenciales. Los bloques son los siguientes:

- Activities
- Services
- Content Provider
- Broadcast Receivers
- Fragments

A continuación vamos a explicar cada uno de los componentes anteriormente nombrados:

3.1.5.2.1 Activities

Una ***Activity*** es cada una de las interfaces de usuario que podemos encontrarnos en una aplicación Android, es decir, es como los sistemas Android representan las interfaces graficas con las que el usuario del sistema podrá interactuar. Para crear una *Activity* únicamente debemos extender de la clase *Activity* proporcionada por Android.

Con lo que hemos visto anteriormente podemos decir qué una aplicación está formada por un grupo o un conjunto de *Activities* relacionadas entre sí. Cuando tenemos una aplicación disponemos de una *Activity* principal que es la primera que se lanza cuando ejecutamos la aplicación. Esta *Activity* principal podrá darnos acceso a otras. Es muy importante saber que cuando se inicializa una *Activity* se almacena en una pila con el algoritmo LIFO[6]y la *Activity* anterior pasa a un estado parada.

Como hemos visto una *Activity* dispone de muchos estados, por ello existe un conjunto de métodos que se pueden sobrecargar en cualquier momento, para realizar acciones en cada uno de los estados en los que se encuentre.

A continuación vamos a ver la figura Figura 3.15 con todos los estados y el flujo de ejecución de los mismos, explicando para cada estado como funciona.

[6] Last In, First Out.

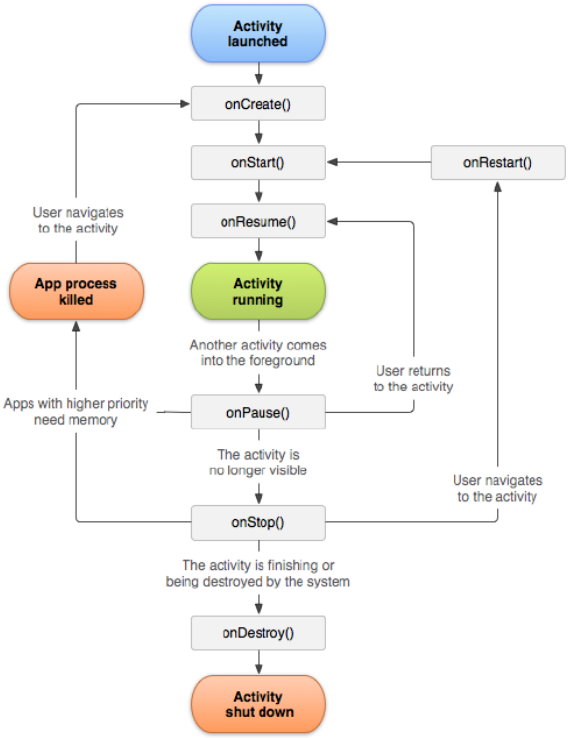


Figura 3.15 Estados y Flujos de una Activity

- **onCreate()**: Se llama cuando se ejecuta una *Activity* por primera vez.

- **onRestart()**: Se llama después de que una *Activity* haya sido parada, justo antes de que se inicialice de nuevo, y siempre seguido de *onStart()*.

- **onStart()**: Se llama siempre antes de que la *Activity* se haga visible al usuario. Seguida por *onResume()*; si la *Activity* a primer plano o por onStop(); si va a permanecer oculta.

- **onResume()**: Se llama justo antes de que la *Activity* comience la interacción con el usuario. En este punto, la *Activity* está en la cima de la pila. Siempre va seguida de *onPause()*.

- **onPause():** Se llama cuando el sistema va a reanudar otra Activity. En este método se guardan datos, se paran animaciones u otras cosas que puedan consumir CPU. Debe ser un método rápido ya que la siguiente *Activity* no se reanuda hasta que esta no termine sus acciones. Es seguida por *onResume()*; si va a primer plano, o por *onStop()*; si se hace invisible al usuario.

- **onStop():** Se llama cuando la *Activity* y no va a ser visible al usuario bien por que vaya a ser destruida o bien porque va a pasar a un

segundo plano. Va seguida de *onRestart()***;** si vuelve a ser visible para el usuario, o por *onDestroy()*; si va a desaparecer.

- **onDestroy():** Se llama antes de que la *Activity* se destruya. Es la última llamada que recibe una *Activity*. Puede llamarse por dos motivos, que la *Activity* haya sido finalizada, o por que el sistema necesite espacio y decida eliminarla.

3.1.5.2.1.1 Crear una Activity

Como hemos explicado anteriormente, para crear una *Activity* lo único que es necesario es extender de la clase *Activity.* En la case hija debemos implementar los métodos que necesarios por la herencia más los métodos que necesitemos para controlar nuestra *Activity.*

La interfaz de usuario de una *Activity* está formada por una jerarquía de vistas. Cada vista controla un espacio dentro de la ventana de la *Activity*. Estas son las encargadas de responder a las interacciones de los usuarios.

3.1.5.2.2 Services

Un servicio es un componente que mejora el rendimiento de operaciones largas en segundo plano y no provee una interfaz de usuario.

Otro componente puede arrancar un servicio y este continuara en segundo plano incluso si el usuario cambia a otra aplicación. Además un componente puede enlazar con un servicio para interactuar con el incluso realizar interprocesos de comunicación. Por ejemplo un servicio puede manejar transacciones de red, reproducir música, todo desde el backgroud.

Un servicio puede adoptar dos formas:

- **Started:** Un servicio esta “started” cuando un componente lo lanza ejecutando una llamada a “startService”. Una vez lanzado, un servicio puede correr en segundo plano indefinidamente, incluso si el componente que lo ha lanzado se destruye. Normalmente, un servicio en ejecución realiza operaciones simples y no devuelve ningún resultado al componente que lo ha inicializado.

- **Bound**: Un servicio esta “bound”, limitado, cuando un componente se une a él haciendo una llamada al método bindService(). Un servicio en este estado ofrece una interfaz cliente servidor que permite a los componentes interactuar con el servicio, enviar peticiones, obtener

resultados. Un servicio “bound” corre solo mientras otro componente está ligado a él. Muchos componentes pueden unirse o enlazarse al servicio una vez, pero cuando todos los componentes se han desligado, el servicio será destruido.

Creación de un Service

Para crear un Service se debe crear una subclase de Service. En su implementación, se deben sobrescribir algunos métodos, para controlar algunos aspectos claves del ciclo de vida de un Service. Los métodos más importantes a sobrescribir son:

- **onStartCommand()**: El sistema llama este método, cuando otro método inicia un Service mediante el método startService(). Una vez que se ejecuta este método, se inicia el Service en background indefinidamente, es responsabilidad del programador, parar el Service cuando haya terminado su acometido, llamando al método stopSelf() o al método stopService().

- **onBind()**: El sistema llama a este método cuando otro componente se quiere ligar a este Service, llamando al método bindService(). Este método se ha de implementar siempre, pero si no se desea hacer ligamientos; simplemente se devuelve “null”.

- **onCreate()**: El sistema llama a este método cuando el Service se crea por primera vez, este se ejecuta antes de llamar a los dos métodos citados anteriormente.

- **onDestroy()**:El sistema llama a este método cuando el Service no se va a usar más, y va a ser destruido. Un servicio debe implementar este método para limpiar cualquier recurso, como por ejemplo, Threads, registered listeners, Receivers, etc. Esta es la última llamada que recibe un Service.

La ejecución más común de un Service, se realiza mediante la llamada al método onStartCommand(), dicho método devuelve un Integer. El Integer, es un valor que describe como debe continuar el servicio, en el caso de que el sistema lo destruya. Para ello, se tienen los siguientes Integers de retorno:

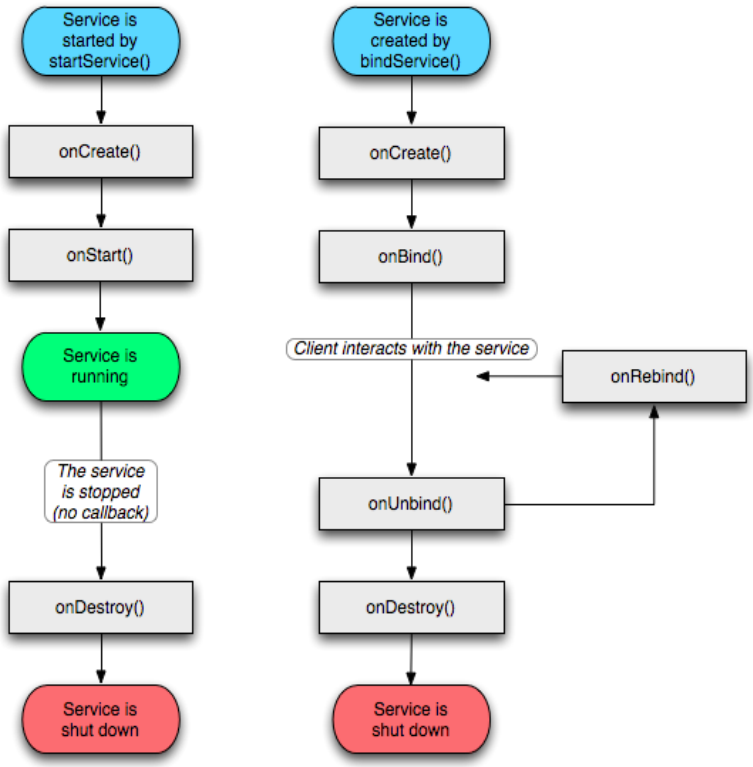


Figura 3.16 Ejecución de un Service

3.1.5.2.3 Content Provider

Un Content Provider, es el encargado de gestionar un conjunto de datos de la aplicación, que se han de compartir. Los datos pueden ser almacenados en: un sistema de archivos, una base de datos SQLite, en la Web o en cualquier otro lugar persistente, a la que la aplicación pueda acceder.

Mediante un Content Provider, otras aplicaciones pueden hacer consultas o incluso modificar los datos (si el Content Provider lo permite). Por ejemplo, el sistema Android proporciona un Content Provider que gestiona la información de los contactos del usuario, por tanto, una aplicación que tenga los permisos adecuados, podrá consultar parte del Content Provider (como ContactsContract.Data), para leer y escribir información sobre una determinada persona. Los Content Provider también son adecuados para leer y escribir datos que son propios de una aplicación, y no deben ser compartidos.

Un Content Provider se ha de implementar como una subclase de ContentProvider, y debe implementar un conjunto estándar de APIs, que habilitan otras aplicaciones para ejecutar transacciones.

Cuando se desea acceder a los datos de un Content Provider, se ha de hacer mediante el uso de un objeto ContentResolver, que esté dentro del contexto de la aplicación, para realizar la comunicación con el Provider como un cliente. El

funcionamiento es el siguiente: el objeto ContentResolver se comunica con el objeto Provider (una instancia de la clase que implementa ContentProvider). Por su parte el objeto Provider recibe las solicitudes de los datos desde los clientes, ejecuta las acciones solicitadas y devuelve los resultados.

3.1.5.2.4Broadcast Receivers

Un Broadcast Receiver, es un componente que responde a la emisión de anuncios/notificaciones en todo el sistema. Existen muchos Broadcasts que los origina el sistema, como por ejemplo: Broadcasts que avisan que la pantalla se ha apagado, que la batería está baja, o que se ha capturado una imagen. Las aplicaciones por su parte también pueden iniciar Broadcasts, por ejemplo, para informar a otras aplicaciones que ya se han descargado los datos en el dispositivo, y pueden ser usados.

Aunque los Broadcast Receivers no muestran una interfaz de usuario, estos pueden crear una notificación StatusBar, para alertar al usuario cuando ocurra un evento Broadcast. Sin embargo, los Broadcast son usados más comúnmente, como una puerta (“Gateway”) a otros componentes, y tienen como propósito, realizar el menor trabajo posible. Por ejemplo, este puede instanciar un Service, para que ejecute algún trabajo basado en el evento.

Un Broadcast Receiver se implementa como una subclase de BroadcastReceiver, y cada Broadcast se envía mediante un Intent.

3.1.5.2.5 Fichero Manifest

El manifest en un fichero XML que se ha de crear para cada aplicación, dicho fichero se debe llamar: AndroidManifest.xml. Éste contendrá, la declaración de cada uno de los componentes, de los que está compuesta la aplicación a la que pertenece. Otras de las cosas que se han de declarar en el manifest son:

- Todos los permisos de usuario que requiera la aplicación, como por ejemplo, acceso a Internet, acceso para la lectura de los contactos del usuario, etc.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Código 3.4 Permisos fichero manifest.xml

- El API Level mínimo requerido por la aplicación, esto es, la versión mínima de Android, para la cual la aplicación es operativa, Código 3.5.

```
<uses-sdk
    android:minSdkVersion="17"
    android:targetSdkVersion="17" />
```

Código 3.5 Versión fichero manifest.xml

- Las características de hardware y software, requeridos por la aplicación (Cámara, Bluetooth, etc.).

- Librerías API que se deben enlazar con la aplicación (diferentes al API de Android), como por ejemplo, las librerías de Google Maps.

3.1.5.2.6 Fragment

Como es conocido en los últimos años han aparecido dispositivos de gran tamaño, tipo tablets, por ello el equipo de desarrollo de Android tuvo que solucionar el problema de la adaptación de la interfaz gráfica de las aplicaciones a ese nuevo tipo de pantallas. Una interfaz de usuario diseñada para un dispositivo móvil no se adaptaba a pantallas de 7 pulgadas o superiores. La solución a esto vino en forma de un componente, llamado Fragment.

Un Fragment podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades. Esto permite poder dividir nuestra interfaz en varias porciones de forma que podamos diseñar diversas configuraciones de pantalla, dependiendo de su tamaño y orientación, sin tener que duplicar código en ningún momento, sino tan solo utilizando o no los distintos fragmentos para una de las posibles configuraciones.

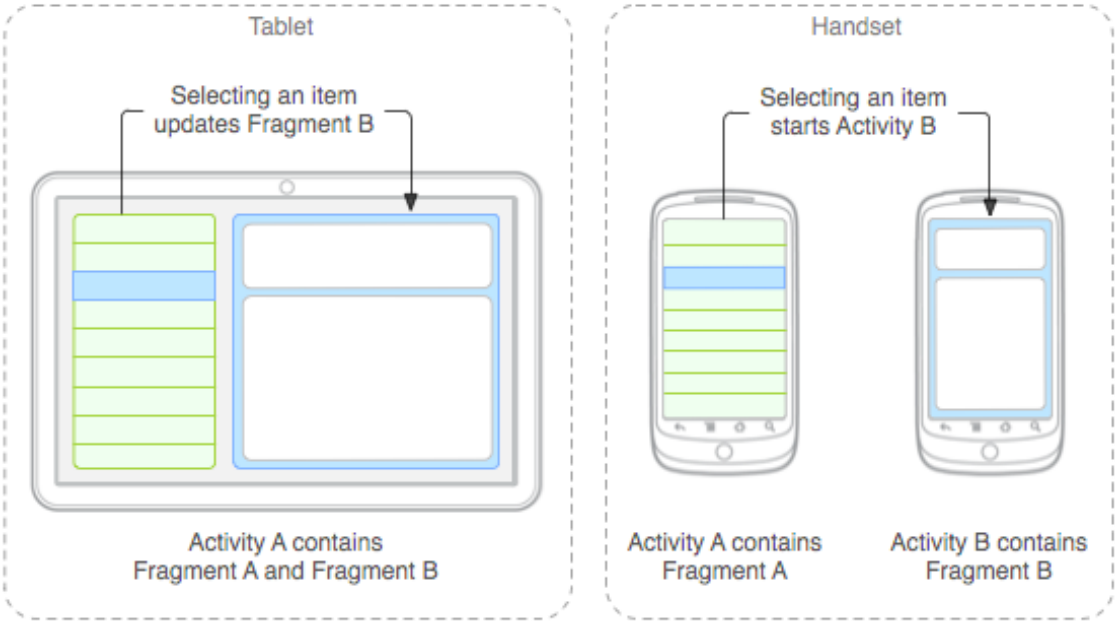


Figura 3.17 Ejemplo Fragment

3.1.6 Patrón de diseño Singleton

El patrón de diseño **Singleton** (Gamma E., 1995), (Jens Jahnke, 1997) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia proporcionando un punto de acceso global a ella.

Para crear el patrón singleton se necesita crear una clase que contenga un método que sea encargado de crear la instancia en caso que no exista ninguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el acceso al constructor.

Este patrón se implementa por ejemplo cuando se controla el acceso a un recurso físico, cuando un cierto tipo de dato debe estar disponible para todos los demás objetos de la aplicación.

El patrón provee una única instancia global gracias a que:

- La propia clase es capaz de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.

- Declara el constructor de clase como privado para que no se pueda instanciar directamente.

3.1.6.1.1 ¿Por qué hemos usado el patrón Singleton?

Como hemos explicado anteriormente el patrón Singleton sirve para asegurarnos que solo existe una instancia de una clase y que puede ser accedida en toda la aplicación. En toda la aplicación necesitamos tener acceso a una variable global donde conozcamos en todo momento el servicio con el cual queremos trabajar. Además de conocer servicio solo podemos trabajar con un servicio en concreto.

De esta forma el patrón Singleton soluciona nuestros dos problemas y es una buena forma de solucionarlos.

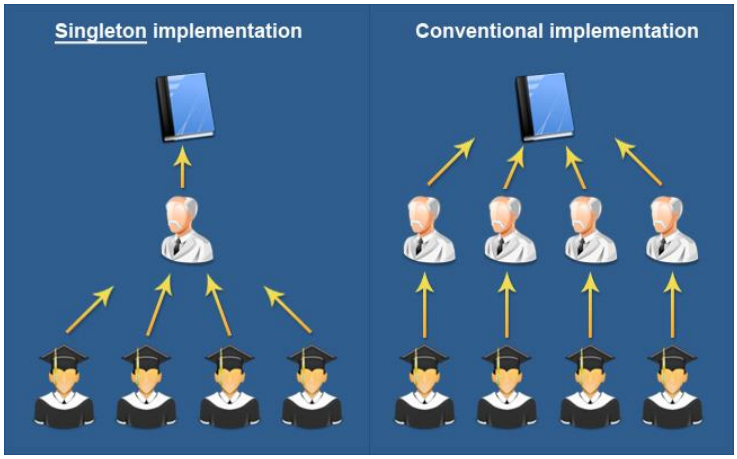


Figura 3.18 Ilustración patrón Singleton

3.1.7 Doctrine

Doctrine (Doctrine, 2013) es un mapeador de objetos-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa por encima del SGBD en nuestro caso MySQL.

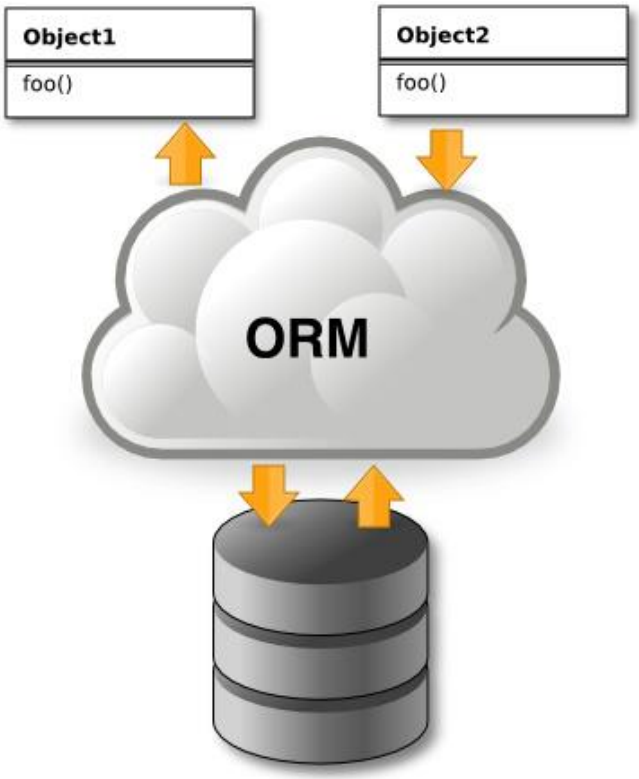


Figura 3.19 ORM

Un ORM es una técnica de programación que permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, es decir, las tablas de nuestra base de datos pasan a ser clases y los registros podemos manejarlos con facilidad.

3.1.7.1 ¿Por qué utilizar Doctrine?

Utilizar un ORM proporciona en una serie de ventajas que facilitan enormemente tareas comunes y de mantenimiento:

Reutilización: La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.

1. **Encapsulación:** La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función

2. **Portabilidad:** Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin

ningún tipo de complicación. Esto es debido a que no utilizamos una sintaxis MySQL, Oracle o SQLite para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.

3. **Seguridad:** Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques más comunes.

4. **Mantenimiento del código:** Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla.

3.1.7.2 Utilización de Doctrine

Doctrine se basa en el active record pattern para trabajar con datos, en los que una clase se corresponde con una tabla de la base de datos. Por ejemplo, si un programador quiere crear un objeto de una tabla llamada “User” en la base de datos, podría no escribir ninguna sentencia SQL, simplemente podría hacer lo siguiente:

```
$user = new User();
$user->setName('User Name');
$user->setPhone('699294657');

$em->persist($user);
$em->flush();
```

Código 3.6 Ejemplo Doctrine

Con las instrucciones anteriores crearíamos un objeto usuario con su nombre y su número de teléfono. No solo utilizamos esta funcionalidad de Doctrine, en utilizamos la facilidad con la que permite crear la conexión a la base de datos.

Vamos a ver un ejemplo a continuación con el método update() de Doctrine, donde veremos que pasándole, el nombre de la tabla, el id del objeto a modificar y el array de datos con los valores es capaz de transformar a SQL y realizar la actualización:

```
public  function update($id, array $data)
{
    $this->db->update($this->tableName, $data, array('id' => $id));

    return $this->find($id);
}
```

Código 3.7 Método update

3.1.7.3 Configuración de la Base de Datos

Una vez se ha visto cómo trabaja Doctrine se va a observar cómo se configura en la aplicación para que sepa que base de datos debe atacar. Esto se hace mediante un array de parámetros. Donde se indica con un par clave valor que opción de configuración se va a definir. Una característica de Doctrine es el bajo nivel de configuración que necesita para empezar un proyecto.

Podemos ver un ejemplo en Código 3.8:

```
$app->register(new Silex\Provider\DoctrineServiceProvider(), array(
    'dbs.options' => array (
        'mysql_read' => array(
            'driver'   => 'pdo_mysql',
            'dbname'   => 'notificationsV2',
            'user'     => 'root',
            'password' => 'root',
            'port'     => 3306,
            'path'     => __DIR__.'/app.db',
        )
    )
));
```

Código 3.8 Registrar Doctrine

Con este ejemplo tenemos a conexión a la Base de datos con el nombre notificationsV2, creada en MySQL, con un usuario llamado root al igual que su password, etc.

También es interesante saber que Doctrine lleva por defecto parámetros que los estándares recomiendan utilizar, como puede ser el puerto 3306 en nuestro caso, es decir, podríamos quitarlo y seguiría funcionando.

3.1.7.4 Características

Doctrine es capaz de generar clases a partir de una base de datos existente y después el programador puede especificar relaciones, añadir funcionalidad sin la necesidad de generar o mantener esquemas en XML.

Otra característica que hemos utilizado es la de poder utilizar SQL directamente, pero no solo permite escribir consultas en SQL sino también en DQL que es un dialecto de SQL que facilita la creación de consultas complejas.

Otras características notables son:

- Soporte para datos Jerárquicos.
- Soporte para Hooks
- Utilización de caches.
- Transacciones ACID.
- Es capaz de unir varios archivos del framework en uno solo para no penalizar el rendimiento.

3.1.7.5 - Influencias

Doctrine tiene influencias de docenas de proyectos de personas muy diferentes. Las mayores influencias son de Hibernate (el ORM de Java) y de ActiveRecord (de Ruby on Rails). Ambos tienen una implementación completa tanto en Java como en Ruby.

El propósito de Doctrine es construir una solución igual de potente para PHP.

4 Desarrollo de la Propuesta

Este cuarto capítulo se divide en cuatro partes; en la primera se van a explicar los requisitos del trabajo, bien sean funcionales o no funcionales. En la segunda parte se desarrolla el proceso de diseño de la tesina. Para continuar, en el tercer apartado del capítulo se va a explicar el proceso de implementación y por último la parte encargada de comprobar el correcto funcionamiento del trabajo propuesto.

Hay que comentar que se han utilizado diferentes metodologías de trabajo para cada una de las propuestas ya que estas se adaptan mejor para solucionar los problemas. Para el desarrollo de la API se ha optado por utilizar una metodología guiada por pruebas y para el desarrollo de la aplicación Android se ha decidido utilizar un desarrollo incremental

4.1 Requisitos

En la tesina se ha desarrollado una herramienta de visualización de adaptaciones end-user a través de la cual los usuarios finales son capaces de gestionar las auto-adaptaciones de la configuración para cada servicio permitiendo hacer rol back a adaptaciones previas de una forma user-friendly.

A continuación vamos a mostrar las funcionalidades y los requisitos (Pressman, 2002) que debe cumplir.

4.1.1 Requisitos funcionales

Un requisito funcional define una función del sistema software. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los detalles funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos etc. Es decir, funcionalidades que un sistema debe cumplir.

Los requisitos funcionales más importantes son los siguientes:

- **API de comunicación**: debemos proporcionar una API que sirva de conexión entre el dispositivo móvil y el servidor de almacenamiento de los datos.

- **Aplicación Android**: nuestro objetivo es proporcionar una aplicación Android donde se muestren todos los servicios disponibles. Tenemos que mostrar el estado actual de la configuración del dispositivo (adaptación actual), así como un historial de las configuraciones pasadas. Además podemos restaurar el dispositivo a una configuración anterior.

La aplicación debe disponer de una lista de servicios activos en el momento de una forma sencilla, pudiendo seleccionar uno u otro rápidamente. Una vez hemos seleccionado el servicio se debe proporcionar un listado con detalle de la configuración de interacción actual para ese servicio (estado actual), viendo para cada una de las opciones disponibles (vibración, brillo, sonido, etc.) como se encuentran.

Por otra parte debemos proporcionar un historial de las adaptaciones del servicio ordenadas cronológicamente de forma descendente, mostrando los historiales del más actual al más antiguo.

Por último, si el usuario considera que hay un estado de configuración anterior que se adapta mejor a su situación actual o sus preferencias, puede restaurar el sistema a ese estado. De esta manera se creará un nuevo historial y se podrá acceder al mismo si se considera necesario, bien para restaurarlo o simplemente para consultarlo.

La interfaz debe ser de forma sencilla, amigable y sin mucha complejidad a la hora de utilizarse.

4.1.2 Requisitos no funcionales

Un requisito no funcional es un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos. Los requisitos no funcionales más importantes del proyecto son:

- **Rendimiento**: la aplicación debe tener un rendimiento aceptable y esto se consigue mediante pocos accesos a la API almacenando los datos en estructuras de datos aumentando considerablemente el rendimiento.

- **Usabilidad**: hemos desarrollado la aplicación con unos patrones de diseño haciendo que esta sea de una gran usabilidad.

- **Portabilidad**: teniendo en cuenta que es una aplicación móvil decimos que cumplimos este requisito sin problemas.

- **Escalabilidad**: toda la aplicación ha sido desarrollada de una forma muy modular haciendo que la escalabilidad sea posible y de una forma muy sencilla.

- **Mantenibilidad**: con la utilización de test creamos una mantenibilidad muy buena ya que podemos asegurarnos si la funcionalidad es correcta a pesar de nuevos módulos.

Estos son los requisitos más comunes pero nosotros dispones de algunos más específicos como son los siguientes:

- **Dispositivo Android**: para poder utilizar la aplicación necesitamos un dispositivo con este sistema operativo.

- **Event Provider**: se necesitan de proveedores de eventos para que el móvil pueda tener avisos para el cambio de preferencias

- **Conexión a Internet**: Para poder realizar una utilización completa de la aplicación es necesario disponer de una conexión a internet.

- **Servicios**: estos son los servicios de los que se va a obtener un historial de preferencias para poder restaurarlas si se desea.

4.2 Diseño

El proyecto ha sido diseñado en varios bloques, para cada uno de estos se ha seguido una metodología distinta, que se ha considerado la más oportuna para este tipo de problema. Una vez hemos tenido todos los bloques desarrollados por separado solo hemos tenido que acoplarlos consiguiendo así una aplicación con un bajo grado de acoplamiento.

Los diversos bloques que hemos construido han sido:

1- Base de Datos MySQL

2- API REST en PHP.

3- Aplicación Android.

Como se puede observar las 3 partes anteriores podemos dividirlas en 2 apartados, el primero sería la parte del servidor que incluiría la base de datos y la API que ofrece una conexión para así poder comunicarnos con ella sin

ningún problema siempre y cuando accedamos de una forma correcta a las URL pasándole los parámetros necesarios y con el método específico para cada una. El tercer punto de los anteriores es la parte cliente, la aplicación va a ser la encargada de comunicarse con la base de datos mostrándonos la información en interfaces gráficas para facilitarnos de una forma muy sustancial la utilización de la aplicación.

4.2.1 Base de Datos

En este apartado se va a realizar una breve descripción de la base de datos que se utiliza para entender de una forma más concreta la implementación interna de la parte servidor desarrollada en la tesina.

4.2.1.1 Estructura de la Base de Datos

En la Figura 4.1 se muestra la estructura de la base de datos para poder disponer de una idea general de cómo ha sido pensada, diseñada y finalmente creada.

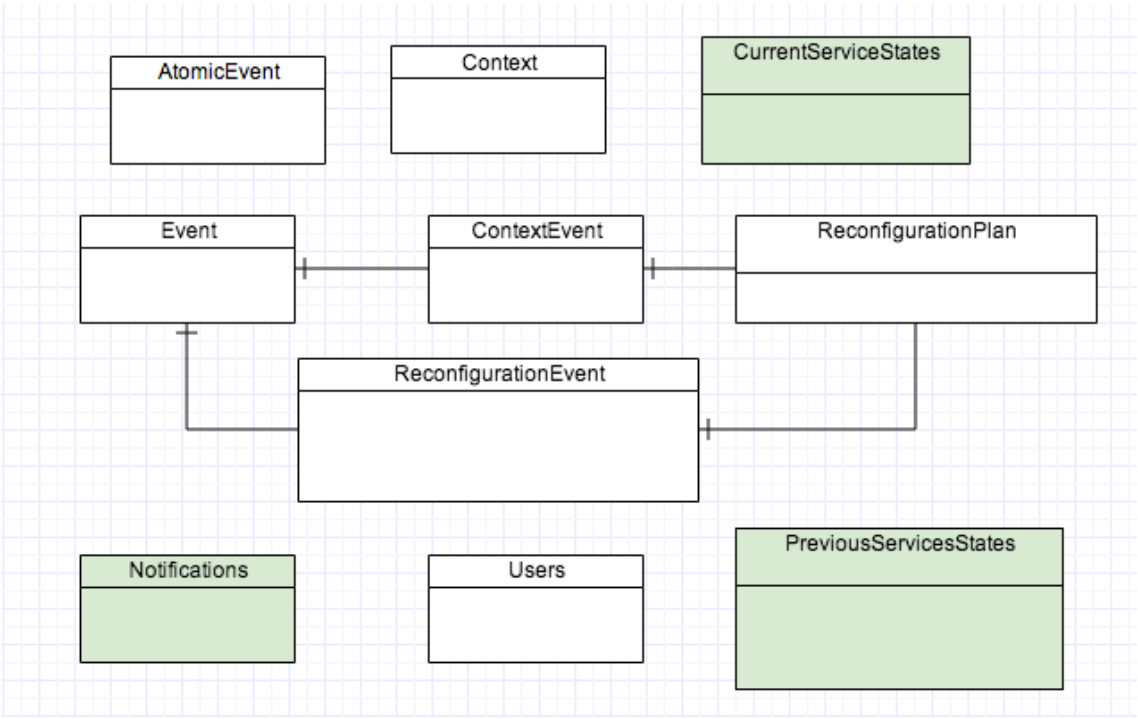


Figura 4.1 Estructura de la Base de Datos

En cuanto a la estructura debemos indicar que hemos pintado de un color más oscuro las tablas utilizadas directamente por nuestra aplicación.

4.2.1.2 Tablas utilizadas

A pesar de que la Base de Datos está formada por una mayor cantidad de tablas para el desarrollo de nuestro proyecto únicamente nos centramos en el acceso a las siguientes:

4.2.1.2.1 Notifications

Esta tabla es la encargada de almacenar todas las notificaciones de los dispositivos. De esta tabla se consigue la lista de los servicios que se disponen para poder mostrar la información relacionada de cada uno de estos.

4.2.1.2.2 Previous Service States

En esta tabla se almacenan todos los estados por los cual ha pasado un servicio. Es decir se buscan las instancias de un determinado servicio logrando crear el historial relacionado.

4.2.1.2.3 Current Service States

Esta última tabla de acceso es la encargada de persistir los datos de las configuraciones actuales de todos los servicios. Esta es la tabla que se accede para mostrar la vista de los estados actuales del servicio seleccionado

4.2.2 API de comunicación

Vamos a comenzar este punto explicando que es, una **API** (Application Programming Interface) o **IPA** (Interfaz de programación de aplicaciones) es un conjunto de funciones y métodos que ofrece una biblioteca para ser utilizada por otro software como una capa de abstracción. Debido al gran crecimiento de las APIs es muy importante que esta sea correcta y por ello hemos utilizado el artículo “How to design a good API and why it matters” (Bloch, 2006). Esta definición carece de sentido para lectores que no dispongan de una base en lenguaje informático así que vamos a explicarlo en un lenguaje más entendible.

Hace tiempo la web dejó de ser una gran masa de páginas enlazadas entre sí mediante links. Los programadores enlazamos también funcionalidades desde una web a otra y para ello se deben crear librerías para poder utilizar las funcionalidades que ofrecen.

Un ejemplo de esto podemos verlo claramente con Google, a menudo entramos a la web a buscar información sobre empresas, y vemos que en sus portales web aparece el típico apartado donde aparece localización o donde estamos y aparece un frame con un mapa extraído de Google maps. Esto se puede hacer gracias a que Google proporciona una API para poder incluir sus mapas en nuestra web. Otro ejemplo es Youtube, ya que de una forma muy sencilla permite embeber videos en nuestra página web.

Como último vamos a ver el ejemplo de otra API muy famosa, como es la de Facebook, gracias a esta API podemos crear juegos e integrarlos en la famosa red social o lo un ejemplo más conocido como son los bloques en páginas externas a Facebook que permite comentar con nuestro usuario.

Ahora vamos a hablar de por qué hemos elegido la implementación de una API para acceder a la información y son los siguientes:

- **Rapidez**: Podemos recuperar de una forma muy rápida toda la información que necesitamos para nuestra aplicación, sin necesidad de almacenarla en nuestros dispositivos la información.

- **Localización**: Da lo mismo el lugar en el mundo en el que te encuentres, simplemente teniendo acceso a internet, tenemos acceso a nuestra información almacenada y esto cada vez es más común gracias a las tecnologías 3G, 4G etc.

- **Robustez**: Una API está siendo accediendo en todo momento por usuarios, aunque al principio pueda dar algunos problemas cuando se solucionen los problemas será muy robusta incluso en situaciones de estrés, por ello podemos asegurar que se conseguirá una respuesta adecuada en la inmensa mayoría de los casos.

- **Mantenimiento**: Es mucho más fácil crear un mantenimiento y una evolución aceptable si accedemos a nuestros recursos mediante una API, ya que las mejoras son totalmente transparentes a los programadores que las usan.

4.2.2.1 Desarrollo Guiado por Pruebas

El desarrollo guiado por pruebas (Test Driven Developmen) (Maximilien, 2003), (Venners, 2002) es una práctica de programación donde mediante la creación de test podemos desarrollar un software de calidad siguiendo el flujo de trabajo de esta metodología.

La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido.

4.2.2.1.1Requisitos

Para que esta metodología sea funcional, el sistema tiene que ser lo suficiente flexible como para permitir que sea probado automáticamente. Cada prueba será lo suficiente pequeña como para permitir determinar unívocamente que el código programado es capaz de superar la verificación creada. El diseño se ve favorecido ya que se evita el indeseado “sobre diseño” de las aplicaciones y se logran crear interfaces más claras y un código más cohesivo.

4.2.2.1.2 Ciclo de desarrollo TDD

1. **Elegir un requisito:** Se elige el requisito que se va a desarrollar siendo este fácilmente de implementar y el cual dará un gran conocimiento del problema.

2. **Escribir una prueba:** Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente la especificación y el requisito que va a implementar.

3. **Verificar que la prueba falla:** Si la prueba no falla es porque el requisito ya estaba implementado o por que la prueba es errónea.

4. **Escribir la implementación:** Escribir el mínimo código necesario para que el test funcione.

5. **Ejecutar la prueba automatizada:** En este momento se lanza la batería de test para comprobar que todos los test funcionan correctamente.

6. **Eliminación de la duplicación de código**: En este paso se realizará una refactorización para la eliminación de código duplicado. Una vez se

ha realizado la refactorización se vuelven a lanzar las pruebas para ver que todo funciona correctamente.

7. **Actualización de Requisitos:** Se marca el requisito como implementado.

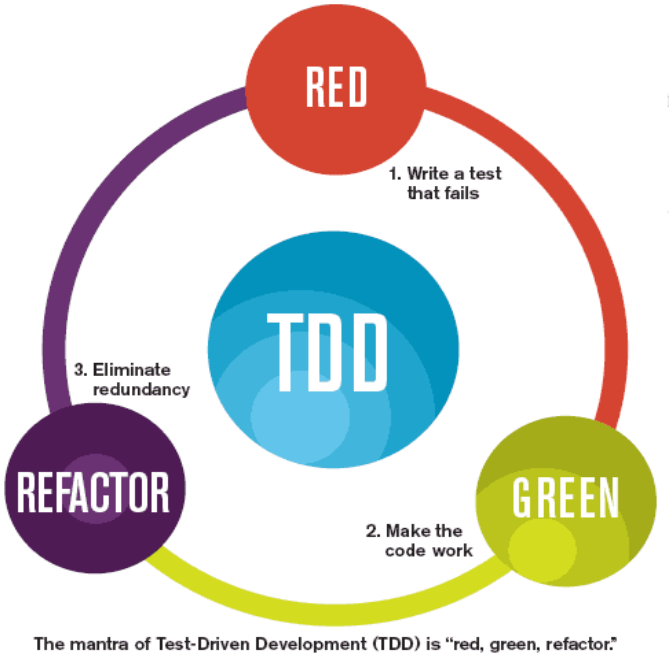


Figura 4.2 Ciclo desarrollo TDD

4.2.2.1.3 Características

Una ventaja de esta forma de programación es el evitar código totalmente innecesario y duplicado. Se intenta escribir el mínimo código posible que sea capaz de conseguir superar la prueba.

Con esto conseguimos que el programador confíe en el código escrito. Esto permite hacer modificaciones profundas de código en una fase de mantenimiento del programa, con la seguridad de que si al lanzar los test funcionan el software desarrollado funcionará.

Otra ventaja es que en primer lugar realizamos que las funcionalidades fallen, esto se hace para asegurarse que los casos de prueba realmente funcionan y si se contemplan todos los posibles errores conseguiremos un software de mucha calidad.

```php
/**
 * @test
 */
public function example()
{
    $this->assertTrue(true);
}
```

Código 4.1 Ejemplo Test

4.2.2.1.4 Reglas FIRST

Las reglas FIRST sobre el código de test son:

- **Fast:** Los test se deben ejecutar rápido y muy a menudo.
- **Indipendent:** Las condiciones de un test no se deben depender de un test anterior.
- **Repeteable:** Se deben poder ejecutar en cualquier entorno.
- **Self-Validating:** El propio test debe ser el encargado de decir si se cumple o no, no debe hacer falta hacer comprobaciones posteriores al test.
- **Timely:** Los test se deben escribir en el momento adecuado que es justo antes de escribir el código de producción, lo que permite escribir fácilmente código testeable.

4.2.2.1.5 ¿Por qué utilizar TDD?

Para responder a esta pregunta tenemos que pensar en la forma que se aplica TDD.

Como se ha visto en la explicación para el correcto uso TDD, en primer lugar se realiza un profundo análisis de los requisitos y de los diversos escenarios evitando la mayor parte de variabilidad y encontrando así los aspectos más importantes. El hecho que únicamente se implemente el código necesario hace que este sea el mínimo necesario, reduciendo la redundancia y los típicos bloques que en poco tiempo se convierten en código muerto.

Hay que remarcar que TDD no sólo se basa en las pruebas, una correcta aplicación de la etapa de refactorización hace que nuestro código sea más legible, óptimo y fácil de mantener.

Como última ventaja importante se puede ver la gran facilidad de la ampliación y mantenimiento del proyecto, ya que únicamente con la ejecución de los test sabemos si todo sigue funcionando correctamente.

4.2.2.2 Diseño de la API

Para conseguir un correcto diseño de la API se han seguido una serie de patrones de diseño expuestos por (Bloch, How to Design a Good API and Why it Matters), ingeniero de Google.

Siguiendo los patrones nombrados anteriormente conseguimos diseñar una API de una gran calidad. Gracias a la correcta utilización de URL y de la generación de estas mediante un orden lógico conseguimos una API muy sencilla de aprender llegando incluso a la posibilidad de utilizar la API sin necesidad de ayuda.

La API está separada por distintos niveles de abstracción (Repositoies, Controlles, etc.), explicados en la parte de implementación se consigue una facilidad terrible a la hora de llevar un mantenimiento o de necesitar una migración de algunas de sus partes a otro lenguaje.

Además de lo explicado anteriormente hay que destacar que dispone de la potencia suficiente para resolver las necesidades de los usuarios de una forma correcta. También destacar la facilidad de la extensión a una nueva versión o incluso de incrementos dentro de la misma.

4.2.3 Diseño Aplicación Android

Para solucionar los problemas propuestos se ha decidido utilizar una filosofía de desarrollo incremental, es decir, se han ido planteando los requisitos de una forma creciente (de más sencillos a más complejos) y poco a poco se han desarrollado cada uno de ellos.

Se ha elegido esta opción puesto que los conocimientos sobre aplicaciones Android eran inexistentes y el poder resolver hitos sencillos hasta conseguir uno más complejo es muy motivante. Por ello se consigue estar ilusionado con el proyecto consiguiendo un gran rendimiento.

Otro punto importante es la facilidad por la cual puedes comprobar que los requisitos son correctos.

4.2.3.1 Desarrollo Incremental

El desarrollo incremental nace ante las carencias del desarrollo tradicional o en cascada. El desarrollo en cascada supone que el desarrollo software se compone de una serie de pasos que finalizan siempre en la entrega del producto final.

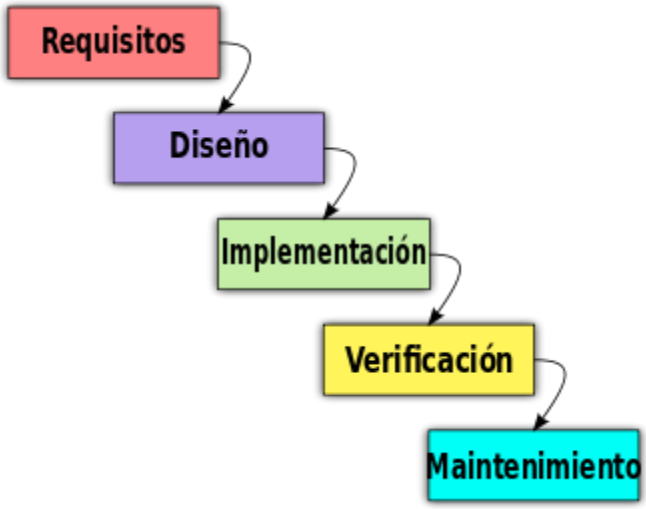


Figura 4.3 Desarrollo en cascada

En cambio, el desarrollo incremental o iterativo propone que el desarrollo del producto final se componga de diversas fases o iteraciones en las que en cada una de ellas se aplique un desarrollo en cascada. De esta forma en cada iteración se obtiene un artefacto software que es una versión del producto final o prototipo.

Así, tras cada iteración, se entrega al cliente una parte del trabajo, viendo éste los progresos. En caso de que detecte algún error en el desarrollo, será capaz de informar de ello al equipo encargado de crear el software, antes de haber llegado al producto final.

Esta corrección puede ser especialmente valiosa en caso de ser un error de análisis, ya que corregir un error de análisis en un producto final puede ser extremadamente costoso o incluso imposible.

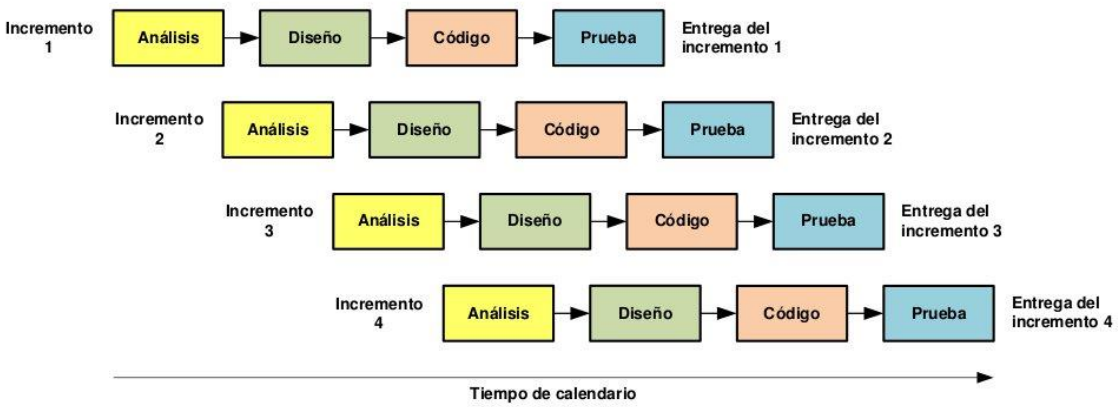


Figura 4.4 Desarrollo Incremental

Mediante el uso de esta metodología de desarrollo se ha demostrado que se aumenta la productividad, se reduce el coste de los proyectos y se obtiene un mayor grado de cumplimiento en los proyectos.

4.2.3.2 Diseño de la Interfaz

Este ha sido uno de los puntos más importantes de todo el proyecto. Al fin y al cabo lo que un usuario percibe de una aplicación es su interfaz, esta no va a hacer ganar usuarios pero sí perderlos, por lo tanto es necesario que la interfaz gráfica de usuario cumpla una serie de características (Shneiderman, 1998).

- Facilidad de compresión, aprendizaje y uso.
- Objeto de interés de fácil identificación.
- Diseño mediante menús, barras de acceso e iconos de fácil acceso.
- Las operaciones se realizan sobre elementos de código visual.
- Las operaciones serán rápidas, tendrán efectos inmediatos.
- Existencia de herramientas de ayuda
- Tratamiento correcto de errores

Además de las características mostradas en la lista anterior se ha de prestar una especial atención a elementos como la tipología y los colores. Tanto la

tipología como los colores elegidos deben cumplir una perfecta armonía tanto en forma como en coherencia interna entre ellos.

Para crear una interfaz enfocada hacia el usuario es necesario tener claros los objetivos, teniendo en cuenta no solo lo que se persigue ofreciendo información, sino las necesidades que van a tener los usuarios a la hora de consultado. También es clave determinar el contenido y la funcionalidad, especificar la estructura organizativa, la navegación y las secciones. Hay que tener en cuenta que cada usuario puede tener diferentes necesidades y un buen sistema de navegación debe cumplir las necesidades de todos.

La interfaz permite al usuario interactuar con los contenidos, no solo se precisa una interfaz atractiva, sino funcional, por que como se ha nombrado anteriormente, gran parte del éxito de un programa proviene de su interfaz. Además mediante la interfaz se facilita y simplifica el acceso y los recorridos a los usuarios.

Se ha tenido en cuenta a la hora de diseñar las interfaces, las aproximaciones ofrecidas por Miriam Gil en su proyecto de doctorado (Gil Pascual, 2013).

Gracias a todas las características aprendidas así como a las aproximaciones vistas se ha creado las interfaces de esta forma, ya que se cumplían todas las características propias para la creación de una buena interfaz gráfica de usuario.

4.3 Implementación

En este tercer apartado vamos a hablar de la implementación de cada uno de los bloques, viendo para cada uno de éstos la parte de código mas importantes.

4.3.1 Implementación de la API

Como bien se ha descrito anteriormente, una vez el sistema ha generado los modelos es necesario almacenarlos en algún lugar para ser consultados por los usuarios si estos lo requieren. Estos se almacenan en una base de datos y debemos acceder a ella para poder recuperar las anteriores adaptaciones que hemos utilizado de un determinado servicio.

Este acceso se realiza mediante una API que haga de capa de abstracción de la base de datos. Esta parte es la que se va a explicar a continuación.

4.3.1.1 Estructura

Para desarrollar la API se ha creado una estructura por capas. En cada una de las capas se implementa un nivel superior de abstracción empezando desde la capa más cercana a la base de datos hasta la capa más lejana de la misma.

Para empezar se ve la base de datos como el núcleo de la estructura y a partir de esta se va a ir explicando cada uno de los niveles superiores.

4.3.1.1.1 Repository

La capa más cercana a la BBDD es la llamada como *Repository.* Es la capa conocida como capa de persistencia o almacenamiento de datos. En esta capa se crean los métodos encargados de acceso a la Base de Datos. Todas las consultas SQL que no estén en esta capa están mal posicionadas y se está acoplando código y por lo tanto perdiendo mucha calidad en el mismo.

Creando esta capa se consigue que si algún día queremos cambiar la tecnología de la base de datos simplemente se deba cambiar el lenguaje de las consultas de acceso y todo funcionará sin ningún tipo de problema.

Dentro de este nivel se dispone de tantos ficheros como tablas. Es muy importante que cada tabla tenga un fichero específico donde estén las consultas relacionadas, única y exclusivamente con la tabla a la que se va a acceder.

Es una buena práctica y por lo tanto muy aconsejable que el nombre de los ficheros estén compuestos el nombre de la tabla que acceden acompañado de un sufijo donde se especifique a que nivel pertenece, como por ejemplo, *AtomicEventRepository, ContextEventRepository, etc.*

En estos ficheros existen unos métodos comunes a todos como pueden ser el método create(), el método update(), o el método delete(). Por ellos para evitar el código duplicado se ha creado una clase abstracta con los métodos anteriormente nombrados llamado BaseRespository, además de estos métodos esta clase contiene el constructor con la conexión necesaria para la base de datos y el nombre de la tabla con la cual se va a trabajar, es decir, esta clase contiene los métodos comunes a todas las clases de acceso de la Base de Datos.

Todas las clases con el sufijo Repository, extienden de esta clase evitando así la necesidad de repetir métodos que son comunes. Si por cualquier razón necesitas modificar uno de los métodos que hemos escrito en BaseRespository solo tenemos que redefinirlo en la clase hija y se utilizará este.

En BaseRespository tenemos como hemos dicho anteriormente el método créate, Código 4.2:

```
public function create(array $data)
{
    return $this->db->insert($this->tableName, $data);
}
```

Código 4.2 Método Create de BaseRepository

En el código anterior se ve que se llama al método insert() de la conexión de la base de datos ($this->db) introduciendo como parámetros el nombre de la tabla y el array de datos con los valores a introducir.

Pero se puede tener un problema y es que se quiera introducir un valor automático, como por ejemplo a qué hora se ha creado la fila para tener un seguimiento. Sería interesante no tener que enviar el valor desde la aplicación hasta la base de datos, si no que se insertara de forma invisible. Para ello lo que se realiza es redefinir el método create() de la siguiente forma, Código 4.3, consiguiendo que la fecha se introduzca de forma automática.

```
public function create(array $data)
{
    $data["changeDateTime"] = date('Y-m-d H:i:s');
    return $this->db->insert($this->tableName, $data);
}
```

Código 4.3 Ejemplo valor por defecto

También se ha de nombrar que todas las clases Repository deben implementar una interfaz para asegurar que los nombres de los métodos son correctos, para todas iguales y que cumplen la funcionalidad.

4.3.1.1.1.1 Interfaz ApiRepositoryInterface

```php
interface ApiRepositoryInterface
{
    public function findAll();

    public function find($id);

    public function create(array $data);

    public function update($id, array $data);

    public function delete($id);
}
```

Código 4.4 ApiRepositoryInterface

Esta es la interfaz a implementar la cual contiene los métodos más comunes para una API. Además de estos hay algunos Repositories donde se implementan métodos particulares de estos, pero no son tan comunes como los vistos en la interfaz como el findAll(), o el delete() entre otros y por lo tanto no se han visto incluidos en la interfaz a implementar.

Si se quisiera aumentar la base de datos con nuevas tablas todos los Repositories deberían implementar esta interfaz creando así una unicidad en todas las clases.

4.3.1.1.2 Controllers

Esta es la capa superior a la capa de persistencia o como hemos visto la capa Repository, esta capa es la encargada de gestionar las acciones y crear la respuesta.

En estos ficheros se controla la lógica de toda la aplicación. Una vez capturada la petición estas clases realizan las llamadas necesarias a otros niveles de abstracción como la de persistencia si necesitamos de ella y creando la respuesta que devolveremos a la petición.

A continuación se va a ver un ejemplo de un método de un Controller para entender cómo se gestionan las peticiones a la API y como se devuelve la respuesta para que pueda ser interpretada:

```php
class PreviousServicesStatesController implements ApiControllerInterface
{
    protected $repo;

    public function __construct(PreviousServicesStatesRepository $repo)
    {
        $this->repo = $repo;
    }

    public function findAll()
    {
        return new JsonResponse($this->repo->findAll());
    }
}
```

Código 4.5 Ejemplo Controller

En el código anterior podemos ver el constructor del Controller donde se introduce como parámetro para su creación el Repository asociado a ese Controller.

Como se ha dicho anteriormente se ha de seguir las buenas prácticas en cuanto a los nombres de los ficheros y cómo podemos observar el Controller sigue una estructura igual a la de los Repositories, nombre de la tabla donde vamos a realizar las acciones más un sufijo donde indica que es un Controller, en nuestro caso PreviousServicesStatesController.

A continuación se va a ver el método findAll(), y cómo podemos observar lo único que hace es llamar al método del Repository findAll() y devuelve la respuesta en un formato JSON.

Aquí se observa la buena separación de capas, en el Controller se podría realizar directamente la consulta pero estarías acoplando muchísimo el código, cosa que no se debe hacer ya que es una muestra de un software de poca calidad.

Se puede observar que todos los Controllers implementan una interfaz llamada ApiControllerInterface, donde definimos los métodos que todos los controladores deben implementar para como hemos dicho en el apartado de Repositories crear una unicidad, para si algún día es necesario ampliar el proyecto, todos los Repositories simplemente deberán implementar esta interfaz Código 4.4

4.3.1.1.2.1 Interfaz ApiControllerInterface

```php
interface ApiControllerInterface
{
    public function findAll();

    public function findById($id);

    public function create(Request $request);

    public function update($id, Request $request);

    public function delete($id);
}
```

Código 4.6 ApiControllerInterface

Como en la Interfaz de los Repositories se implementan los métodos más comunes para una API. Todos los Controllers que se implementaran deberían hacer que implementaran esta interfaz consiguiendo así una aplicación más fácil de mantener y muy fácil de escalar.

4.3.1.1.3Acceso API

Una parte muy importante es la de acceso a la API, como se ha visto a esta le llega una petición con una URL. Esta URL debe ser examinada, ejecutada y devolver el servicio que estén pidiendo. Para ello se utiliza el sistema de Routing de sílex.

Estas rutas están separadas por dos partes, que son las siguientes:

- **Patrón**: Toda ruta debe cumplir un patrón para poder transformar esta por un recurso.

- **Método**: Uno de los métodos HTTP: GET, POST, PUT, DELETE.

Una vez atacamos la API se debe conseguir las dos partes anteriores para poder saber que controlador va a ejecutar la petición. Para ello se utiliza la factoría de controladores que proporciona Silex. Para poder trabajar con esta se crea una variable donde se almacena la variable, en nuestro caso $api.

```php
$api = $app['controllers_factory'];
```

Código 4.7 Almacenar Factoría

Para poder capturar rutas se debe utilizar la factoría, para ello se le indica mediante una serie de métodos el patrón que debe coincidir la ruta y que

controlador se envía la petición. Como se ha visto la petición puede enviarse por varios métodos HTTP, por ello se utilizará el método que queramos capturar.

```
$api->get('/users', "user.controller:findAll");
```

Código 4.8 Capturar ruta

Como vemos estamos indicando que cuando la petición se realice mediante GET y la ruta cumpla el patrón “/users”, la petición sea ejecutada por el método findAll() del controlador definido como “user.controller”

En ocasiones se deben enviar variables en las ruta, como por ejemplo cuando se quiere acceder a un recurso por id. Si este es nuestro caso se debe indicar mediante corchetes que en esa posición se va a recibir una variable.

```
$api->get('/users/{id}', "user.controller:findById");
```

Código 4.9 Ejemplo Ruta Con Variable

Se debe prestar principal atención que la búsqueda de la ruta se realiza de forma ordenada y será ejecutada por la primera coincidencia del patrón. Si las rutas se crean siguiendo una estructura lógica no aparecerán estos problemas.

Para acabar con el apartado de URL cada decir que toda la API se debe acceder mediante la ruta “/api”, para crear un prefijo común para todas las rutas debemos definirlo Código 4.10:

```
$app->mount('/api', include "api.php");
```

Código 4.10 Prefijo /api para las rutas

En esta línea se puede ver como indicamos a la aplicación que la ruta /api se busca en el fichero api.php. En este fichero se encuentran todos los patrones que pueden coincidir con la ruta de la petición.

Si quisiéramos crear una versión nueva de la API o cambiar el prefijo /api únicamente se modificaría código en un lugar.

```
$app->mount('/api/v2', include "api.php");
```

Código 4.11 Modificación del prefijo

```php
$app->mount('/apiv2', include "apiv2.php");
```

Código 4.12 Nueva versión de la API

4.3.1.1.4 Model

En esta carpeta se crean los modelos donde se almacenan variables que van a ser utilizadas por toda la aplicación. Es muy importante no utilizar magic numbers a la hora de realizar un buen código. Para ello se han creado unas clases de ayuda donde se almacenan los magic number con una serie de constantes utilizando estas siempre que se necesite utilizar esos valores.

Como ejemplo se ha creado una clase con los códigos http más importantes, así siempre que se quiera devolver un código 200 se accede a la constante que indica esta variable, así se evita que si algún día estos códigos cambian solo haya que modificar la clase donde se definen las constantes y no a lo largo de toda la aplicación Código 4.13.

```php
class HttpCodes
{
    /**
     * Standard response for successful HTTP requests.
     * The actual response will depend on the request method used.
     * In a GET request, the response will contain
     * an entity corresponding to the requested resource.
     * In a POST request the response will contain
     * an entity describing or containing the result of the action
     */
    const OK = '200';

    /**
     * The request has been fulfilled and
     * resulted in a new resource being created
     */
    const CREATED = '201';
}
```

Código 4.13 Códigos HTTP

Se han creado dos clases en Model, como son la clase HttpCodes y la clase TableNames, como bien indican sus nombres, en la primera se almacenan los códigos HTTP y en la segunda el nombre de las tablas de la aplicación.

4.3.1.1.5 Configuración de la API

Toda la configuración de la API (Moreno, 2012) se realiza en el fichero app.php de la aplicación. En este fichero se encuentran el registro de los servicios, de

los Providers, en general de todo lo necesario para que la aplicación funcione, tanto en entorno de desarrollo como en entorno de producción.

Para el correcto funcionamiento se declaran todos los Controllers y todos los Repositories de la aplicación. Para registrar un Provider se hace mediante el método register() de la aplicación pasando como primer parámetro la instancia del Provider y como segundo un array de pares clave valor que sobrescribirá parámetros anteriormente declarados por el propio servicio.

```
$app['notifications.user.repository'] = $app->share(function() use ($app){
    return new UserRepository($app['db'], TN::USERS);
});
```

Código 4.14 Declaración Repository

```
$app['user.controller'] = $app->share(function() use ($app) {
    return new UserController($app['notifications.user.repository']);
});
```

Código 4.15 Declaración Controller

```
$app->register(new TwigServiceProvider(), array(
    'twig.path'    => array(__DIR__.'/../templates'),
    'twig.options' => array('cache' => __DIR__.'/../cache/twig'),
));
```

Código 4.16 Registrar Provider

4.3.2 Implementación Aplicación Android

En este apartado se va a explicar toda la creación de la aplicación Android su estructura, las partes más importantes de la creación y los problemas que han ido surgiendo durante el desarrollo.

Además se va a explicar la metodología utilizada (desarrollo incremental) explicando en que consiste y por qué se ha decidido utilizar esta metodología.

4.3.2.1 Listados de servicios

Para comenzar se va a explicar la creación de la lista general de servicios. Para crear la lista general se utiliza un ListView (Vogel, 2013) . Si se fija la

atención en la Figura 4.5 se observa que un listado no es más que unir tantas veces como se desee una entrada a la lista.

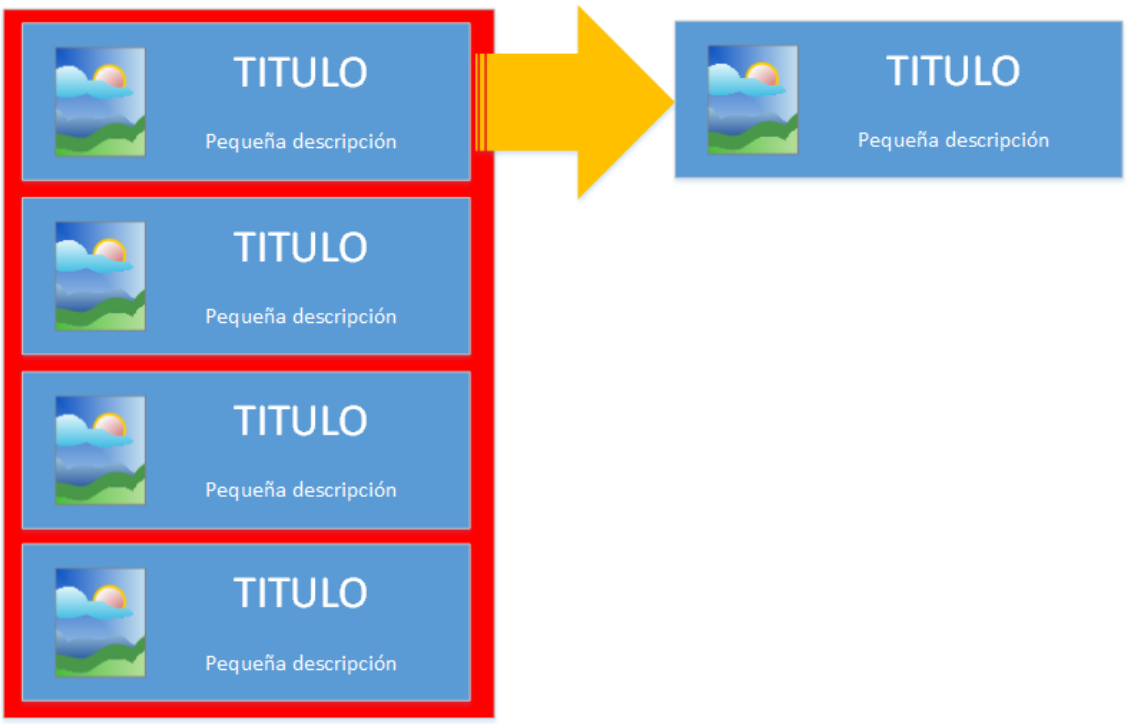


Figura 4.5 Estructura de los Listados

Para el listado se deben crear dos layouts:

- En el primer layout se crea la estructura que va a contener cada una de las entradas que contendrá el ListView.
- En el segundo layout contendrá los huecos para introducir las entradas en la lista.

Además de la estructura del listado también cada una de las entradas está formada por una serie de componentes. Hay que tener decidido cómo se va a mostrar la información ya que este layout es el que van a utilizar todas las entradas.

En el desarrollo de la tesina se ha decidido utilizar una estructura como la mostrada en la Figura 4.6, en la cual se muestra una imagen del servicio, y el nombre:

Figura 4.6 Estructura de una entrada

Como se explicó en el tema de Funcionalidad, se disponen de dos listados diferentes, el primero es la lista de los servicios generales y el segundo es para el estado de los servicios. Por ello se va a explicar la creación de los estados de los servicios más en profundidad y a continuación las diferencias con la creación del listado general.

4.3.2.2 Layouts

Como bien se ha nombrado anteriormente se dispone de dos layouts, uno para cada una de las entradas y otro el contenedor que contendrá todas las entradas. Como bien conocemos los layouts se representan mediante ficheros xml que vamos a ver a continuación.

En primer lugar se observa el layout para las entradas donde se verá la relación directa que existe con la Figura 4.6.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/imageView_imagen"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:adjustViewBounds="true"
        android:scaleType="fitXY"
        android:layout_margin="10dp"
        android:src="@android:drawable/ic_menu_gallery" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView_superior"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Large Text"
      android:textAppearance="?android:attr/textAppearanceLarge" />

        <TextView
            android:id="@+id/textView_inferior"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Small Text"
            android:textAppearance="?android:attr/textAppearanceSmall" />

    </LinearLayout>
</LinearLayout>
```

Código 4.17 Layout Entrada

Si se visualiza el Código 4.17 todos los elementos están nombrados con un identificador, esto será muy importante ya que luego serán utilizados para poder utilizar cada uno de los elementos de layout.

Por otro lado se ver el Código 4.18 que contiene las entradas creadas:

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/ListView_listado"
 android:layout_width="match_parent"
 android:layout_height="match_parent" >

 </ListView>
```

Código 4.18 Layout Listado

En el Código 4.18 se observa que está formado únicamente con un ListView que será el que se va a rellenar de forma dinámica con los datos de los servicios.

4.3.2.3 Lógica

En este apartado se explica cómo cargar los datos y rellenar tanto las entradas como los listados. Para ello se dispone de una Activity que será la encargada de mostrar el listado. Esta Activity hará uso de otras dos, en la primera encapsularemos los datos de las entradas y en la otra que será la encargada de adaptar todas las entradas a la lista. La Activity encargada de encapsular los datos se va a llamar “Lista_Entrada.java” y la que encapsula los datos “Lista_adaptador.java”.

Se va a definir en primer lugar la clase “Lista_Entrada.java”, esta clase es la encargada de crear unos paquetes de datos para cada uno de los conjuntos de las entradas. Para separar los datos de los servicios se utiliza un Handler (manejador).

Para cada servicio se dispone de una estructura como la siguiente:

- **Título:** nombre del servicio o de la propiedad del servicio que vamos a trabajar.
- **Descripción:** estado de la propiedad del servicio que mostramos en los historiales.
- **Imagen:** imagen que define cada propiedad o cada servicio.

La utilización del Handler es muy sencilla, únicamente se le debe enviar los tres parámetros que se necesitan, la imagen, los dos textos y el manejador se encarga de manejarlo como vemos a continuación.

Como primer paso se crea cada una de las entradas, que se implementa como se muestra en el fragmento Código 4.19:

```
new Lista_entrada(Imagen, “Título”, “Descripción”);
```

Código 4.19 Creación Lista_Entrada

En este momento es cuando entra en acción el Handler organizando la vista de la siguiente manera Código 4.20**:**

```
public class Lista_entrada {

    private int idImagen;
    private String textoEncima;
    private String textoDebajo;

    public Lista_entrada (int idImagen, String textoEncima, String textoDebajo) {
        this.idImagen = idImagen;
        this.textoEncima = textoEncima;
        this.textoDebajo = textoDebajo;
    }

    public Lista_entrada (int idImagen, String textoEncima) {
        this.idImagen = idImagen;
        this.textoEncima = textoEncima;
    }

    public Lista_entrada (String textoEncima) {
        this.textoEncima = textoEncima;
    }

    public String get_textoEncima() {
        return textoEncima;
    }

    public String get_textoDebajo() {
        return textoDebajo;
    }

    public int get_idImagen() {
        return idImagen;
    }
}
```

Código 4.20 Handler Lista Entrada

Si se observa la clase la única función es la de separar unos datos de otros de una forma correcta.

Una vez se ha creado una entrada hay que almacenarla ya que se necesita una colección, por ello se va a utilizar la estructura de datos ArrayList:

```
ArrayList<Lista_entrada> datos = new ArrayList<Lista_entrada>();
```

Código 4.21 Colección de entradas

Para rellenar la lista se debe utilizar el método add() de ArrayList. Se dispone de un bucle para rellenar todos los datos:

```
JSONObject help;
for (int i = 0; i < arrayServices.length(); i++) {
    try {
        help = arrayServices.getJSONObject(i);
        servicesArray.add(help.getString("service_id"));
        datos.add(new Lista_entrada(
            getIdImage(help.getString("icon")),
            help.getString("nTitle"))
        );

    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Código 4.22 Insertar datos en Lista_entrada

Para continuar se va a ver la clase “Lista_adaptador.java” y como se utiliza, para así crear las listas de forma dinámicas. Esta clase crea un adaptador para ListView y utilizando el ArrayList que se ha creado anteriormente y tras la sobre escritura del método onEntrada() donde se conecta cada una de las partes de “Lista_entrada.java” con las partes de “Lista_Adaptador.java”.

Una vez se realiza la creación de las Views se relacionan con cada entrada y eso se hace como mostramos en Código 4.23**:**

```
lista = (ListView) findViewById(R.id.ListView_listado);
lista.setAdapter(new Lista_adaptador(this, R.layout.entrada, datos){
    @Override
    public void onEntrada(Object entrada, View view) {
        if (entrada != null) {
            TextView texto_superior_entrada =
            (TextView) view.findViewById(R.id.textView_superior);
            if (texto_superior_entrada != null)
                texto_superior_entrada.setText(
                    ((Lista_entrada) entrada).get_textoEncima()
                    );

            ImageView imagen_entrada =
            (ImageView) view.findViewById(R.id.imageView_imagen);
            if (imagen_entrada != null)
                imagen_entrada.setImageResource(
                    ((Lista_entrada) entrada).get_idImagen()
                );
        }
    }
});
```

Código 4.23 Relacionar datos

4.3.2.4 Evento onClick

Para acabar con el ListView se necesita capturar el evento onClick, con Android esto se realiza de una forma muy sencilla gracias a los Listener. Una vez se ha capturado el evento que devuelve la posición en la lista se puede realizar todo lo que se necesita con el servicio que se ha seleccionado, en nuestro caso todos los cálculos para mostrar el historial.

4.3.2.5 Datos de las entradas

En el apartado anterior hemos visto como creamos los ListView que se adaptan a la forma de mostrar los datos. Pero no se va a profundizar en cómo se consiguen los datos reales de la aplicación. En este apartado estudiaremos como se accede a la base de datos para conseguir datos de los usuarios de la misma.

En primer lugar vamos desarrollar cómo se consigue la lista de los servicios. Esta lista se crea de una forma dinámica buscando los servicios almacenados.

Para acceder a los datos se utilizan servicios, que explicaremos en el apartado correspondiente y después utilizamos los datos que son de utilidad.

Se dispone de una clase llamada APICalls, esta clase es la que sirve de almacenamiento de todos los métodos que se han utilizado para consultar los datos. Por ello tras crear las variables que se necesitan se realiza la llamada al método getAllServices(), este método es el encargado de devolver un JSONArray con la información de todos los servicios. Para mostrar la lista con todos estos lo único que se hace es recorrer el objeto y obtener de este el nombre, y el identificador de la imagen. Con estos datos y el handler encargado de dibujar el ListView se crea la lista de servicios disponibles.

Para que la aplicación se comporte de una forma rápida haga que el usuario tenga los mínimos tiempo de espera se ha tomado la decisión de cargar todos los datos en objetos JSON, como ya sabemos el acceso a variables locales siempre será más rápida y efectiva que a servicios externos.

4.3.2.6 Historiales de un Servicio

Como segundo punto importante se puede ver la creación de los historiales. Una vez se consigue el identificador del servicio seleccionado se necesita acceder a los datos para sacar toda la información disponible. Para ello se divide en dos partes la creación de los historiales; en la primera se creará el estado actual y en la segunda todos los historiales.

Para saber en qué posición nos encontramos disponemos de un paginador, este devuelve la posición actual y de esa manera se sabe que vista vamos a crear.

Si la posición actual es la 0, se sabe que vamos a visualizar las opciones actuales por ello no se necesitan más parámetros, pero si la posición es superior a la 0 se debe saber a qué posición de historiales del JSONArray vamos a acceder por ello se pasa como parámetro el número de la posición que necesitamos.

Para la creación del JSONArray de almacenamiento de estados lo que se hace es rellenarlo de forma progresiva, en el momento vamos a accediendo a la Base de Datos para conseguir la información vamos creando el JSON, de esta forma siempre que se necesiten los datos para las vistas se acceden a buscarlos en primer lugar en el JSONArray y si estos no están ya se accede al servidor. Utilizando esta técnica se consigue acceder una vez al servidor para conseguir los datos, las siguientes, la información estará almacenada y el tiempo de acceso a la información será despreciable.

Como final a la creación de los historiales se hablará del botón para restaurar las preferencias. Este botón se crea de forma dinámica para cada historial excepto para las preferencias actuales que no disponen de este botón.

Para la creación del botón se ha decidido utilizar la estrategia del borrado en caso de no uso, se crea el botón para todas las vistas del historial y si el paginador indica que la vista a mostrar no cumple las condiciones necesarias es eliminado del layout.

4.3.2.7 Restaurar adaptaciones

Este es el objetivo de la aplicación, el poder restaurar las preferencias del servicio seleccionado. Una vez se conoce el historial que se quiere restaurar se realizan las acciones necesarias para almacenar los datos. En primer lugar se copian los datos de las preferencias actuales y copiarlas en la tabla de los historiales, en segundo lugar se deben eliminar las preferencias actuales que se acaban de copiar. En tercer lugar se tiene que cambiar el historial de las preferencias que se han seleccionad por el estado actual y como último paso se recarga la vista de la aplicación para que sea visible el nuevo historial desde el momento de la creación.

4.3.2.8 Clases de ayuda

Como se observa todas las preferencias funcionan con constantes, y estas constantes tienen un valor asociado. De esta forma cuando accedemos a los datos las preferencias devuelven un valor número y nosotros debemos transformarlo al texto asociado a ese valor.

Con esta finalidad se ha creado una serie de clases de ayuda que con método estático cambia el valor de la constante que se le introduce por el texto que tiene asociado esa constante.

Para esta transformación utilizamos un Map, esta estructura de datos está formada por un par clave valor, en la clave se almacena la constante y en el valor, el valor relacionado.

```
private static final Map<String, String> device;
device = new HashMap<String, String>();
device.put("1", "iOS");
device.put("2", "Android");
return device.get(value);
```

Código 4.24 Clase ServiceStateSelected

Las clases de ayuda se utilizan también para sacar la imagen relacionada según el estado del servicio.

```
private static final Map<String, Integer> device;
device = new HashMap<String, Integer>();
device.put("1", R.drawable.ios);
device.put("2", R.drawable.android);
device.get(value);
```

Código 4.25 Utilización Clase ImageSelected

Como se observa en Código 4.24 y en Código 4.25 se ve la utilización de las clases de ayuda, en la primera se devuelve un texto con el estado y en la segunda el id de la imagen relacionada con el servicio para mostrarlo.

4.3.2.9 Clases Singleton

Una vez tenemos un servicio seleccionado se debe tener acceso a él en todo momento, por ello se utiliza el patrón singleton. Esta técnica se ha utilizado siempre que se necesita tener una única instancia de una clase.

4.3.2.9.1 Implementación patrón Singleton

A continuación vamos a explicar cómo se ha creado el patrón singleton mediante código. Como se ha dicho durante el desarrollo de la memoria, el proyecto es para dispositivos Android, por ello está desarrollado en Java. Pero este patrón se puede implementar en muchísimos lenguajes de programación.

```
public class ServiceSelected {

    private static ServiceSelected INSTANCE = null;
    private String service = "";

    public String getService()
    {
        return service;
    }

    public void setService(String serv)
    {
        service = serv;
    }

    public static ServiceSelected getInstance() {
        if(INSTANCE == null)
            createInstance();
        return INSTANCE;
    }

    private synchronized static void createInstance() {
        if(INSTANCE == null)
            INSTANCE = new ServiceSelected();
    }
}
```

Código 4.26 Utilización Patrón Singleton

Esta es la clase Service Selected, es la encargada de guardar el servicio elegido por el usuario para posteriormente poder mostrar toda la información que se dispone del servicio almacenado.

Si vemos el código la clase contiene el método getInstance() que es el encargado de comprobar si existe una instancia creada de la clase, si la instancia existe devuelve instancia, en caso contrario llama al método createInstance(), que es el encargado de crear la instancia de la clase.

Por ello para trabajar con la instancia debemos hacerlo como se muestra en Código 4.27:

```
ServiceSelected ss = ServiceSelected.getInstance();
```

Código 4.27 Recuperar Instancia

Utilizando la variable ss se dispone de acceso a todos los métodos públicos de Service Selected, utilizando getService() cuando sea necesario saber qué servicio esta seleccionado y setService() cuando sea necesario setear la variable privada service.

Este patrón se ha utilizado en varias ocasiones puesto que es la forma conveniente de trabajar con datos cuando son necesitados en varios lugares ya que aseguran la integridad de la información siempre y cuando el patrón este correctamente construido y se haga una buena utilización del mismo.

4.4 Pruebas

Por último, se va a explicar la forma en que ha sido testeada cada una de las partes de la aplicación. En este apartado se va a seguir una estructura similar a los anteriores, explicando en primer el testo de la API y a continuación las pruebas para la parte de Android.

4.4.1 Pruebas en la API

Esta es una de las implementaciones más importantes de toda la aplicación ya que permite saber no solo que los métodos implementados funcionan, si no que funcionan bien.

Todos las peticiones que la API es capaz de resolver están verificadas y simplemente lanzando los test, se comprueba si las modificaciones que se han realizado son compatibles con todo lo que llevamos o por el contrario las modificaciones estropean alguna funcionalidad anterior.

Como en los apartados anteriores se dispone de un fichero de test por cada tabla en la base de datos en el cual se comprueban los métodos implementados en el Controller relacionado con esa tabla.

En este apartado también se ha seguido el convenio explicado anteriormente de los nombres de los ficheros, creando estos con el nombre de la tabla acompañados por el sufijo Test.

```
class AtomicEventTest extends WebTestCase
{
```

Código 4.28 Ejemplo Clase de Test

Todas las clases de Test deben extender de la clase WebTestCase, esta es una clase de ayuda donde se proporcionan los métodos más comunes evitándonos así las tareas más costosas y repetitivas pudiendo desarrollar más

rápidamente lo que realmente importa, la verificación de la funcionalidad de nuestra API.

Esta clase extiende de PHPUnit_Framework_TestCase que esta es la encargada de permitir que los test puedan ser ejecutados.

4.4.1.1 ¿Cómo es un Test?

En este momento se va a explicar exactamente que es un test. Un test es un método encargado de comprobar que una funcionalidad del código que va a comprobar es correcta. Cada test debe contener un Assert, que es el método encargado de la validación.

Cada uno de los test está formado por varias partes, que son las siguientes:

- **Título del test**: El nombre del test debe ser lo suficientemente descriptivo para decirte que ha ido mal si este falla.
- **Arrange**: Se crean los datos que vayamos a necesitar en el test.
- **Act**: Creación de las llamadas necesarias a los métodos que se van a comprobar.
- **Expect**: Todos los asserts que se vayan a comprobar que funcionan.

```php
/**
 * @test
 */
public function getAllAtomicEvents()
{
    //Act
    $this->client->request('GET', '/api/atomicEvents');

    //Assert
    $clientResponse = $this->client->getResponse();
    $statusCode = $clientResponse->getStatusCode();
    $this->assertEquals(Http::OK, $statusCode);

    $this->assertEquals(
        self::NUMBER_ATOMIC_EVENTS,
        $this->repository->getNumberAtomicEvents()
    );
}
```

Código 4.29 Ejemplo test con anotación

También se puede ver la anotación @test, esta es la encargada de indicar que el método se debe ejecutar cuando la batería de test se ejecute. Esto se puede indicar mediante la anotación @test o empezar a nombrar el método con la palabra test. Como se observa en Código 4.30.

```php
public function testInsertAtomicEvent()
{
    //Arrange
    $dataArray = array(
        "aeKey"               => 'aa',
        "aeNewValue"          => 'aaNv',
        "event_id"            => 1,
        "considerateSystemID" => 'cSID',
        );

    //Act
    $this->client->request('POST', '/api/atomicEvents', $dataArray);

    //Assert
    $statusCode = $this->client->getResponse()->getStatusCode();
    $this->assertEquals(
        Http::CREATED,
        $statusCode
    );

    $this->assertEquals(
        4,
        $this->repository->getNumberAtomicEvents()
    );
}
```

Código 4.30 Ejemplo test sin anotación

Se ha elegido indicar que es un test mediante la anotación @test ya que queda un código más limpio y permite introducir más funcionalidades dentro del comentario como los dataProviders.

4.4.1.1.1Data Provider

Un dataProvider es una función que permite definir un conjunto de casos para un mismo test. Son perfectos para hacer pruebas unitarias de valores límite.

Para la utilización de estos se debe tener en cuenta una serie de consideraciones.

Estas consideraciones se pueden ver a continuación:

- Untest se asocia con un DataProvider mediante la anotación @dataProvider.

- El método provider debe ser público y estático.

- El método provider debe devolver un aray de arrays o un iterator sobre arrays; cada subarray es un caso de test y debe contener tantos elementos como parámetros tenga el test.

En Código 4.31 se ve un ejemplo de un dataProvider y como se utiliza:

```
public static function getAtomicEventIds()
{
    $ids = array();

    for ($i = 1; $i <= self::NUMBER_ATOMIC_EVENTS; $i++) {
        $ids[] = $i;
    }

    return array($ids);
}
```

Código 4.31 Data Providers

Como vemos se crea un array de ids, que se los serán enviados a las funciones que los necesiten evitando así la creación de un bucle dentro de cada función, consiguiendo así un código más limpio y claro.

Para acabar con los dataProviders vamos a ver un método que utiliza el dataProvider mostrado en Código 4.32 :

```
/**
 * @test
 * @dataProvider getAtomicEventIds
 */
public function deleteAtomicEventById($id)
{
    $numberAtomicEvents = self::NUMBER_ATOMIC_EVENTS;
    $this->assertEquals(
        $numberAtomicEvents,
        $this->repository->getNumberAtomicEvents()
        );

    //Act
    $this->client->request('DELETE', '/api/atomicEvents/'.$id);
    //Assert
    $statusCode = $this->client->getResponse()->getStatusCode();
    $this->assertEquals(Http::OK, $statusCode);

    $this->assertEquals(
        --$numberAtomicEvents,
        $this->repository->getNumberAtomicEvents()
        );
}
```

Código 4.32 Utilización dataProvider

Como vemos en las anotaciones superiores, el método utilizará un dataProvider llamado getAtomicEventIds() que es el encargado de ir introduciéndole como parámetro del método el id del objeto AtomicEvent que se va a trabajar, en el ejemplo mostrado para borrarlo de la base de datos.

4.4.1.2 Como es una clase de Test

Para crear una clase de Test es aconsejable extender de WebTestCase puesto que proporciona una serie de funcionalidades muy importantes evitando así mucha faena, aunque no es obligatorio extender de esta clase. Para que los test funcionen hay que extender de la clase PHPUnit_Framework_TestCase así que este será el primer paso para crear la clase de Test.

Todas las clases que heredan de WebTestCase deben implementar un método llamado createAplication(), en este método se carga toda la configuración necesaria para trabajar en los test y se devuelve la aplicación. Código 4.33 muestra nuestro método createApplication():

```php
public function createApplication()
{
    $app =  require __DIR__.'/../../../../app.php';
    require __DIR__.'/../../../../controllers.php';

    $app['debug'] = true;
    $app['exception_handler']->disable();
    $app['dbs.default'] = 'mysql_test';

    $this->eventRepository = $app['notifications.event.repository'];
    $this->repository = $app['notifications.atomicEvent.repository'];

    return $app;
}
```

Código 4.33 Método createApplication()

Por defecto, dentro de un test la aplicación se comporta exactamente igual que si la estuviera accediendo un usuario a través de su navegador. No obstante, cuando se produce un error es más útil que el test acceda directamente a la excepción PHP en vez de la página HTML del error. Para ello, modificamos la configuración de la aplicación con las líneas:

```php
$app['debug'] = true;
$app['exception_handler']->disable();
```

Código 4.34 Configuración Test

También se observan dos líneas más, que son las de la creación de las variables locales repositories, con esto se consigue que las clases sólo se carguen una vez y no por cada vez que las llamemos, evitando así un sobre coste temporal y de recursos muy amplio.

```
$this->eventRepository = $app['notifications.event.repository'];
$this->repository = $app['notifications.atomicEvent.repository'];
```

Código 4.35 Inyección de dependencias

Por último se inserta la línea que indica que base de datos vamos a utilizar, esto se explicara en el capítulo de la creación de la base de datos para test así que no vamos a darle mayor importancia en este apartado.

4.4.1.3 Configuración de los Test

Para que la aplicación sea capaz de ejecutar la batería de test se debe configurar PHPUnit. Para ello la opción recomendable es crear un fichero phpunit.xml.dist, un directorio llamado test/ y como hemos dicho anteriormente almacenar cada test en el archivo, a continuación vamos a ver la forma recomendada y como se ha seguido este estándar para crear nuestras clases de Test:

```
test/<AplicationName>/Test/<ClassTestName>Test.php
test/Notifications/Test/AtomicEventTest.php
```

Código 4.36 Estructura directorio Test

Como se ha nombrado anteriormente se debe crear un fichero phpunit.xml.dist, este fichero es el encargado de configurar toda la configuración necesaria para PHPUnit, el framework encargado de ejecutar los test. Para explicar el fichero se va a dividir en dos partes.

En la primera parte del fichero se indica que parámetros de configuración queremos que PHPUnit utilice y cuáles no. Esto vamos se va a ver mejor en el primer fragmento de nuestro fichero de configuración:

```
<phpunit
    backupGlobals="false"
    backupStaticAttributes="false"
    colors="true"
    convertErrorsToExceptions="true"
    convertNoticesToExceptions="true"
    convertWarningsToExceptions="true"
    processIsolation="false"
    stopOnFailure="false"
    syntaxCheck="false"
    bootstrap="./src/Notifications/tests/bootstrap.php">
```

Código 4.37 Fichero phpunit.xml.dist (1)

En Código 4.37 se observan todos los parámetros de configuración que phpunit ofrece, como son, colors, convertErrorsToExceptions, precessIsolation, etc.

También debemos fijarnos en la línea de la opción bootstrap, ya que este fichero es el encargado de cargar todo lo necesario de la aplicación para que los test dispongan de sus dependencias funcionales. En este fichero lo único que se hace es cargar en la variable loader el fichero donde se almacenan todas las librerías.

```
$loader = require __DIR__.'/../../../vendor/autoload.php';
```

Código 4.38 Fichero bootstrap

Para continuar vamos a ver la segunda parte del fichero, en esta segunda parte se definen los testsuites, o lo que es lo mismo los directorios donde se almacenan los test para ser ejecutados:

```
<testsuites>
    <testsuite name="Notifications Test Suite">
        <directory>./src/Notifications/tests/</directory>
    </testsuite>
</testsuites>
```

Código 4.39 Fichero phpunit.xml.dist (2)

Si quisiéramos comprobar otros directorios únicamente se debería incluir este directorio con una línea más dentro del fichero xml indicando la ruta de este, sería algo similar al ejemplo mostrado en Código 4.39:

```
<testsuites>
    <testsuite name="Notifications Test Suite">
        <directory>./src/Notifications/tests/</directory>
        <directory>./src/OtherDirectory/tests/</directory>
    </testsuite>
</testsuites>
```

Código 4.40 Insertar directorios test

De esta forma se ejecutarían los test de los dos directorios incluidos en testsuite.

Para acabar se va a explicar que no es necesaria la creación de este fichero pero sí que es muy aconsejable. Los test podríamos lanzarlos desde la línea de comandos con la instrucción phpunit acompañada de las opciones y la carpeta donde queramos comprobar los test. Esto no es aconsejable puesto que se puede crear una línea de instrucción muy larga y debemos cambiarla cada vez que queremos comprobar una carpeta.

4.4.1.4 Configuración Test y Producción

Como se puede no se puede atacar la base de datos de producción para ejecutar los test, ya que para estos se realizan operaciones de inserción de borrado, o de actualización. Por ello se han creado dos entornos de ejecución, uno para cuando las peticiones atacan a la base de datos real, de producción, y otra para cuando las peticiones atacan a la base de datos de pruebas o test, cuando se lanzan los test.

Para ello se deben crear dos bases de datos completamente iguales, la única excepción es que la de producción tiene datos reales y la base de datos de test está completamente vacía, se van introduciendo y eliminando datos según la aplicación lo requiera.

Para ello a la hora de cargar el Provider de Doctrine se indica que tenemos dos bases de datos, con los parámetros específicos de cada una, para ello se creará la definición mostrada en Código 4.41:

```php
$app->register(new Silex\Provider\DoctrineServiceProvider(), array(
    'dbs.options' => array (
        'mysql_prod' => array(
            'driver'   => 'pdo_mysql',
            'dbname'   => 'notificationsV2',
            'user'     => 'root',
            'password' => 'root',
            'port'     => 3306,
            'path'     => __DIR__.'/app.db',
        ),
        'mysql_test' => array(
            'driver'   => 'pdo_mysql',
            'dbname'   => 'notificationsV2Test',
            'user'     => 'root',
            'password' => 'root',
            'port'     => 3306,
            'path'     => __DIR__.'/app.db',
        ),
    ),
));
```

Código 4.41 Registrar configuración para doble base de datos

Como vemos en el Código 4.41 se definen las dos bases de datos. La primera dispone del identificador ‘mysql_prod’ que indica que es la base de datos de producción ya que se deduce por el sufijo del nombre que es la base de datos que se atacará en producción. Por otro lado vemos la segunda, está identificada como ‘mysql_test’ y de la misma forma que en la anterior se deduce por la terminación del nombre que es la encargada de utilizarse en los test.

4.4.1.4.1Utilizar configuración Test o Producción

Como es de imaginar es muy incómodo que se tenga que indicar cada vez que tipo de configuración debe adoptar nuestra aplicación ya que un simple fallo puede hacer que todos los test se ejecuten sobre la base de datos de producción, perdiendo todos los datos almacenados en esta. Por ello se debe crear algún tipo de mecanismo que automáticamente identifique con que base de datos sea de trabajar.

Para ello tenemos que conocer que cuando damos de alta las bases de datos con las cuales vamos a trabajar por defecto se utilizan la que está en primera posición, por lo tanto si no indicamos nada a la aplicación se accederá a la base de datos de producción.

Por la razón anterior únicamente se debe indicar que vamos a utilizar a base de datos de test cuando los test sean lanzados. Para hacer saber a la aplicación que se va a atacar a la base de datos creada para los test, hay que indicarlo,

esto se realiza en el método createApplication() de los test con la siguiente instrucción:

```
$app['dbs.default'] = 'mysql_test';
```

Código 4.42 Indicar base de datos Test

Con el código mostrado en Código 4.42 le se avisa a la aplicación que la base de datos por defecto es la identificada en la clave 'mysql_test' y por lo tanto se pueden realizan las operaciones básicas de inserción, actualización o de borrado sobre la base de datos sin miedo a perder datos importantes que puedan perder integridad en la información que hay almacenada en la base de datos de producción.

4.4.1.5 Interpretación de los resultados de los test

Para acabar con la parte dedicada a los test se va a explicar una de las partes más importantes, como interpretar los resultados.

Hay que comentar que según los parámetros introducidos en el fichero phpunit.xml.dist la información se mostrará de una forma u otra, por ello se va a explicar cómo aparece la información en la configuración elegida para esta tesina. La configuración elegida es una configuración muy buena y potente puesto que con pocos parámetros de configuración se dispone de la información necesaria para saber cómo ha ido la ejecución de los test.

Si se escribe en la consola la instrucción phpunit como se indica en el apartado de configuración de los test la batería de test se ejecutará, mostrando dos posibles resultados.

En primer lugar se supone que todos los test se pasan sin problemas Código 4.43:

```
MacBook-Air-de-Javier:silex javier$ phpunit
PHPUnit 3.7.13 by Sebastian Bergmann.

Configuration read from /Users/javier/proyectos/silex/phpunit.xml.dist

.................................................

Time: 1 second, Memory: 40.50Mb

OK (51 tests, 100 assertions)
```

Código 4.43 Resultado test OK

Se va a realizar la explicación del cuadro anterior por fragmentos.

Como primera línea se observa la instrucción phpunit para lanzar los test.

A continuación aparece información sobre la versión de PHPUnit que está instalada (3.7.13) y el nombre de su creador (Sebastian Bergnam).

La tercera línea informa del fichero de configuración que se ha leído y se va a ejecutar, en nuestro caso el fichero (Users/Javier/proyectos/sílex/phpunit.xml).

Hay que prestar mucha atención a las tres últimas líneas, la primera de estas tres a simple vista parece que sea una línea de puntos sin más importancia pero cuando un test se ejecuta de forma correcta se introduce un punto, por esta razón se observa que tenemos 51 puntos. La penúltima línea da información sobre el tiempo que ha tardado y la memoria que se ha utilizado en la ejecución de los mismos.

Por último se ve la última línea, donde indica que se han realizado 51 test, 100 assertions y que el resultado ha sido positivo, mediante la primera palaba OK.

Ahora se va a estudiar el caso contrario, el caso en que los test no se ejecuten correctamente para ver cómo se detecta dónde está el error, y el motivo por el que falla.

Una vez se han lanzado los test PHPUnit devuelve una respuesta como la mostrada en Código 4.44:

```
MacBook-Air-de-Javier:silex javier$ phpunit
PHPUnit 3.7.13 by Sebastian Bergmann.

Configuration read from /Users/javier/proyectos/silex/phpunit.xml.dist

F.................................................

Time: 1 second, Memory: 40.50Mb

There was 1 failure:

1) Notifications\Test\AtomicEventTest::insertAtomicEvent
Failed asserting that '4' matches expected 5.

/Users/javier/proyectos/silex/src/Notifications/tests/Notifications/Tests/AtomicEventTest.php:73

FAILURES!
Tests: 51, Assertions: 100, Failures: 1.
```

Código 4.44 Resultado test FAIL

La primera diferencia se encuentra en la línea de puntos, donde en primer lugar no aparece un punto si no un F (Failure), para indicar que el un test ha fallado.

PHPUnit es capaz de dar mucha más información. Es capaz de indicar el número de fallos que se han cometido, en que clase y en que test ha fallado (AtomicEventTest::insertAtomicEvent()), el valor esperado y el que se ha calculado en los test (valor esperado = 5, devuelto por el método = 4). Línea en

la que se encuentra el Assert que ha fallado (/Users/javier/proyectos/silex/src/Notifications/tests/Notifications/Tests/AtomicEventTest.php:74).

La última línea indica que existen fallos mediante la palabra FAILURE, y una información similar a cuando los test se ejecutan correctamente, añadiéndole en última posición el número de test que han fallado, en nuestro caso 1.

4.4.2 Pruebas Aplicación Android

Para comprobar el funcionamiento de la aplicación Android se ha optado por la utilización de pruebas manuales. Esta técnica consiste en ir probando todas las funcionalidades de la aplicación intentando forzar el fallo, para poder solucionar los problemas en el momento se van descubriendo.

El programa ha sido testeado por diferentes usuarios, con varios niveles de familiarización con dispositivos Android, viendo problemas tanto de funcionalidad como de usabilidad pudiendo corregir estos consiguiendo una aplicación más estable.

La utilización de pruebas manuales de software ha sido posible gracias al desarrollo incremental. Debido al uso de esta metodología se ha podido comprobar la correcta funcionalidad de pequeños requisitos evitando así fallos más difíciles de encontrar cuando el proyecto va creciendo.

5 Manual de Usuario

En el capítulo que vamos a desarrollar a continuación vamos a describir cómo se puede utilizar el sistema desarrollado viendo como este funciona y como debemos utilizarlo.

5.1 Lista de los servicios disponibles

Nada más arrancar la aplicación se observa el listado de todos los servicios disponibles con los que el usuario puede interactuar. Esta información aparece mediante una lista de elementos, donde aparece un dibujo descriptivo de cada uno de los servicios y el nombre de los mismos.

Para continuar trabajando con la aplicación se indica con qué servicio se va a trabajar, y esto se indica a la aplicación pulsando en el servicio elegido.

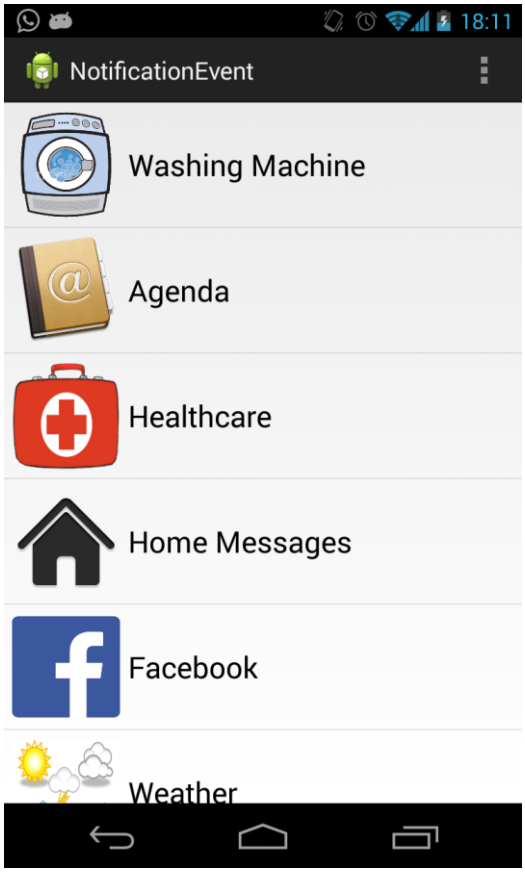


Figura 5.1 Listado de los servicios disponibles

En la ilustración se observa el listado de los servicios disponibles, mediante un scroll vertical es posible deslizarse por la lista buscando el servicio que deseamos en caso que no se encuentre visible por tamaño de la pantalla.

En la Tabla 5.1 se muestran los servicios disponibles y una breve descripción de cada uno de ellos:

Servicio	Descripción
Healthcare service	Este servicio permite gestionar la salud del usuario. Genera alertas para saber cuándo se tiene que tomar las pastillas o las citas con el médico entre otras.
Agenda service	Este servicio permite gestionar el tiempo al usuario dando acceso a las tareas de su calendario. Además, los usuarios pueden habilitar la recepción de eventos para recordar las diferentes tareas.
Home Messages service	Este es un servicio de mensajes de texto que permite a los miembros de la familia comunicarse con otros miembros de la familia. Este servicio se utiliza en el contexto del núcleo de la familia.
Washing Machine service	Este servicio permite a los usuarios estar informados de cuando la lavadora está completa y lista para empezar. Además, se notifica cuando el ciclo de lavado ha finalizado.
Shopping service	Este servicio se encarga de recordar los artículos que debe comprar el usuario cuando se encuentra cerca de un supermercado.
Weather service	Este servicio proporciona información sobre el tiempo, ofrece predicciones, avisos y recomendaciones para los usuarios.
Facebook service	Es un servicio de notificaciones de Facebook que muestra las notificaciones del usuario.

Tabla 5.1 Definición de los servicios disponibles

A continuación, se expone la Tabla 5.2**,** en esta tabla se muestran todos los servicios explicados anteriormente junto con la imagen descriptiva asociada,

para así permitir al usuario asociar el servicio de una forma más sencilla y cómoda.

Servicio	Imagen
Healthcare service	
Agenda service	
Home Messages service	
Washing Machine service	
Shopping service	
Weather service	
Facebook service	

Tabla 5.2 Imagenes de los servicios disponibles

5.2 Selección del servicio

Para poder continuar utilizando la aplicación se debe seleccionar el servicio del cual se quiere obtener ver el historial de las configuraciones pasadas, para ello un vez se presiona el servicio disponible aparece un mensaje en la pantalla indicando el servicio que se ha seleccionado mientras se carga toda la información disponible relacionada.

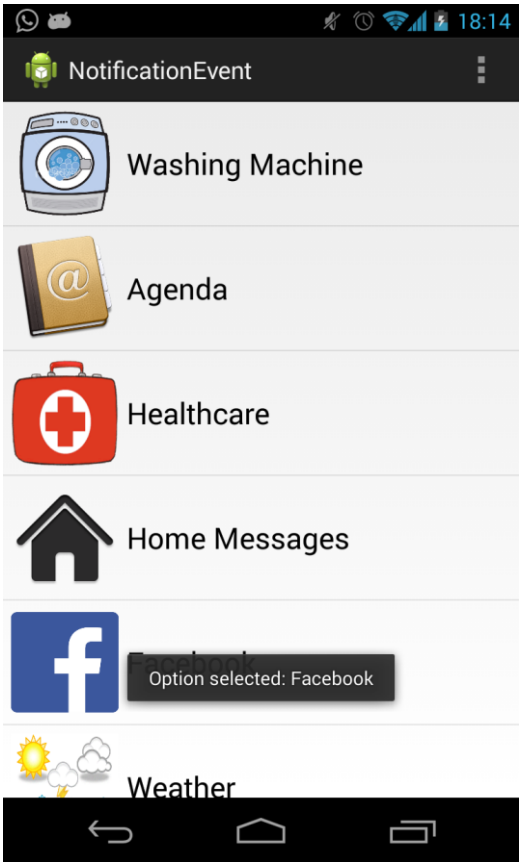


Figura 5.2 Seleccionar servicio

En la Figura 5.2, se observa que hemos seleccionado el servicio Facebook. Esto se indica mediante un Toast de Android, utilizando este para dar la información al usuario y para hacer la espera más corta y amigable mientras se accede y se almacenan los datos del servicio Facebook.

5.3 Vista del historial de configuración

Cuando se ha seleccionado la opción de todas las disponibles para trabajar se puede visualizar tanto la configuración actual como los historiales anteriores, por ello en primer lugar aparece un listado de todos los estados de la preferencia en el estado actual y a continuación todos los estados posteriores ordenados del más moderno al más antiguo.

Para indicar la fecha de utilización de esas preferencias en la parte superior se muestra una fecha para que el usuario final sea consciente en cada momento de a qué día pertenece el historial al que se ha accedido.

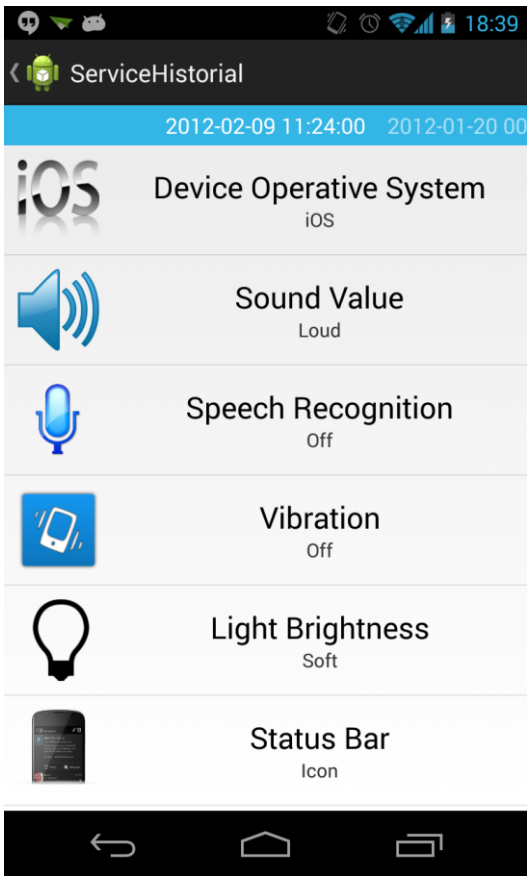


Figura 5.3 Estado actual del servicio seleccionado

La Figura 5.3 muestra el estado de la configuración actual, viendo para cada una de las adaptaciones de la configuración de cada servicio un dibujo representativo del estado de la notificación acompañado con un texto de la preferencia y del valor de esta como información adicional.

Si se visualiza la parte inferior de la imagen no aparece el botón para restaurar las preferencias seleccionadas ya que este es el estado actual y seleccionar

estas seria cambiar una configuración a la misma configuración pudiendo crear esto carga de datos innecesarios en el servidor.

Si queremos ver historiales anteriores únicamente debemos deslizar la vista hacia el lado derecho o izquierdo. Si se quiere desplazar la vista a un historial más antiguo se selecciona el movimiento hacia la derecha, si se quiere seleccionar un historial más moderno debemos deslizarnos hacia la izquierda.

Vamos a desplazarnos hacia una vista anterior para restaurar las preferencias.

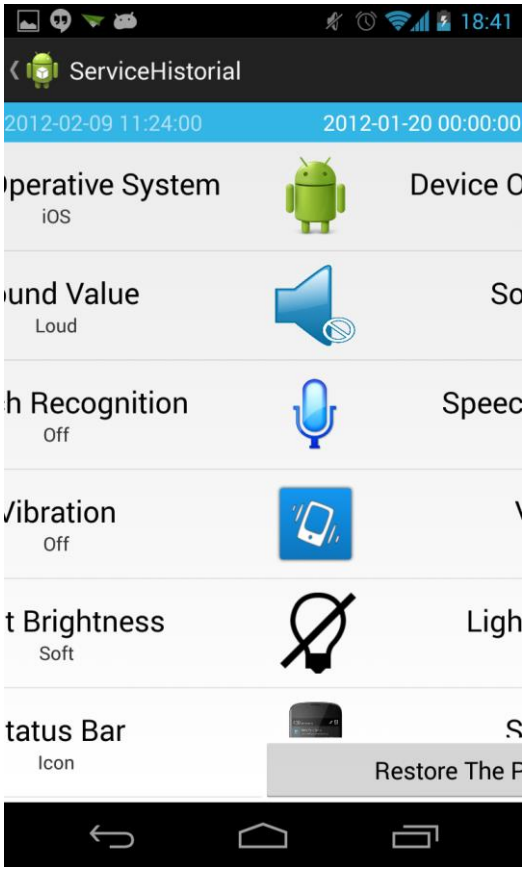


Figura 5.4 Navegación entre historiales

En la Figura 5.4, podemos ver la transición de un historial a otro.

Una vez la transición se ha realizado aparece las preferencias con estado anterior:

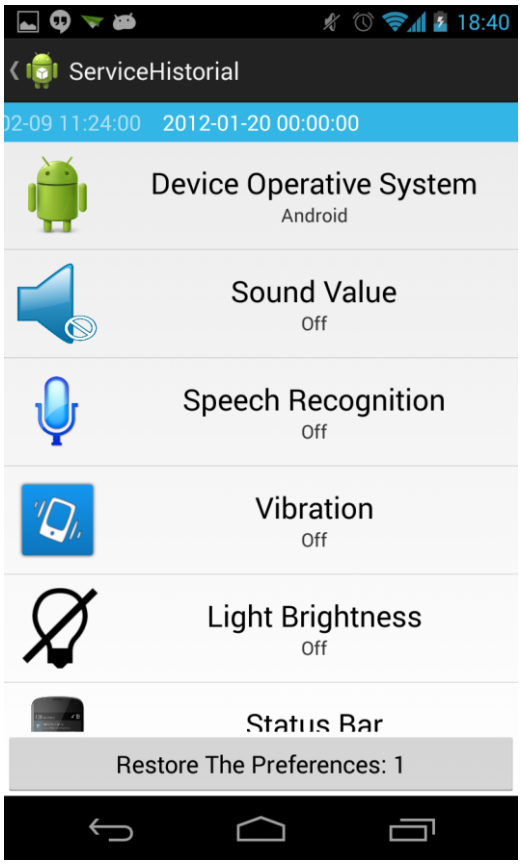


Figura 5.5 Historial relacionado con el servicio seleccionado

Si se fija la atención en las imágenes utilizadas observamos ciertos cambios según el valor de la preferencia que se obtienen, esto es visible en el altavoz o en el brillo, entre otras.

También aparece el botón que permite restaurar las preferencias de la vista seleccionada.

5.4 Restaurar Preferencias

Para restaurar las preferencias una vez hemos visto el estado que más se adapta al usuario se debe presionar el botón “Restore Preferences”. Pulsando este botón se indicara que se van a restaurar las preferencias mediante un aviso Toast y se recargara la vista, mostrando en primera posición las preferencias que acaban de ser seleccionada y un historial de todos los estados anteriores.

Si se quiere acceder a otro servicio para realizar las mismas opciones únicamente se debe volver al menú principal, donde se muestra la lista de todos los servicios y seleccionar el que se necesita restaurar, repitiendo la

actualización de estados siempre que se quieran modificar las preferencias de un servicio.

Una vez la aplicación se ha recargado aparecerá un nuevo historial pudiendo restaurar estas preferencias cuando el usuario final lo considere oportuno según nuestras necesidades.

En la Tabla 5.3 se va a realizar una breve descripción de las preferencias configurables para cada uno de los servicios:

Preferencia	Descripción
Device Operative System	Muestra el sistema operativo del dispositivo que se ha utilizado.
Sound Value	Valor de los sonidos que reproduce el dispositivo móvil.
Speech Recognition	Define si el reconocimiento de voz del dispositivo está activado o desactivado.
Vibration	Intensidad de la vibración que el dispositivo móvil tiene configurada.
Light Brightness	Intensidad de la iluminación del dispositivo.
Status Bar	Indica si se deben recibir notificaciones en la barra de notificaciones.
Toast	Muestra si se deben utilizar notificaciones Toast.
Dialog	Identifica si se deben mostrar notificaciones de tipo dialog.
Change Date Time	Guarda la fecha y hora del cambio de preferencias del dispositivo.

Tabla 5.3 Descripción de las preferencias

Para acabar, se va a mostrar la Tabla 5.4. En esta tabla se muestran todas las preferencias disponibles unidas con el valor que pueden adoptar según los parámetros del sistema y del contexto. Además se van mostrar las imágenes para cada uno de los diferentes valores, para así darle una información al usuario de forma más rápida e intuitiva.

Preferencias	Estados	Imágenes
Device Operative System	iOS	
	Android	
Sound Value	No	
	Soft	
	Loud	
Speech Recognition	No	
	Yes	
Vibration	No	

		
	One	
	Pattern	
Light Brightness	No	
	One	
	Pattern	
Status Bar	No	
	Icon	
	Icon & Text	

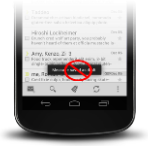
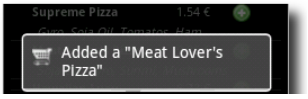
Toast	No	
	Yes	
Dialog	No	
	Text	
	Text & Buttons	

Tabla 5.4 Estados posibles de las preferencias

6 Casos de Uso

En este capítulo se presenta un caso de estudio real para aplicar sobre él todos los conceptos, procedimientos y la herramienta desarrollada que han expuesto y detallado a lo largo de los capítulos anteriores.

El principal objetivo del capítulo es el de comprobar el uso de la herramienta ante un día real y de esta forma exponer su aplicación sobre casos de estudio reales. Para ello vamos a realizar el siguiente caso de estudio.

Para finalizar con el capítulo se va a realizar un enfoque empresarial viendo el funcionamiento de la aplicación en el ámbito de un hospital. Además se va a añadir el ámbito de la automoción. Un ámbito muy interesante para explotar el potencial de la solución propuesta.

6.1 Presentación del caso de uso

El caso de estudio elegido consiste en simular un día completo de un de un usuario. Debido a diferentes circunstancias las preferencias de todos sus servicios cambian habitualmente y no siempre son de una forma correcta.

Se ha escogido este caso de estudio ya que se presentan diferentes opciones que pueden ser de interés:

- Se necesitan tener almacenados todos los datos disponibles, ofreciendo estos una gran información de los estados del usuario. Gracias a los datos como el brillo, el sonido, la vibración, etc. Se debe crear una experiencia de usuario correcta.

- Se necesita que el acceso y la creación de datos sea de una forma rápida y correcta. No se deben tener largos momentos de espera puesto que esto es molesto para el usuario final.

- En caso de que las preferencias seleccionadas en ese momento no sean las correctas debe poder configurarse de una forma correcta. Esto debe hacerse de una forma correcta, rápida y es posible de forma autónoma.

- Debe ser seguro, las preferencias almacenadas pertenecen a un usuario determinado, por ello las opciones que se utilicen deben de pertenecer a este.

- Debe estar en contacto con el mundo real, las alertas se proporcionan mediante proveedores de eventos. Estos provienen desde muchos lugares del mundo, por ello debemos estar pendientes de las alertas recibidas.

Se puede observar que el proyecto general cumple todas las características de un sistema auto-adaptable que plantea Horn en su manifiesto de computación autónoma. (Horn, 2001).

Debido a que no es un sistema crítico, ya que no dependen vidas humanas de él, tenemos un cierto margen de error. Estos fallos perjudicarían gravemente la experiencia de usuario por ello no se pueden permitir estos fallos haciendo que la modificación del sistema en tiempo de ejecución es algo muy interesante.

Como usuario vamos a utilizar un joven trabajador que dispone de un teléfono móvil con sistema operativo Android. Vamos a describir un viernes común en el cual realizará varias tareas como son, trabajar o pasar la tarde con la familia.

El caso de uso se va a realizar en relación con un grupo básico de servicios puesto que realizar la presentación con un número elevado de estos sería más confuso para el lector y perjudicaría notablemente el entendimiento del texto. El grupo de servicios elegidos para el caso de uso son las siguientes: Facebook, Healthcare, Agenda, Speech. Además, para no mostrar todas las preferencias cada vez que se modifique un servicio se van a definir tres valores para conocer el nivel de aviso; intrusividad baja, media y alta.

6.2 Escenario matutino

En este apartado vamos a explicar cómo transcurriría la primera parte del día.

Como todas las mañanas suena el despertador y por ello comienza el primer escenario para nuestro usuario. El móvil es consciente que comienza la jornada laboral y por ello se activan todas las conexiones, brillos de pantalla, sonidos y todas las demás preferencias de una forma que permitan al usuario enterarse de los avisos sin ser molestos para sus familiares.

Servicios	Estado Alerta	Estado Futuro
Todos	Intrusividad Baja (mientras duerme)	Intrusividad Media (mientras puede resultar molesto para la familia)

Tabla 6.1 Cambios de las preferencias al despertar

Una vez salimos de casa y entramos en el garaje se activa la configuración para viajar en coche, pero hoy hace un buen día y el usuario decide desplazarse en moto creando al móvil cierta confusión. Al salir por la puerta del garaje y el móvil no estar conectado al coche deduce que viajamos en moto y se auto-adapta, creando unas preferencias correctas para cuando viajemos en este tipo de vehículos. El móvil aprende que solo debe informar de las preferencias catalogadas como muy importantes.

Servicios	Estado Alerta	Estado Futuro
Healthcare	Intrusividad Media (mientras duerme)	Intrusividad Alta (hasta nueva orden)
Resto	Intrusividad Media (mientras duerme)	Intrusividad Baja (trayecto al trabajo)

Tabla 6.2 Cambios de las preferencias trayecto trabajo

Una vez entramos en el despacho y pasamos el teléfono por el RFID de la entrada, las preferencias se modifican desactivando las adaptaciones de los servicios poco importantes para este escenario como son las proporcionadas por Facebook o las del canal meteorológico entre otros. Como el usuario está en el despacho todos los volúmenes están activados por si recibe un aviso por parte de sus jefes.

Servicios	Estado Alerta	Estado Futuro
Todos (Excepto Healthcare)	Intrusividad Baja (trayecto al trabajo)	Intrusividad Media (en el despacho)

Tabla 6.3 Cambios de las preferencias durante la estancia en el despacho

Todos los viernes a las 12:00 se realiza una reunión para ver cómo han ido los objetivos de la semana, en esta reunión está completamente prohibida la utilización de alertas sonoras en los teléfonos móviles, al no ser que sean por motivos de salud. Una vez se entra en la sala de reuniones y mediante el sensor de presencia se cambian las preferencias poniendo todas las alertas de una forma visual no molesta y todas las alertas sonoras apagadas, excepto las recibidas por el seguro médico que están en un estado de aviso máximo.

Servicios	Estado Alerta	Estado Futuro
Todos (Excepto Healthcare)	Intrusividad Media (despacho)	Intrusividad Baja (reunión)

Tabla 6.4 Configuración de las preferencias durante la reunión

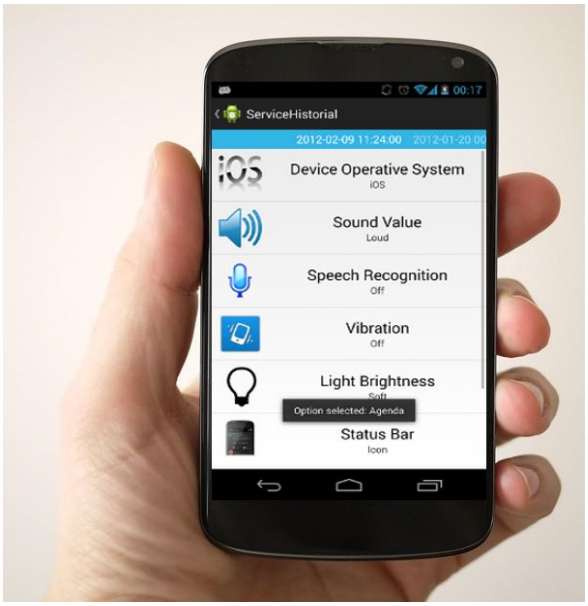


Figura 6.1 Ejemplo Caso de uso 1

Con esta reunión acaba la jornada laboral y justo al salir por la puerta de la oficina, el móvil selecciona las preferencias para viajar en moto, ya que ha aprendido y sabe que volvemos en moto a casa.

Como podemos observar todos los cambios han resultado correctos, y no hemos necesitado acceder al historial de preferencias para seleccionar unas que se adapten mejor a los gustos del usuario. Demostrando que el sistema funciona de una forma correcta y sobre todo se auto-adapta en caso de fallo.

6.3 Escenario vespertino

Como segundo escenario vemos cómo sería una tarde de nuestro usuario, en la cual va a realizar las tareas comunes de un viernes cualquiera excepto una, en la cual se deberán restaurar las preferencias del historial.

A las 18:00 toda la familia va en dirección al centro comercial donde van a pasar toda la tarde. Esta vez sí que se desplazan en el coche por ello se han seleccionado las preferencias para el móvil, activando el bluethooth o el reconocimiento vocal, para que el móvil se pueda utilizar mediante la voz.

Una vez en el centro comercial y justo en el acceso llega un evento para avisarnos que entramos al centro comercial. El parking estaba bastante completo y han aparcado en la entrada más cercana al cine. El evento que ha

llegado al móvil es de la entrada del cine por ello el móvil cree que se va a entrar al cine y se utilizan unas preferencias de cine.

Servicios	Estado Alerta	Estado Futuro
Todos (Excepto Healthcare)	Intrusividad Media (trayecto al centro comercial)	Intrusividad Baja (detectar cine)

Tabla 6.5 Configuración de las preferencias con la familia

Esto es incorrecto así que el usuario entra al historial del servicio “Agenda”, y selecciona las que considera más correctas para el estado actual, consiguiendo así una configuración de preferencias que se adapta a la perfección al usuario final.

Servicios	Estado Alerta	Estado Futuro
Agenda	Intrusividad Media (jornada laborar)	Intrusividad Baja (trayecto a casa)

Tabla 6.6 Modificación de las preferencias del servicio "Agenda"

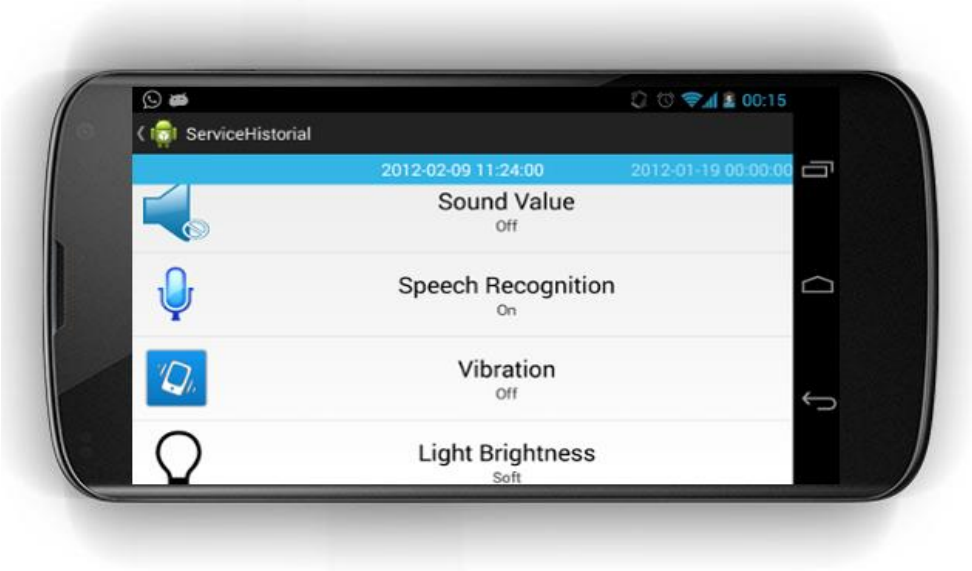


Figura 6.2 Ejemplo Caso de uso 2

En este escenario ha ocurrido un fallo pero accediendo al historial se ha podido seleccionar un estado del teléfono que se adaptaba de forma correcta a las circunstancias.

6.4 Ámbito Empresarial

A continuación, se va a aproximar este trabajo final de máster a un ámbito empresarial, detallando las posibilidades que en un futuro puede aportar.

La propuesta desarrollada puede ser de gran utilidad y por ello se va a mostrar la aplicación en el ámbito de un hospital, un escenario con muchos cambios de contexto y por lo tanto con muchas auto-adaptaciones.

Para mostrar otra posible utilización se va a desarrollar el caso de un coche con un dispositivo Android incorporado que es capaz de adaptar diferentes configuraciones del mismo.

6.4.1 Utilización en un Hospital

Para comenzar, vamos a situar el entorno en un hospital, siendo el usuario de la aplicación un médico llamado Álvaro. Debido a las necesidades de su oficio el contexto está cambiando constantemente, por ello la aplicación puede ser de gran ayuda. También se debe exponer que el dispositivo móvil utilizado no es personal, es propio del hospital y por lo tanto no recibe notificaciones de servicios externos al trabajo como Facebook. El propio hospital dispone de una lista de servicios, como aviso para reuniones, alertas de nuevos ingresos, falta de medicamentos, resultado de análisis, etc.

A primera hora de la mañana y cuando Álvaro se encuentra en su despacho hablando con sus ayudantes, el dispositivo móvil adapta un estado relacionado con el contexto, es decir las notificaciones no deben comportarse de forma intrusiva al no ser que sea una alerta grave por necesidades de salud de un paciente. Por ello todos los servicios que dispone la aplicación se encuentran con las alertas sonoras desactivadas encontrándose únicamente conectadas las alertas visibles.

Servicios	Estado Alerta	Estado Futuro
Emergencia	Intrusividad Alta (durante la jornada laboral)	Intrusividad Alta (durante la jornada laboral)
Resto	Intrusividad Media (mientras está solo)	Intrusividad Baja (reunión con compañeros)

Tabla 6.7 Modificación de las preferencias durante la reunión

Una vez acaba la visita de los ayudantes, el médico se dispone a pasar visita yendo una por una a todas las habitaciones para ver cómo se encuentran sus pacientes. En el dispositivo Álvaro introduce toda la información que considera oportuna de los pacientes para poder completar así el historial, el tratamiento y demás cuidados. Durante toda la visita no se debe molestar al médico puesto que daría una mala imagen al paciente y a pesar que llegan diversas notificaciones nadie se percibe de estas, excepto Álvaro que las observa en la barra de notificaciones.

Figura 6.3 Utilización en un hospital (1)

Una vez se ha acabado de pasar visita, el usuario se encuentra en la hora del almuerzo, el instrumento de gestión es consciente que en el usuario se encuentra en la cafetería y por ello se auto-adapta permitiendo a los servicios comportarse de una forma más intrusiva, pudiendo estos utilizar vibraciones, ventanas modales, cuadros de dialogo o sonidos.

Servicios	Estado Alerta	Estado Futuro
Todos	Intrusividad Baja (reunión)	Intrusividad Media (durante el almuerzo)

Tabla 6.8 Modificación de las adaptaciones durante el almuerzo

Durante el almuerzo aparece por la cafetería el jefe de médicos del hospital que se sienta con Álvaro para almorzar, por ello el dispositivo de Álvaro vuelve a adaptar sus preferencias poniéndose en un modo menos molesto. Por otro lado se encuentra el instrumento del jefe de médicos, este es consciente del role de su usuario ya que ha conseguido aprender tras diversas restauraciones que a pesar de encontrarse en una situación que puede resultar molesta debe comportarse de una forma intrusiva.

Servicios	Estado Alerta	Estado Futuro
Todos	Intrusividad Media (mientras está solo)	Intrusividad Baja (con el jefe de médicos)

Tabla 6.9 Modificación de las adaptaciones con el jefe de médicos

Tras acabar el almuerzo y pasando consulta en su despacho, el dispositivo dispone de muchas dudas, a primera hora con los ayudantes debe comportarse de una determinada manera, cuando esta solo de otra, cuando esta con un paciente de otra y así múltiples posibilidades. En estos momentos se encuentra pasando consulta a enfermos por ello continuamente están cambiando las necesidades. Mientras dura la visita se reciben avisos constantemente puesto que toda la información del hospital se trata sobre estos aparatos. Álvaro ha de ser consciente de todos estos, en cualquier momento puede recibir el análisis de sangre que está esperando para un paciente o la radiografía para poder diagnosticar si existe rotura ósea o no. En primer lugar se recibe el aviso de un nuevo paciente en la sala de espera, cosa que en estos momentos no debe molestar al usuario pero seguidamente se reciben los resultados del paciente que se encuentra en el despacho, información de gran interés y por ello aparecen de forma modal en la pantalla del dispositivo. Como se puede observar hasta el momento todas las auto-adaptaciones han sido correctas.

Cuando se acaba de atender a todos los usuarios con cita el usuario se va a quedar en el despacho preparando informes para el día siguiente, el dispositivo debe ser consciente que a pesar de estar en el despacho no dispone de compañía y por lo tanto se pueden utilizar alertas sonoras en caso de que uno o varios servicios lo necesiten.

Servicios	Estado Alerta	Estado Futuro
Todos	Intrusividad Baja (con el jefe de médicos)	Intrusividad Media (elaboración de informes)

Tabla 6.10 Modificación de las adaptaciones durante la elaboración de informes

Durante la elaboración de los informes, el jefe de médicos, que se habían encontrado anteriormente, entra al despacho para hablar del partido de pádel que habían jugado la noche anterior. El dispositivo es consciente que ha entrado visita y además que es el jefe por ello se adapta a un estado de intrusión mínima. Esta vez la configuración es incorrecta, es una reunión informal y Álvaro gestiona las adaptaciones seleccionando un servicio de configuración para todos los servicios con una capacidad de intrusión mayor.

Figura 6.4 Utilización en un hospital (2)

Como se ha observado durante toda la utilización en el hospital, tener un software de estas características sería de gran ayuda para los trabajadores. En el ejemplo se ha explicado la posible utilización en un médico, pero esto sería posible para todos los trabajadores del hospital. Además se ha introducido un nuevo sistema de servicios propios del hospital no utilizando los servicios personales de los usuarios. Esto será extrapolable a otros ámbitos, como por ejemplo a la policía o a los bomberos creando una serie de servicios diferentes para cada oficio, adaptándose siempre de forma correcta.

6.4.2 Utilización en un Automóvil

Otro ámbito muy interesante para la utilización de la aplicación puede ser el ámbito de la automoción. En la actualidad es difícil encontrar un coche nuevo que no disponga de una pantalla táctil para controlar todo el sistema electrónico, y poco a poco se va introduciendo el acceso a internet en los vehículos. También hay que decir que cada día se busca más la conectividad entre el dispositivo móvil y el automóvil llegando incluso a poder abrir el coche con el teléfono móvil. Cuando un usuario accede a su vehículo es muy interesante que éste sea el encargado de gestionar el dispositivo puesto que se está prestando atención a la carretera y no puede manejarlo.

Todos los avisos que deba proporcionar deben de ser gestionados por el vehículo, bien mediante los altavoces, un aviso en la pantalla de navegación o con una proyección directamente sobre el parabrisas delantero.

Figura 6.5 Pantalla navegación Android

Como se ha explicado anteriormente no queremos intrusión al usuario por ello si se recibe un aviso y se está manteniendo una conversación con un acompañante no queremos que el aviso aparezca de forma sonora por los altavoces, es más conveniente que aparezca en el panel de navegación. Pero si por ejemplo se está utilizando el panel de navegación resultaría molesto que apareciera de forma intrusiva el aviso, por ello el móvil debe comportarse de una forma correcta según las circunstancias.

Otro punto de vista dentro de los vehículos sería la configuración de diferentes parámetros de conducción como por ejemplo las luces, limpia parabrisas o el bloqueo de puertas. Hoy en día y gracias a diferentes sensores son capaces de encenderse o apagarse pero la propuesta llega más lejos, sería una configuración propia y auto-adaptativa para cada usuario. Por ejemplo si a un usuario del vehículo le gusta encender las luces durante todo el dia, este debería ser consciente y que estén encendidas aunque el sensor de luminosidad dicte lo contrario, por otro lado un usuario solo quiere llevarlas encendidas de noche el coche deber ser consciente y solo encenderse de noche. Como se ha comentado en los puntos anteriores todos los cambios deben almacenarse y permitir una completa gestión de las adaptaciones para cada servicio al usuario.

6.5 Conclusiones

Como conclusiones al caso de uso se observan ventajas muy importantes en este ámbito, como es que el sistema se adapta mediante eventos. Gracias a estos el móvil logra comportarse de una forma correcta en todo momento. También hay que destacar la gran facilidad de reconfiguración en caso de un fallo.

Como ventaja se ve que se puede ganar mucho tiempo si no tenemos que configurar en cada momento las preferencias. Además de esto se evita que un olvido a la hora de configurar las preferencias puedan evitar una serie de problemas en cualquier momento.

Añadir que la aplicación puede ser utilizada en muchos ámbitos diferentes. Día tras día la inclusión de dispositivos móviles en los trabajos es más amplia y disponer de un sistema que mediante eventos permita la configuración automática del dispositivo es muy útil. Hay que destacar que la aplicación funciona con Android, por lo tanto se podría adaptar a cualquier dispositivo que trabaje bajo este sistema operativo. Debido al “Internet of things” y el crecimiento esperado por Android en distintos dispositivos como coches o relojes hacen un software muy interesante.

7 Conclusiones y trabajo futuro

En este último capítulo vamos a presentar las conclusiones extraídas una vez desarrollada la propuesta que motivó esta tesis de máster, así como que el trabajo que aún se podría mejorar una vez completada esta.

7.1 Conclusiones

Las aproximaciones actuales para desarrollar sistemas auto-adaptables según el contexto se han creado asumiendo que todos los comportamientos del sistema han sido anticipados en tiempo de diseño. Sin embargo, con el paso del tiempo todo sistema requiere evolucionar para dar soporte a cambios, mejoras, extensiones y sobre todo a la adaptación a nuevas tecnologías, es decir, un sistema debe reciclarse para evolucionar y ser un sistema de una gran calidad durante muchos años.

En este trabajo fin de máster se ha propuesto una herramienta donde se presenta al usuario todos los cambios creados en las preferencias de sus dispositivos móviles permitiendo realizar roll-back a cualquier configuración de preferencias anteriores. Toda la auto-adaptación de la configuración se genera mediante modelos en tiempo de ejecución.

Concretamente se proporciona una herramienta de visualización de adaptaciones end-user. Con esta herramienta se permite al usuario gestionar las auto-adaptaciones generaras en tiempo de ejecución, permitiendo volver a estados anteriores en caso que estas no sean del total agrado del usuario. Además esta herramienta se encarga de toda la propagación de datos hacia el servidor.

Además de la herramienta explicada anteriormente se ha desarrollado una completa API REST de comunicación. Con esta se proporciona todo el acceso al servidor de almacenamiento de adaptaciones. En este servidor se reúnen todos los datos necesarios para la total gestión de las auto-adaptaciones así como toda la información necesaria para el correcto funcionamiento de la herramienta.

7.2 Trabajo Futuro

El trabajo realizado en esta tesis de máster no es trabajo cerrado, sino todo lo contrario, es una línea de investigación abierta donde se pueden abordar muchísimas mejoras. A continuación vamos a proponer una serie de principales trabajos futuros que se podrían realizar:

- **Utilización de una ontología:** Cambiando el almacenamiento de datos a una ontología se podría crear un sistema donde la comunicación entre diferentes sistemas y entidades sea más sencilla. (Deborah L. McGuinness, 2004)

- **Utilización de la aplicación en modo Offline:** En este estado de maduración de la aplicación, esta necesita una conexión total a internet, si no disponemos de esta la aplicación no funciona. Sería muy interesante poder almacenar los datos en una base de datos local al teléfono cuando el teléfono esté en modo Offline y enviar toda la información al servidor cuando dispongamos de conexión a Internet.

- **Generación de modelos en la restauración de datos:** También se plantea como trabajo futuro implementar el envío de datos al módulo donde se generan los modelos, para poder crear estos de una forma más rápida y en paralelo mientras se almacenan en el servidor.

- **Otros sistemas operativos (iOs, Windows Mobile):** Otra posible ampliación sería la de crear esta aplicación para otros sistemas operativos móviles como pueden ser iOs, o Windows Mobile, ya que en este momento únicamente se ha desarrollado la aplicación para móviles con un sistema operativo Android.

- **Utilización mediante reconocimiento vocal:** Finalmente, se propone una aplicación para poder manejar toda la aplicación mediante reconocimiento vocal. Esto sería muy interesante para poder utilizarla en situaciones donde no disponemos de la libertad de las manos necesaria para poder utilizarla como puede ser, por ejemplo mientras conducimos.

8 Bibliografía

Joseph, J. (2002). *Handling attachments in SOAP.*

json org. (2013). Obtenido de http://www.json.org/: http://www.json.org/json-es.html

Android. (2013). *Llama Location Profiles*. Obtenido de https://play.google.com/store/apps/details?id=com.kebab.Llama&hl=es

Bloch, J. (2006). *How to design a good API and why it matters.*

Bloch, J. (s.f.). *How to Design a Good API and Why it Matters*. Obtenido de How to Design a Good API and Why it Matters: http://lcsd05.cs.tamu.edu/slides/keynote.pdf

Brahler, S. (2010). *Analysis of the Android Architecture.*

Burnete, E. (2009). *Hello, Android: Introducing Google's Mobile Development Platform.*

Calvary G, C. J. (2003). *A Unifying Reference Framework for multi-target user interfaces.*

Crockford, D. (2006). *The application/json Media Type for JavaScript Object Notation (JSON).*

David Booth, H. H. (2004). *Web Services Architecture.*

De Lemos, R. G. (2011). *Software Engineering for Self-Adpaptive Systems: A second Research Roadmap. In: Software Engineering for Self-Adaptive Systems.* No. 10431 in Dagstuhl Seminar Proceedings.

Deborah L. McGuinness, F. v. (2004). *OWL Web Ontology Language Overview.*

Doctrine. (2013). *Doctrine Project*. Obtenido de Doctrine Project: http://www.doctrine-project.org/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.*

Fowler, M. (2010). *http://martinfowler.com/.* Obtenido de http://martinfowler.com/articles/richardsonMaturityModel.html

Gamma E., H. R. (1995). *Design Patterns: Elements of Reusable Object Oriented Software .* Addison Wesley.

Gil Pascual, M. (2013). *Adapting Interaction Obtrusiveness: Making Ubiquitous Interactions Less Obnoxious.* Valencia.

Horn, P. (2001). *Autonomic Computing, IBM's Perspective on the State of Information Technology.*

Jens Jahnke, A. Z. (1997). *Rewriting poor Design Patterns by good Design Patterns.*

M. Gil, P. G. (2012). *1. Personalization for unobtrusive service interaction. Personal and Ubiquitous Computing .*

Marqués, A. (1 de Abril de 2013). *Conceptos sobre APIs REST*. Obtenido de http://asiermarques.com/2013/conceptos-sobre-apis-rest/

Maximilien, E. (2003). *Assessing test-driven development at IBM.*

Milanovic, N. (2004). *Current solutions for Web service composition.*

Moreno, F. (2012). *Show me the code*. Obtenido de Show me the code: http://showmethecode.es/php/silex/crear-un-proyecto-con-silex/

Nadav Savio, J. B. (2007). *The Context of Mobile Interaction. International Journal of Mobile Marketing, .*

Pautaso, C. (2007). *SOAP vs. REST Bringing the Web back into Web Services.*

Pressman, R. (2002). *Ingenieria del Software, un enfoque práctico.*

R. Fielding, J. G.-L. (1999). *Hypertext Transfer Protocol -- HTTP/1.1.*

Schmidt, D. (2006). *Guest editor's introduction: Model-driven engineering Computer .*

Shneiderman, B. (1998). *Designing The user interface, Strategies for effective Human-computer interaction.* Addison-wesley.

Venners, B. (2002). *Test-Driven Development A Conversation with Martin Fowler.*

Vogel, L. (2013). *Android ListView - Tutorial*. Obtenido de http://www.vogella.com/articles/AndroidListView/article.html

Página web oficial desarrolladores Android:

http://developer.android.com/intl/es/index.html