



# UNIVERSIDAD DE EXTREMADURA

## ESCUELA POLITÉCNICA

## GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DE SOFTWARE

## TRABAJO FIN DE GRADO



# UNIVERSIDAD DE EXTREMADURA

## ESCUELA POLITÉCNICA

## GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DE SOFTWARE

## TRABAJO FIN DE GRADO

**Extracción del contexto de ejecución del usuario**

**Autor: Alberto de la Fuente Cruz**

**Tutor: Juan Manuel Murillo**

# Resumen

Vivimos en una sociedad hiperconectada. Tenemos acceso a Internet en cualquier momento, en cualquier lugar. Podemos hablar con nuestra familia, amigos o pareja en cualquier circunstancia, cuando queramos. Hemos movido gran parte de nuestra vida a la tecnología, y lo que es peor, somos dependientes de ella.

Todo esto lo tenemos a un sólo gesto de distancia, el de sacar el teléfono del bolsillo. Llevamos con nosotros dispositivos inteligentes que son balizas de datos, que sólo están ahí, disponibles para quien los quiera consumir.

Esta ingente cantidad de datos que generamos, normalmente, van a parar a las compañías responsables del producto que estamos consumiendo en ese momento. Estos datos se almacenan y se realizan estudios con el objetivo de conocer mejor que nunca al consumidor. Pero este conocimiento es siempre sesgado, sólo pueden estudiar al usuario mientras interactúa con su servicio, lo cual les aporta únicamente una visión parcial de la realidad.

Debemos afrontar este hecho y, si la recolección de datos existe, tomar parte en ella, tanto desde el punto de vista del desarrollador/empresario como del usuario. Este trabajo nace con la idea de integrar en un único servicio una recolección completa, veraz y objetiva, a la vez que le abre la puerta al usuario para conocer los frutos de esa recolección.

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

Hoy en día, el uso de la tecnología y los servicios que nos ofrece Internet tienen un coste adicional invisible para el usuario, no siendo otro más que la invasión de su privacidad, la recolección de sus datos, el análisis de los mismos, su perfilado y en el peor de los casos, su venta al mejor postor.

Esta recolección de datos normalmente obedece a dos necesidades, por un lado, conocer al usuario del servicio y ofrecerle así una versión cada vez más personalizada del mismo, y por otro, la venta de esos datos a terceros.

Esta venta de datos, que parece un cuento de brujas, a día de hoy es una realidad, no hay más que ver uno de los casos más recientes donde *Facebook* vendió los datos de los perfiles de sus usuarios de EEUU a *Cambridge Analytica* [**?**], empresa dedicada al análisis y minería de datos, para influir en los resultados de las elecciones presidenciales.

Aún así, la recolección de datos no es mala per sé, por un lado, ayuda los desarrolladores a conocer al usuario y detectar posibles carencias y nuevas necesidades en sus aplicaciones. Por otro, ayuda al empresario, puesto que permite conocer el mercado, identificando necesidades específicas, focalizar el marketing, realizar una

publicidad efectiva e incluso identificar nuevos nichos de mercado.

Con lo cual, podemos concluir que conocer a tu usuario es algo bueno y necesario cuando se quiere crecer y ofrecer un mejor servicio, el problema viene cuando esta recolección de datos de realiza sin control alguno e invadiendo de forma directa la privacidad del usuario. Un ejemplo aún más reciente que el de *Facebook* es el de la aplicación oficial de La Liga [**?**], donde se mandan datos de ubicación y fragmentos de audio del micrófono cada 30 minutos a sus servidores bajo el pretexto de identificar fraudes. Esta recolección sin medida produce tal volumen de datos que podemos llegar a dudar de su validez y efectividad.

Pero, ¿cómo hemos llegado a esta situación? La culpa es del marketing y su técnica *Targeting Market*, y su aplicación directa en la informática a través del *Segmenting-Targeting-Positioning* [**?**], a la que nos referiremos simplemente como *Product Targeting*

Esta estrategia se basa en segmentar el mercado en grupos cada vez más pequeños de acuerdo a unas variables, que de forma tradicional son; Variables geográficas. En referencia a la región del mundo, clima, país, región, ciudad, etc. Variables demográficas. Edad, género, ingresos, orientación sexual, nivel educativo, cultura, etc. Variables psicográficas. Estilo de vida, personalidad, valores, intereses, etc. Variables conductuales. Tasa de uso del servicio, fidelidad con la marca, etc.

Estas variables (datos, al fin y al cabo) se someten y a un proceso de relación y análisis que permite profundizar cada vez más en la segmentación, dando lugar a la segmentación profunda y si se dispone de la suficiente cantidad de información, acaba produciendo un perfil de usuario. Es obvio que este perfil mejorará en tanto en cuanto mayor sea el volumen de datos del que se dispone.

Entra en juego entonces la fuente de estos datos, puesto que deben salir de algún sitio. A día de hoy, si nos ponemos en el contexto de Internet, la mayoría de las páginas, por no decir todas, disponen de herramientas analíticas que permiten conocer al menos, el país origen de la conexión al servicio así como la fecha y hora exactas en la que se produjo. Este es el primer nivel que está enormemente implantado en el mercado.

En un segundo nivel, y situándonos en el uso de los servicios que ofrecen las distintas compañías, es habitual que todos nuestros movimientos dentro de ella se recolecten con el fin de aportar a los desarrolladores información acerca del uso de las herramientas que ellos mismo han producido. Esto va desde las funcionalidades utilizadas hasta el registro del movimiento del ratón a lo largo de la interfaz.

En el tercer nivel, la recolección normalmente invade la privacidad del usuario, incluyendo datos como su posición GPS, capturas programáticas del sonido ambiente a través del micrófono, rastreo de conexiones a través de cookies, etc.

El problema derivado de este festival del rastreo viene de la mano con la enorme cantidad de datos que se acumulan en los servidores del rastreador. En otras palabras, no se realiza una recolección efectiva de datos que realmente aporten valor al conjunto, veamos un ejemplo;

Hace poco, escuchando un Podcast de desarrollo [**?**] donde el tema central era el BigData y las bases de datos orientadas a este, surgía en la conversación el tema del almacenamiento masivo de datos del usuario y cómo debían adaptar sus servicios para satisfacer la demanda de los desarrolladores. En un punto determinado del programa, hablabla Nuria Ruiz (@pantojacoder) [**?**], profesional que gestiona el equipo de Analitics de Wikipedia, ponía en relieve el problema al que se enfrentan en la plataforma, quizás desde un extremo poco habitual pero necesario en su servicio,

puesto que pretenden recolectar los menos datos posibles de sus usuarios para evitar persecución política de sus editores. En el caso expuesto a lo largo del programa, citaba como los desarrolladores querían rastrear todos los clicks de todos los usuarios para rastrear el uso de una feature que habían puesto recientemente en producción.

Nuria, cuyo trabajo consiste en el almacenamiento y análisis de estos datos, se lamentaba de que se exija tal cantidad de información cuando con la tercera parte de la misma las necesidades del equipo de desarrollo quedaban sobradamente cubiertas. Esto nos da una idea de cual es el problema al que debemos enfrentarnos, que no es otro que la recolección efectiva de datos que aporten valor al perfil del usuario, y no recolectar por recolectar.

Definamos entonces los límites de estos datos y su recolección, puesto que nos referimos a rastrear y registrar datos que verdaderamente nos permitan conocer al usuario, cuál es su rutina, el uso que realiza de las herramientas que dispone, para qué las emplea... en definitiva debemos aportar un contexto a los datos recolectados, puesto que sin este contexto estos datos únicamente son información en bruto. Debemos entonces realizar una recolección inteligente de la información de acuerdo al uso del servicio a lo largo de su ciclo de vida.

Este proyecto debe, por tanto, extraer esos datos del usuario de algún modo para almacenarlos en un servidor, donde más adelante serán procesados. El proyecto cubre por tanto la primera capa de esa idea, idear y desarrollar una aplicación capaz de capturar el máximo número posible datos del usuario, a la vez que permite al usuario una vista objetiva del uso que le da al terminal, sus rutinas y en definitiva, evaluar su grado de adicción a la tecnología, mostrando estadísticas de uso, acciones más habituales dentro de las aplicaciones, aplicaciones más frecuentes, etc.

La idea principal es alejarnos de la captura enfocada a una aplicación, como

viene siendo la norma en el sector. Esto únicamente aporta información de uso en una única aplicación, aportando así información sesgada e incompleta. Desaprovechando el potencial que esta captura de datos tiene para mejorar establecer un perfil psicológico directamente asociado al usuario.

Debemos por tanto diseñar e implementar un sistema capaz de reunir y analizar los datos de un usuario de tal manera que estos aporten un perfil completo del usuario, no centrado en una aplicación, si no en un conjunto de ellas (la intención es registrar las aplicaciones más usadas en cada ámbito) cuyos datos de uso nos ayuden a extraer un modelo de los rasgos de la personalidad de un usuario, incluyendo sus rutinas, sus gustos y preferencias, aficiones, datos que en última instancia nos ayuden a establecer un patrón asociado a una persona y detectar cambios en su comportamiento.

Esta información, además, debe ser pública y accesible a través de una API. Esta base de conocimiento solo estará accesible a los responsables del equipo de investigación, permitiendo así explorar el potencial que tiene la recolección masiva de datos, puesto que hasta ahora las compañías jamás han liberado los resultados de este proceso, convirtiéndose en fórmulas de éxito protegidas bajo siete llaves.

Se pretende así abrir nuevas vía de investigación, dado que la recolección de datos no sólo sirve para incrementar los beneficios de un negocio, si no que cuenta con el potencial suficiente como para tener un impacto real y positivo en la vida de las personas, más allá de sugerir amistades o zapatillas.

## 1.1. Motivación y desarrollo

El proyecto surge como primera etapa de un proyecto más grande enmarcado en un Trabajo de Fin de Máster, donde la idea es poder inferir patrones de comportamiento de los usuarios de teléfonos inteligentes y así poder diagnosticar y predecir trastornos de comportamiento dentro del patrón de una persona.

El desarrollo se dividirá en varias etapas. La primera será estudiar las herramientas existentes en el mercado y averiguar las técnicas que usan las compañías para capturar los datos del usuario, puesto que este es nuestro principal objetivo.

Se seguirá con la definición de los requisitos y casos de uso del proyecto para así acotar los términos del sistema y poder definir una primera arquitectura, independiente de la plataforma, que nos permita identificar los componentes fundamentales.

Una vez definida esta arquitectura a alto nivel, se procederá a acoplar lo planteado al sistema Android y definir los flujos que permitan obtener la lógica necesaria para realizar la captura. De esta forma, podremos definir una arquitectura a bajo nivel, dependiente de la plataforma, que permita organizar los componentes que forman el sistema final.

El desarrollo seguirá un proceso iterativo a base de *Sprints*, persiguiendo la idea de mantener siempre un artefacto software funcional que evolucionará conforme avance el proyecto.

# Capítulo 2

# Objetivos

Una vez establecidos los límites del dominio que se pretende manejar, vamos a establecer una serie de objetivos, partiendo de la base de que se implementará una aplicación Android donde el usuario no deberá tomar ningún partido a la hora de introducir los datos. Así pues;

- Se debe realizar una captura pasiva de los datos, es decir, no se debe interrumpir el flujo normal de interacción del usuario con su teléfono.
- Esta captura será pasiva en tanto en cuanto correrá una aplicación en segundo plano (servicio Android) que atenderá a un determinado tipo de eventos, siendo tarea del desarrollador implementar sendos modelos que extraigan la información perseguida.
- Esta información estará acotada en cinco grupos. Así, contaremos con un grupo de navegación web, mensajería instantánea, comunicación, redes sociales y sistema (sensores y notificaciones). Dentro de cada grupo se atenderán a las aplicaciones más utilizadas en cada campo.
- Esta captura debe ser tener valor semántico, es decir, debe tener información referencial que ayude a situar en un contexto la información.
- La información capturada será almacenada de forma temporal de forma local en el almacenamiento hardware del dispositivo, y, de forma programática realizar

volcados periódicos cada poco tiempo, con doble objetivo. Por un lado, no saturar el espacio de almacenamiento y por otro, asegurar los datos en un servidor remoto de manera que estos estén siempre disponibles.

- Este volcado se realizará a través de una API que permitirá el propio volcado y su consumo, comportándose así como una interfaz de acceso al manejo de los datos.

## 2.1. Planificación

Para conseguir estos objetivos se realizó una planificación inicial en el momento en el que se acordaron los términos del trabajo. Pero antes, un poco de contexto.

Este proyecto surgió en uno de los grupos de investigación de la Escuela Politécnica de Cáceres, en concreto, en el SpiLab. Como condición necesaria para hacer el trabajo de fin de grado, se estableció que debía asistir al laboratorio de Lunes a Jueves durante todo el curso. En esa planificación inicial, los tiempos estaban claros, durante el primer cuatrimestre realizaría labores de investigación y análisis en el horario establecido. Durante el segundo, realizaría el diseño y la implementación. De esta forma, la fecha de entrega estaba proyectada para el mes de Junio.

Pero es estimación no se pudo cumplir. Durante el primer cuatrimestre hice un parón para dedicarme a terminar las asignaturas pendientes que me quedaban, así, durante el mes de noviembre y diciembre no pude asistir al laboratorio. Ahí se rompió la planificación. Además, en Enero encontré trabajo, con lo cual el tiempo que podía dedicar al TFG, pese a no reducirse en gran medida, sufrió desgaste. A partir de ese momento sólo podía trabajar entre 5 y 6 horas por las tardes.

De esta forma, la planificación real que resume los tiempos trabajados, las fechas y las tareas dedicadas en cada una, se resume en la siguiente figura.

# Capítulo 3

# Estado del Arte

Está claro que para este proyecto no estamos inventando la rueda, el rastreo del usuario es una técnica que existe prácticamente desde que el uso de internet se hizo mayoritario, por ello vamos a ver como ha evolucionado este rastreo y captura a lo largo de los años.

Comenzaremos viendo las técnicas de análisis web, implantadas desde hace décadas, como primera forma de conocer a los usuarios a través de sus visitas a páginas y aplicaciones web.

Siguiendo el mismo camino que ha marcado el desarrollo de la tecnología, veremos como estas técnicas se han adaptado, persiguiendo el mismo objetivo, a los Smartphones, centrándonos además en el ecosistema Android.

Por último veremos el caso de los dispositivos IOT que poco a poco comienzan a hacer su aparición en el mercado, nos referimos a asistentes personales como Amazon Echo ó Google Home, de reciente aparición pero con potencial para instalarse en los hogares de muchas familias.

## 3.1. Análisis Web

Entendemos por análisis web la medición, recolección, análisis y procesado de los datos generados la navegación con la idea de comprender y optimizar el uso de una determinada página.

Este análisis no se limita únicamente a la medición del tráfico, si no que puede utilizarse como una herramienta de negocio mediante la cual mejorar el *marketing* y el impacto de una web, además de aportar la información necesaria para realizar campañas publicitarias por zonas geográficas, variables demográficas, etc. Se trata entonces de establecer un perfil de usuario y comprender las tareas que realiza dentro de un sitio a través del *tracking*. Este *trackeo* puede hacerse de distintas maneras, muchas veces implementando todas ellas para obtener información más completa.

### 3.1.1. Geolocalización

A través de la dirección IP es posible conocer la localización del usuario en distintas granularidades, que van desde el país hasta la ciudad.

Esto es posible gracias a la propia naturaleza del protocolo TCP/IP [**?**], donde a cada dispositivo de acceso a Internet se le asigna una dirección única que identifica la conexión. Además, como se ha estudiado durante la carrera, esta dirección IP que identifica al usuario va adjunta en los datagramas de las peticiones generadas durante la navegación, quedando registrada como IPorigen de la conexión.

De esta forma, ya sabemos que nuestra dirección IP viaja continuamente en cada comunicación que establecemos en internet, pero ¿quién controla y registra la vinculación de una dirección con un espacio físico? Hay diversos organismos, que a través de la organización *Regional Internet Registry* (RIR) [**?**], se dedica a supervisar la asignación y registro de recursos de números de internet en cada región del mundo.

Tenemos así 5 RIR en funcionamiento, una por cada continente.

Gracias a estas organizaciones es posible saber la situación geográfica de cada dirección, conociendo además detalles de la conexión [**?**] como los servidores DNS utilizados, los puntos de acceso por los que ha pasado la conexión para realizar el *routing*, etc. A continuación se adjunta una captura de la información obtenida al consultar la IP en la propia plataforma del RIPE.

```
inetnum:        158.49.0.0 - 158.49.255.255
netname:        UNEX
descr:          Universidad de Extremadura
descr:          Extremadura
country:        ES
admin-c:        GSR1-RIPE
tech-c:         GB316
abuse-c:        RIAC2-RIPE
status:         LEGACY
remarks:        For information on "status:" attribute read
https://www.ripe.net/data-tools/db/faq/faq-status-values-legacy-resources
mnt-irt:        IRT-IRIS
remarks:        mail spam reports: iris@certsi.es
remarks:        security incidents: iris@certsi.es
notify:         iris-nic@rediris.es
mnt-by:         REDIRIS-NMC
created:        1970-01-01T00:00:00Z
last-modified:  2017-12-11T13:16:22Z
source:         RIPE
```

Figura 3.1: Datos obtenidos mediante RIPE

Se puede

### 3.1.2. Análisis de los gestos del ratón

Este tipo de captura y análisis está centrado en el movimiento que hace el usuario a lo largo de la pantallla y los clicks realizados con el ratón sobre los diferentes enlaces que componen la interfaz.

Normalmente, la información que se obtiene por este método es útil tanto para desarrolladores, diseñadores y redactores de una página a la hora de evaluar el rendimiento, diseño y conocer los contenidos más relevantes, respectivamente. Esta técnica es tremendamente útil puesto que permite conocer cómo se *mueven* los

usuarios dentro de un sitio, que secciones dentro de una página visitan más, incluso generar mapas de calor con las zonas por las que más se mueven los usuarios y detectar así puntos calientes a los que prestar más atención, así como fugas de interés, encontrando puntos ciegos, para proceder a un análisis de las causas y mejorar así el rendimiento de un portal.

Desde un punto de vista técnico, esta recolección de datos puede ocurrir en tiempo real o bien los datos serán almacenados para manipularlos después. A día de hoy muchas *suites* analíticas implementan esta funcionalidad a través de código Javascript insertado en las páginas con métodos que capturan y rastrean la interacción del ratón dentro de la página.

Este método a ido evolucionando con el tiempo hacia nuevas aproximaciones, como la predicción del click, donde mediante un programa que siga el movimiento del ojo del usuario y midiendo la dilatación de su pupila podemos predecir donde se realizará la pulsación y adaptar así la respuesta del servicio a la nueva situación prevista [**?**].

### 3.1.3. Cookies

Las cookies, aunque estén tan de moda últimamente con la salida del *nuevo* RGPD [**?**] (Reglamento General de Protección de Datos) sustituyendo a la nacional LOPD (Ley Oficial de Protección de Datos), surgieron en los albores de internet, siendo el primer método de *rastreo* utilizado.

Las cookies surgieron por la necesidad de mantener información sobre la sesión del usuario después de un login, mantener los elementos de un carrito de la compra o sencillamente para guardar información sobre preferencias del sitio (idioma, tema...).

Estas cookies consisten en un par clave-valor que, generado por el servidor, es

enviado al cliente para que este lo almacene en local. Más tarde esta información es consultada para adaptar el contenido en base a esa información.

En teoría, una cookie solo puede ser consultada por su origen, es decir, por el servidor que la generó, pero con el tiempo y la aparición de servidores compartidos (servidores de publicidad, por ejemplo) es posible generar una cookie que podrá ser consultada por todas las páginas que empleen ese servidor, estas son las llamadas *third party cookies*, que guardan o usan información del usuario sin que este tenga que navegar directamente en su dominio, dado que está incrustada en la web anfitrión.

Entre la información que es posible capturar con una cookie se incluye el sistema operativo, la dirección IP, el navegador, la ubicación, uso horario, idioma y lo más importante, los sitios por los que se navega. Por ejemplo, si se busca en una tienda online (pongamos Amazon como ejemplo), esa información sobre la búsqueda se queda almacenada en la cookie. Si otra web emplea publicidad de esa tienda, sabrá los términos de la búsqueda y podrá mostrar publicidad relacionada.

Normalmente, la técnica para el rastreo con cookies pasa por la creación de un identificador único de usuario para asociarlo a la misma. Este ID nos mantendrá identificados durante nuestra navegación y será empleado por las páginas que usen los servicios del proveedor de esa cookie. De esta manera, si cada página emplea cookies propias o de terceros, seremos conscientes de que estamos continuamente generando información y que además esta información no pasa desapercibida, si no que es registrada, enviada, usada y compartida entre compañías.

## 3.2. Google y la plataforma Android

Hasta ahora hemos visto los métodos de captura de información a través de la web, pero en los últimos 10 años el mercado ha cambiado mucho con la aparición del smartphone. Las cifras son apabullantes, en 2008 nace Android y en el tramo de 10

años la implantación del smartphone en la sociedad está fuera de cualquier duda.

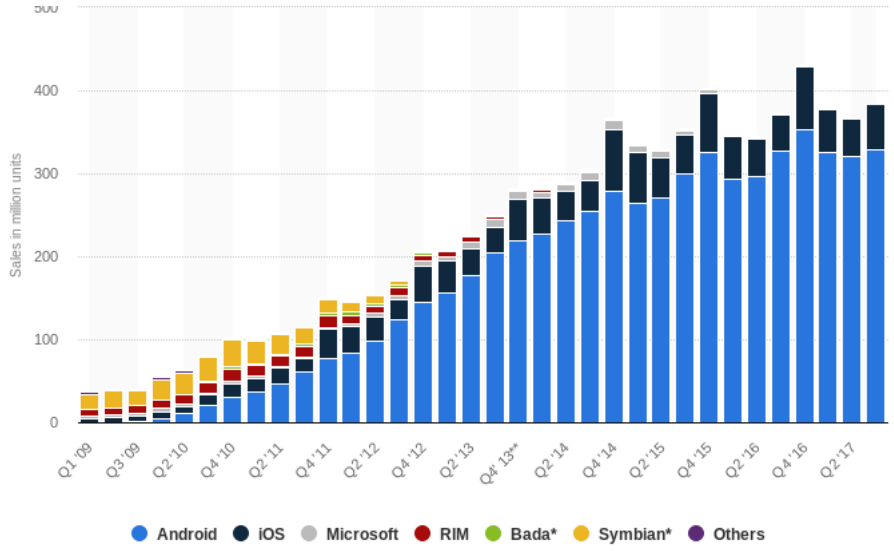


Figura 3.2: Venta global de smarthpones según el sistema operativo

Es por tanto una línea muy a tener en cuenta, estamos hablando que llevamos un dispositivo inteligente con nosotros todo el día, se ha convertido en una extensión más de nuestro cuerpo. Un pedazo de tecnología que nos permite estar conectados 24 horas al día, sin interrupción.

Pensar que aún teniendo eso en el bolsillo aún somos anónimos es pecar de insensatos, cuanta más conexión mayor control. Las grandes compañías, en su afán por conocer a sus usuarios, no desaprovechan este hecho, aunque a nosotros muchas veces se nos pase por alto. Llevamos en nuestro bolsillo un aparato que no deja de ser un ordenador miniaturizado, que nos permite comunicarnos con nuestros contactos, acceder a Internet y usar servicios que nos hacen la vida más fácil. Pero no se paga nada, a parte del coste del propio dispositivo físico.

Recordemos entonces las palabras citadas al inicio del trabajo, cuando no pagas por un producto es porque el producto eres tú. Consumimos tecnología mientras que la tecnología nos consume a nosotros, usamos teléfonos cuyo código fuente sólo

es accesible para los creadores del mismo, no sabemos que llevamos encima, en el bolsillo, un aparato que expande nuestras capacidades de comunicación como nunca antes ha sucedido en la historia de la humanidad.

Personalmente, creo que esto debería y que realmente está empezando a cambiar, si el teléfono es una parte más de nuestro cuerpo (algo incuestionable), deberíamos aplicar el mismo celo que al resto de cuestiones de nuestra vida. Del mismo modo que no consumimos determinados productos (por ejemplo, gente que evita los transgénicos, carnes rojas, etc) deberíamos comenzar a plantearnos el origen de ese *cacharro* que llevamos todo el día en nuestro bolsillo, con el que compartimos nuestra intimidad y usamos para hablar con amistades y seres queridos.

En el caso que nos atañe, Android, viene de la mano de Google, empresa que vive (entre otras cosas) del *segmentation-targeting*. Quieren nuestros datos y nosotros se los damos sin pestañear. Incluso siendo conscientes de que nuestras costumbres están siendo analizadas, perfiladas y almacenadas dentro de un modelo, no dejamos de consumir sus servicios.

Esa esclavitud tácita está con nosotros desde que abrazamos la tecnología, cada vez más conectados, cada vez más esclavos. Por ello, es bueno mirar con ojos críticos a las compañías a las que entregamos nuestras vidas (la vida digital y real están tan mezcladas que hacer distinciones es absurdo) y pararnos a pensar que sucede realmente con nosotros, los consumidores. ¿Realmente no estamos pagando nada por los servicios que se nos ofrecen? ¿Deberían pagarnos ellos a nosotros por invadir nuestra privacidad? ¿Asumimos el coste que conllevan las facilidades que este modelo de consumo tecnológico nos ha impuesto?.

Veamos cuánto de verdad hay en estas palabras y si realmente existen casos documentados de que esto esté sucediendo así.

### 3.2.1. Los servicios de Google y la localización

Durante el 2017 salió a la luz la noticia de que Google registraba y almacenaba la localización de los teléfonos Android incluso cuando este servicio estaba desactivado [**?**]. Esto fue confirmado poco después por Google, quien afirmó que abandonaron esa práctica al poco de darse a conocer estos hechos.

Es lógico pensar que cuando estamos usando algún servicio de google que incluya localización esta se mande a sus servidores, son sus reglas y si quieres usar el servicio es un coste asumido. De hecho, Google no oculta esta práctica, pudiendo consultar tu histórico de localizaciones en su propio portal [**?**]. A continuación se adjunta la imagen del histórico de localizaciones del autor en la ciudad de Cáceres.

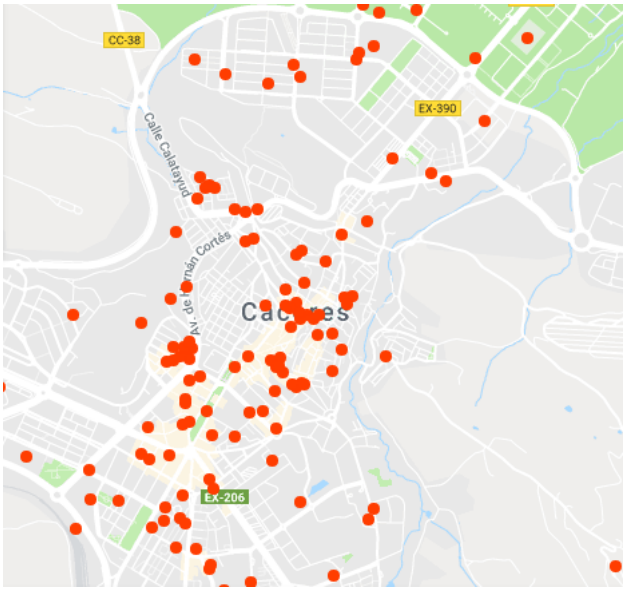


Figura 3.3: Historial de localizaciones en Cáceres

Por lo tanto, esta práctica de Google no es ningún secreto, siendo además una *feature* de su sistema y sus servicio de mapas, ofreciendo rutas alternativas al itinerario habitual en función del tráfico, guardar la posición del aparcamiento, etc.

El problema viene cuando se pierde el control de esta información. La noticia

citada al principio de esta sección pone en relieve la problemática, puesto que Google envía información de tu localización a sus servidores incluso cuando este servicio está desactivado en el teléfono y la SIM quitada.

Pero, ¿cómo sabe Google donde estás si tienes este servicio desactivado? Aprovecha las antenas de telefonía. En primer lugar se identifica al usuario por su CellID (identificador único de cada teléfono) y cada vez que el terminal se conecta a una torre se manda la información de esa conexión junto con el cellID a sus servidores. Esa información de la conexión incluye el identificador de la torre, que permite saber su localización exacta.

En un primer momento pudiera parecer que esta práctica no es demasiado intrusiva, solamente se manda información sobre la torre más cercana a la que se ha conectado el teléfono, pero cuando esta práctica se extiende a lo largo del tiempo (durante 24 horas por ejemplo), se puede llegar a triangular la posición exacta del teléfono gracias a la información de las torres a las que se ha ido conectando.

Google reconoció los hechos y aportó más información al respecto, afirmando que además de la información de las torres de telefonía empleaban otros datos, como la cercanía con redes Wifi u otros dispositivos.

Esto hila con otro escándalo que esta vez esta relacionado con el coche que Google emplea para mapear las ciudades para su servicio de Google Maps. Resulta que este coche, además de fotografiar las calles, iba recolectando la información de las redes Wifi abiertas, obteniendo así un mapeo de las redes y su localización exacta [**?**].

Con estas fuentes de datos es tremendamente sencillo para Google saber que está haciendo el usuario en cada momento, puesto que es posible determinar si estás en un núcleo urbano, en lo alto de una montaña, si vas andando, bicicleta, usas el

transporte público, etc. Esta información es tremendamente útil para hacer crecer la base de conocimiento de Google y mejorar el targeting de su servicio de publicidad (AdWords).

### 3.2.2. Aplicaciones de terceros

Hemos visto como Google aprovecha su propia plataforma para registrar datos del usuario, veamos ahora como otras compañías se aprovechan de ella para recolectar información de los usuarios a través de las aplicaciones que implementan para este mismo sistema.

#### Aplicación de La Liga

A principios de Junio de 2018 salió a la luz un nuevo caso de espionaje al usuario. En esta ocasión se trataba de la aplicación oficial de La Liga [**?**], la cual usaba el micrófono de los usuarios para, en teoría, detectar posibles fraudes.

Para ello la aplicación pedía permisos de micrófono y de localización para, de nuevo supuestamente, activar esta funcionalidad en las franjas horarias de los partidos de fútbol y analizando el audio capturado junto con la localización, detectar posibles fraudes.

El revuelo vino porque los usuarios no eran conscientes de que esto sucediese, lo cual dice muy poco de nosotros como usuarios, se estaban aceptando unas condiciones legales en las figuraba una línea indicando claramente este uso, así como la autorización explícita a la aplicación para usar el micrófono.

Al poco tiempo de que la noticia salía a la luz la empresa realizó un comunicado donde se repetían lo dicho arriba, que ese audio era únicamente para detectar fraudes en el territorio Español y no era usado para otros propósitos. Dada la vaguedad de las declaraciones, hubo usuarios que analizaron la aplicación y estudiaron su

comportamiento [**?**], lo que nos permite hacer un resumen de como está implementada esta lógica.

En primer lugar, la aplicación comprobaba los permisos de micrófono y localización en cada ejecución, solicitándolos en caso de no estar asignados. Después, mediante un listener, detecta cada cambio de ubicación y va creando un registro que, cada 30 minutos, se manda a un servidor y se borra del teléfono.

El micrófono, en cuanto está disponible, comienza a grabar y mandar los archivos al servidor. Durante cuanto tiempo permanece activa esa grabación es un misterio, así como el comportamiento de la activación del micrófono, puesto que lo lógico sería pensar que la aplicación, en función de la localización y el horario, active el micrófono, pero el análisis del código rebela que esta activación es arbitraria y nada tiene que ver con cuándo suceden los partidos de fútbol

Esto ha demostrado que la escucha a los usuarios por parte de las compañías es una práctica que está a la orden del día, de hecho, cada vez son más las voces que afirman que Facebook a través de su aplicación emplea el micrófono para escuchar conversaciones privadas y aunque Facebook ha negado siempre esta práctica, las sospechas siempre estarán ahí, quedando demostrado además, como acabamos de ver, que ésta práctica existe y lo preocupante realmente es saber que pasa con ese archivo de audio desde el momento en el que sale del teléfono, puesto que se pierde cualquier control sobre estos datos.

**Aplicación Linterna**

En este caso, vamos a irnos a una aplicación que saltó a la palestra a lo largo de 2013, cuando se descubrió que una aplicación a primera vista inocente dedicada a activar el led de la cámara y proveer así de una linterna, mandaba los datos de ubicación de sus usuarios a sus servidores junto con datos acerca de las redes wifi

disponibles para más tarde vender los datos a terceros.

Este caso está ampliamente documentado, llegando a publicar un comunicado de la Comisión Federal del Comercio de EEUU (un homólogo al patrio FACUA) donde se describen y condena esta práctica [**?**].

En el documento citado se condena a la compañía por estas acciones, entre las que se incluyen la posibilidad de tomar fotos a través de la cámara del dispositivo sin que el usuario tome partido en ello y el envío de datos de localización a empresas de publicidad.

Este caso nos sirve como muestra, junto con el anterior, de que estas prácticas llevan muchos años en el sector y pone en relieve la falta de control de los datos del usuario, además, nos hace plantearnos que hacen las grandes compañías con sus aplicaciones, en las que entra en juego mayor alcance de usuarios y por tanto mayor rédito económico.

## 3.3. Dispositivos IOT

En los últimos años hemos visto como *el internet de las cosas* ha comenzado a implantarse en los hogares, desde bombillas inteligentes que podemos controlar con nuestro teléfono hasta frigoríficos que nos avisan cuando falta algún producto en la nevera.

Esta evolución ha seguido su camino y las compañías no han querido perderse la fiesta, desarrollando cada una sus propias versiones de asistentes personales para, en teoría, facilitarnos la vida dentro de nuestra casa, es el caso del Google Home de Google, Alexa de Amazon o el Apple homePod de Apple. Estos dispositivos comparten todos la característica de activarse mediante comandos de voz, lo cual implica una escucha pasiva en segundo plano a la espera de reconocer algún comando

y activarse.

Esta tecnología, dado su enorme potencial, hizo que muchos usuarios mirasen con lupa a estos dispositivos, dando como resultado lo que veremos a continuación.

### 3.3.1. Asistentes personales

En este último año han irrumpido en el mercado los asistentes personales, introducidos por Google, Amazon y Apple. Estos asistentes son precisamente eso, un asistente personal que nos ayuda a gestionar nuestra agenda, consultar Internet e interactuar con el ecosistema de cada compañía.

Todos comparten la misma funcionalidad básica, incorporan un micrófono que permanece siempre activo esperando a recibir el comando de activación. Normalmente este comando es una palabra clave que permite que el dispositivo se active, Ok Google! en el caso del Google Home, Alexxa... en del Amazon Echo, etc. Una vez el dispositivo está activo pasa a un estado de escucha activa donde atiende las consultas del usuario a través de la voz.

No hay duda de que se tratan de aparatos tremendamente interesante y que a día de hoy es hacia donde se dirige el mercado, Meter en casa un dispositivo con inteligencia artificial que nos facilite la vida, podamos encender la televisión, reproducir música, hacer anotaciones, llamar a nuestros contactos, etc, gestionado únicamente por voz es un producto realmente atractivo.

Pero como hemos estado viendo hasta ahora, las compañías quieren nuestros datos, meter en nuestros hogares dispositivos inteligentes que se encuentran siempre escuchando en segundo plano, a estas alturas, no parece una decisión muy prudente.

Esto demuestra el afán de las compañías por descubrir más de sus usuarios, uno

de los grandes escollos que tienen hasta el momento es que sólo capturan la información cuando el usuario interactúa con sus servicios, pero permanecen a ciegas el resto del tiempo. Si consiguen meter en los hogares estos aparatos, que permiten escuchar conversaciones privadas en cualquier momento, aumentará su capacidad de recolección de datos y por tanto mejorar infinitamente su *targeting*, puesto que no sólo sabrán lo que hace el usuario cuando usa su aplicación, si no que les permitirá saber como se comporta durante su vida privada.

En el caso de Amazon Alexa ha quedado demostrado que el dispositivo realiza escuchas aleatorias [**?**], de duración indeterminada, que, hasta donde se ha podido averiguar, quedan almacenadas en el dispositivo. No es así en el caso del Google Home, donde todos los fragmentos de audio que se registren son enviados a sus servidores, de hecho lo dejan bien claro en su política de privacidad [**?**]

Esto demuestra el afán de estas empresas por conocer cada vez más detalles de sus usuarios, hasta el punto de perseguir el objetivo último de que todos los hogares cuenten con un dispositivo inteligente que les permita conocer los hábitos de sus usuarios en su vida privada.

Este afán queda explicado cuando reparamos en el problema que llevamos denotando durante el desarrollo de estas páginas, la información recogida, hasta ahora, es parcial, sólo nos muestra al usuario cuando usa sus productos, el resto del tiempo permanece invisible para ellos. Si finalmente consiguen su objetivo (un dispositivo en cada casa) conseguirán saltar esa brecha y poder obtener un perfil completo del usuario, sabiendo que dice, hace y cómo se comporta a lo largo de las 24 horas del día, sin interrupción, los 365 días del año.

Justo ese es el objetivo de este trabajo, lo cual nos indica que no vamos del todo mal encaminados. Cada vez es más importante conocer a nuestros usuarios,

pero hasta ahora este conocimiento estaba limitado al producto en sí, si conseguimos extraer un perfil de comportamiento completo del mismo tendremos en nuestras manos una información con un enorme potencial, que nos permitirá abrir nuevas líneas de investigación, las cuales, en buenas manos, podrán cambiar para bien la vida de muchas personas.

Nuestro objetivo final, y lo que se persigue con este proyecto, es que esta información no esté exclusivamente en manos de compañías que sólo quieren ver incrementados sus beneficios, si no que sea accesible para cualquier equipo de desarrollo que tenga buenas intenciones (en nuestro caso, el equipo del SpiLab de la Epcc).

# Capítulo 4

# Requisitos del proyecto

En esta capítulo se analiza el problema planteado por el proyecto y se detalla su descomposición como requisitos de un sistema software. No hay que perder de vista que nuestro objetivo final será monitorizar las aplicaciones del usuario y registrar información acerca de cómo éste interactúa con el medio.

## 4.1. Requisitos

Se propone el desarrollo de una aplicación móvil que, en segundo plano, registre los movimientos del usuario y la interacción que realiza con el teléfono. Este registro de actividad debe cubrir los requisitos listados a continuación para componer un sistema final que satisfaga los objetivos expuestos con anterioridad.

### 4.1.1. Requisitos funcionales

El sistema deberá:

- Registrar cada aplicación que lanza el usuario detectando y almacenando el nombre del paquete asociado a ella.
- Aportar un contexto temporal de cuando las aplicaciones son ejecutadas, esto es, el *timestamp* del momento de su ejecución.
- Incluyendo lo anterior, registrar el uso dado a aplicaciones relacionadas con, comunicación, mensajería instantánea, búsqueda web,redes sociales y por último, la información generada por el propio sistema.
- Dentro de comunicación se incluye la aplicación de Gmail, Mensajes (sms) y el Teléfono.
- Dentro mensajería instantánea prestaremos atención a Telegram, Whatsapp Messenger y Facebook Messenger.
- Como aplicaciones relacionadas con la búsqueda en la web estarán la Chrome Browser y Google Search.
- En las aplicaciones de redes sociales; Twitter y Facebook.
- Dentro del sistema, deberemos escuchar tanto las notificaciones recibidas como los sensores, incluyendo el GPS, acelerómetro, presión, giroscopio y temperatura.

**Aplicaciones de comunicación**

- En **Gmail** capturar, la dirección de correo del emisor.
- La dirección de correo del receptor(es).
- El asunto del correo electrónico.
- El contenido, cuerpo de dicho correo.
- En **Mensajes** se deberá capturar, el destinatario.
- El cuerpo del mensaje.
- En **Teléfono** se deberá capturar, el destinatario de la llamada.
- Tiempo total invertido en ella.

**Aplicaciones de mensajería instantánea**

- En **Telegram** capturar, el nombre del contacto o grupo interlocutor.
- Los mensajes enviados en la conversación por parte del usuario.
- En **Whatsapp** capturar, el nombre del contacto o grupo interlocutor.
- Los mensajes enviados en la conversación por parte del usuario.
- En **Facebook Messenger** capturar, el nombre del contacto o grupo interlocutor.
- Los mensajes enviados en la conversación por parte del usuario.

**Aplicaciones de búsqueda en la web**

- En **Chrome** capturar las páginas visitadas.
- El tiempo invertido en cada una de ellas.
- En **Firefox** capturar las páginas visitadas.

- El tiempo invertido en cada una de ellas.
- En **Búsqueda de Google** el objeto de búsqueda.

**Aplicaciones de redes sociales**

- En **Twitter** capturar, el nombre de los perfiles visitados.
- Los tuits publicados.
- El contenido de los mensajes directos.
- En **Facebook** capturar, el nombre de los perfiles visitados.
- Los comentarios hechos en dichos perfiles.
- Los estados compartidos dentro de la red social.
- Las subidas de fotos y su título.

**Sistema**

- Deberán registrarse las notificaciones recibidas por el usuario, incluyendo el contenido de la notificación y la marca horaria del momento en el que se recibe.
- Del sensor relacionado con la ubicación **GPS** deberemos registrar cada cambio de coordenadas que registre el teléfono.
- Mediante el **acelerómetro** registraremos los cambios en la aceleración del teléfono.
- Con el sensor de **presión** registraremos los cambios de altitud del teléfono.
- Con el **giroscopio** para detectar y registrar los movimientos que realiza el teléfonpo.
- El sensor de **temperatura** nos permitirá registrar, como no, la temperatura ambiente del dispositivo.

### 4.1.2. Requisitos no funcionales

- Versión. El sistema deberá soportar una versión limpia de Android 8.0.
- Eficiencia. Dado que el sistema debe registrar en tiempo real toda la información generada por el usuario, deberá ser capaz de capturarla en tiempo real, al mismo tiempo que se analiza y almacena.
- Privacidad. El usuario deberá autorizar explícitamente al software desarrollado permisos de accesibilidad, de modo que sea consciente de que existe una aplicación monitorizando parte de su actividad.
- Robustez. Debido a que el sistema debe estar constantemente capturando, el software debe ser tolerante a fallos y mantener el servicio siempre disponible.
- Seguridad. La base de datos local deberá estar protegida ante el acceso indebido a los datos.

Los requisitos listados en este capítulo tienen como objetivo definir la implementación de un sistema que permita, en última instancia, obtener un perfil completo del usuario.

Por un lado, los requisitos funcionales nos guiarán en la captura de información del usuario. Como se ha visto, tenemos 5 grandes grupos, que nos permiten conocer todas las facetas de la vida de este, ó, al menos, todas las que pasen por usar el terminal.

Por un lado la comunicación, escuchando aplicaciones como Gmail, el envío de sms y las llamadas telefónicas. Con la mensajería instantánea cubrimos las tres grandes aplicaciones en este campo, Whatsapp, Telegram y Facebook Messenger. En la navegación web, otra de las grandes tareas que hacemos con nuestros teléfonos, escucharemos Chrome, Firefox y la barra de Búsqueda Rápida de Google.

También podremos saber como interactúa el usuario con sus contactos a través de las redes sociales, puesto que almacenaremos datos provenientes de Facebook

y Twitter. Y por último, para conocer cómo interactúa el usuario con el medio, recogeremos las notificaciones que entran en el dispositivo, así como los datos de los sensores físicos que incorporan la mayoría de terminales.

Todos estos requisitos, junto con los no funcionales, cuyo objetivo es aportarle robustez al proyecto, nos permitirán desarrollar un sistema completo que abarque los usos mayoritarios del teléfono, obteniendo así un perfil completo de cómo un usuario interactúa y se relaciona con su terminal y entorno.

# Capítulo 5

# Análisis

Para centrar el problema y enfocar los requisitos a un proceso que nos lleve a obtener un proyecto de ingeniería, haremos uso de distintos diagramas que nos ayudaran a resolver el problema y plantear una solución. Así en el capítulo nos encontramos con los diagramas de casos de uso, indicando las tareas que debe resolver el sistema, y un diagrama de arquitectura a alto nivel que plantea una solución inicial.

## 5.1. Diagramas de Casos de Uso

Los casos de uso del sistema están planteados de modo que el usuario, al interactuar con el teléfono, dispara siempre la acción, aunque en muchos casos la acción del usuario será pasiva, es decir, no tendrá que realizar ninguna acción en concreto para disparar, por ejemplo, la captura de los sensores, pero será su propia actividad la que nos permita realizar dicha captura.

Por lo demás, hemos agrupado las aplicaciones en 5 categorías, comunicación, mensajería, navegación web, redes sociales y sensores. Dado que la mayoría de estas aplicaciones comparten la información a capturar, los veremos agrupados por comodidad para el lector.

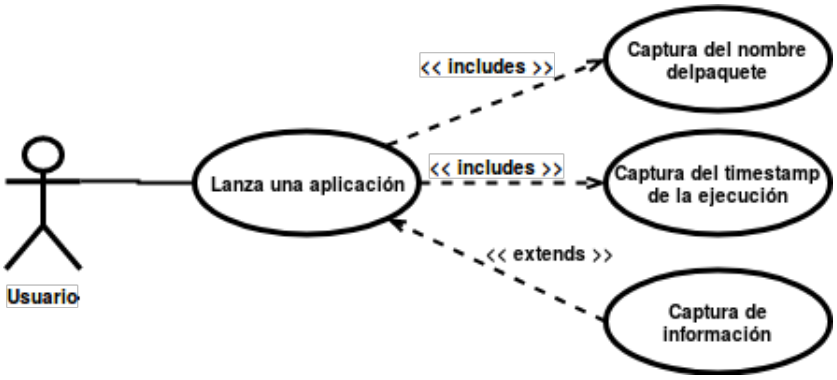


Figura 5.1: Casos de uso de cualquier aplicación.

Vemos como la captura de información extiende del lanzamiento de una aplicación, esto es así porque esta captura no se hará siempre, si no que dependerá de si la aplicación lanzada es objeto de captura. Si nos centramos en el apartado de

aplicaciones de comunicación;

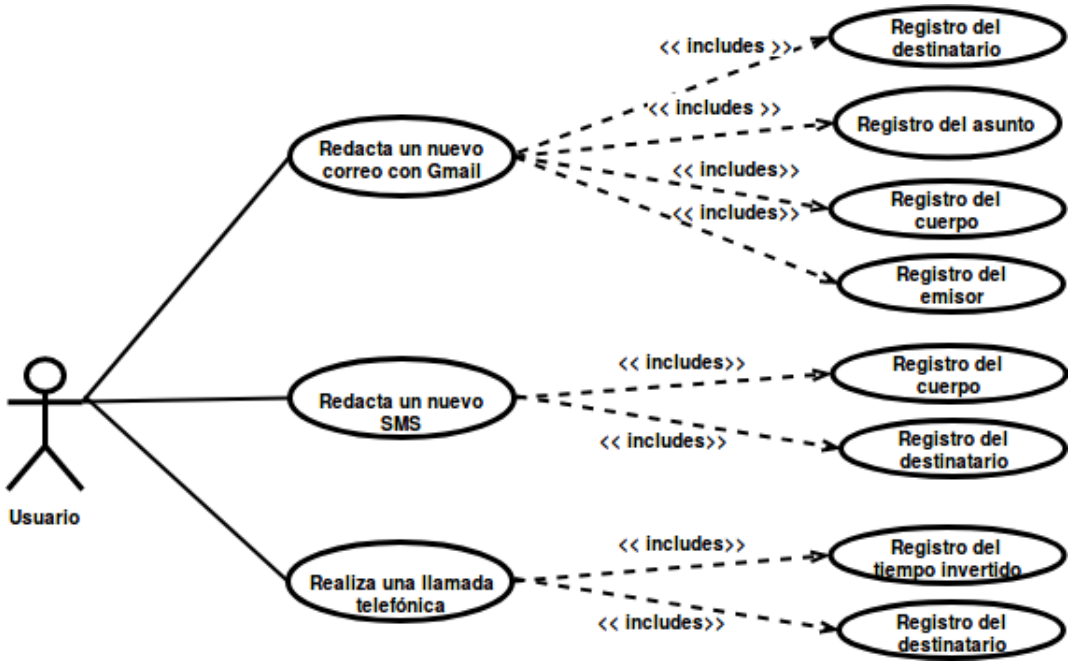


Figura 5.2: Casos de uso de aplicaciones de comunicación.

En el grupo de mensajería instantánea.

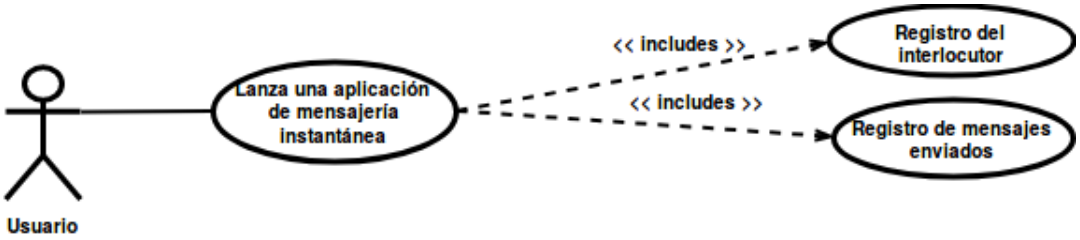


Figura 5.3: Casos de uso de aplicaciones de mensajería instantánea.

Aplicaciones de navegación y consulta de la web.

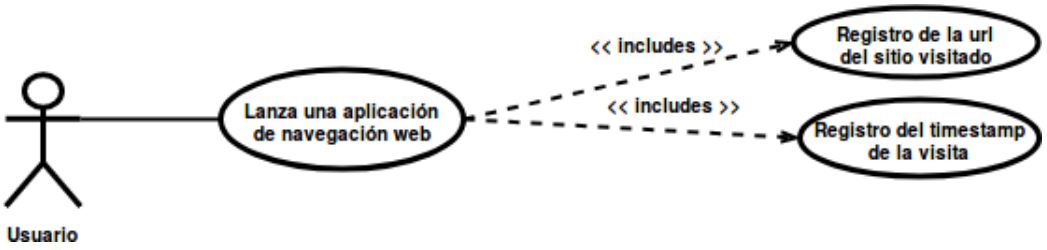


Figura 5.4: Casos de uso de aplicaciones de navegación web.

Si nos fijamos ahora en las aplicaciones de redes sociales, tenemos los siguientes casos de uso;

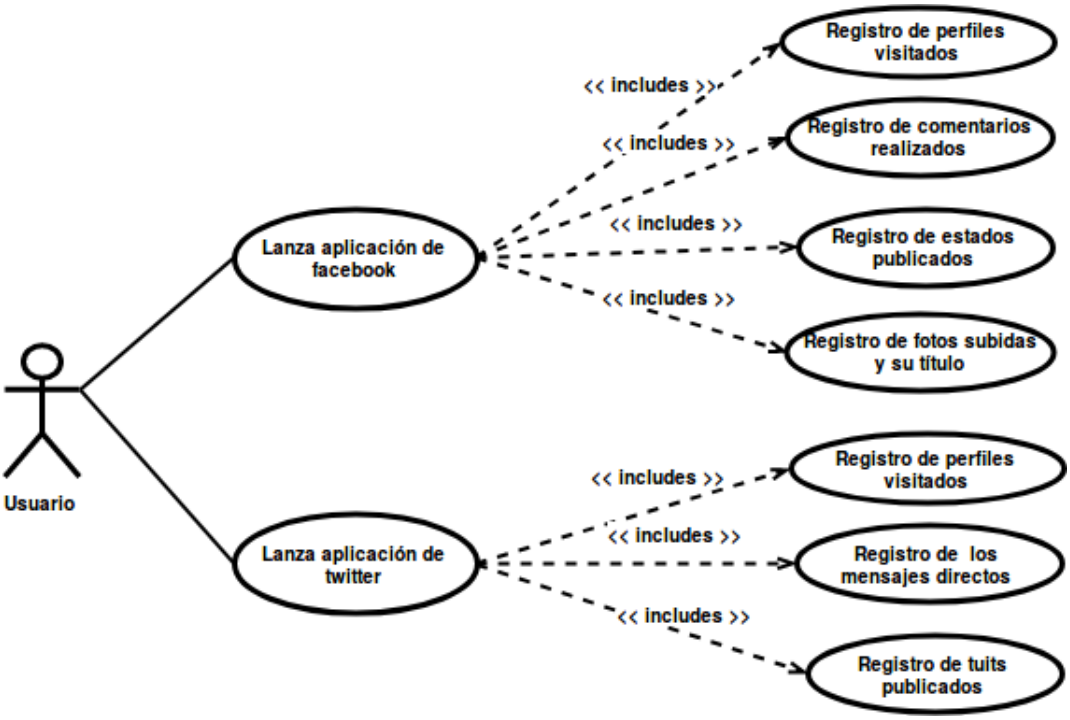


Figura 5.5: CU Apps RRSS.

Por último, en el apartado de la información del sistema;

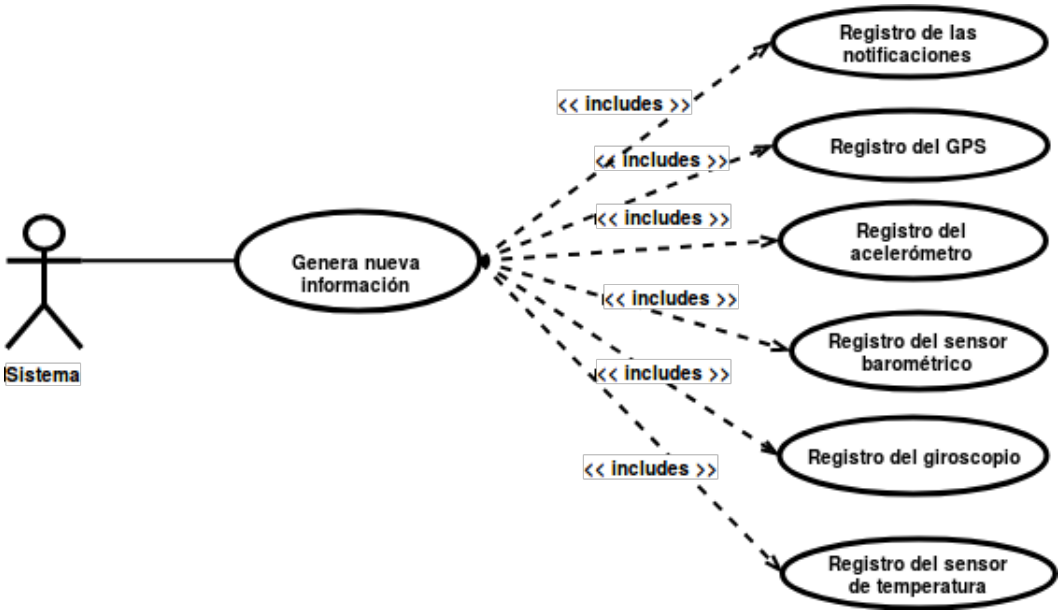


Figura 5.6: CU Sistema.

## 5.2. Arquitectura de alto nivel

Teniendo claro cual es el propósito de nuestro sistema, debemos plantearnos cuáles serán los elementos que, interactuando entre ellos, resuelvan el problema final.

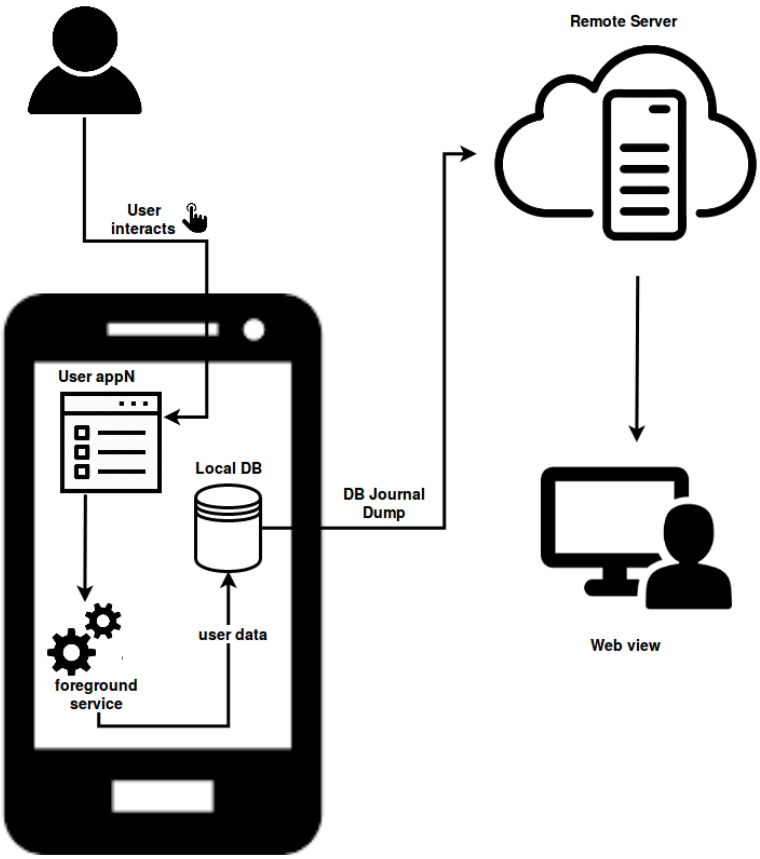


Figura 5.7: Arquitectura de alto nivel.

Vemos que el sistema se compone gracias a tres componentes fundamentales, por un lado el usuario, fuente de nuestros datos, el propio teléfono, en donde capturaremos la información, y un servidor en la nube que recibirá de forma periódica los datos recogidos por el teléfono y que permitirá además consultarlos.

### 5.2.1. Smartphone

Vayamos por partes pues y analicemos cada grupo de componentes. Por un lado el **smarthphone** y el **usuario**. Dado que el smartphone es la interfaz de acceso del usuario a las funciones del teléfono y sus aplicaciones, será entonces aquí donde centremos nuestro sistema y donde estarán la mayor parte de sus componentes, al menos todos los relacionados con la captura de información.

Este proceso de captura se detalla en el siguiente fragmento de la arquitectura. Toda la arquitectura parte del usuario, el cual interactúa con su teléfono mediante pulsaciones en la pantalla, que son recogidas por aplicaciones que le devuelven información al usuario mediante esa misma pantalla. Será entonces esta interfaz de usuario lo que nuestro sistema deberá escuchar y capturar. Por lo tanto tendremos una aplicación que recibirá las pulsaciones en los elementos de la interfaz del usuario (UI) bajo la forma de eventos.

Vamos a desglosarla por cada elemento, viendo antes de cada componente la unidad de datos que va a manejar.

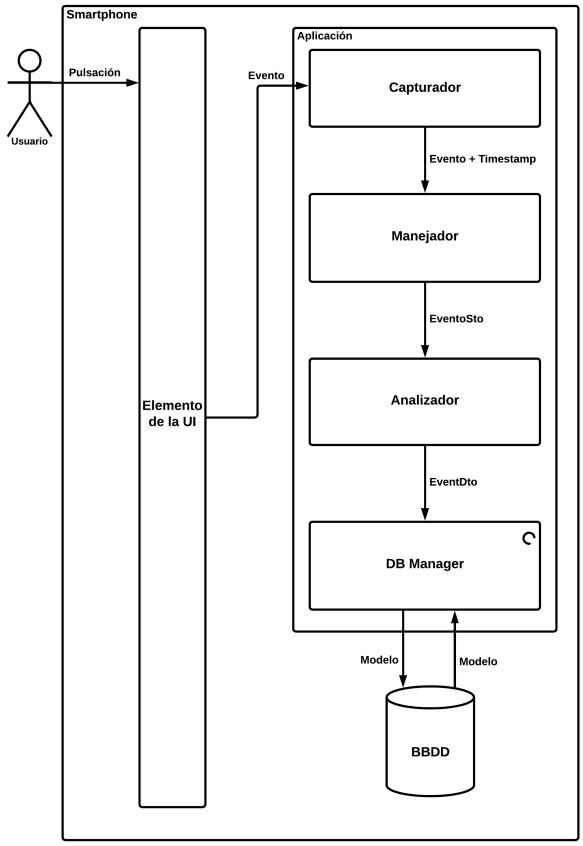


Figura 5.8: Arquitectura de alto nivel. Detalle de la parte móvil.

### Evento

Estamos ante la información de entrada única y por tanto fundamental del sistema. Se trata de una unidad de datos compleja y sin estructurar.

Los genera el sistema cada vez que ocurre un cambio notable en la pantalla, y su propósito es dar el mayor feedback posible del usuario.

Muchas veces esta información no surge en el instante en el que se produce el evento, que no deja de ser una instantánea de la pantalla en ese momento, si no que se necesita un contexto. Esa es la parte complicada del desarrollo, necesitamos desarrollar sendas máquinas de estado que nos ayuden a relacionar los eventos entre

ellos y procesarlos de tal modo que se produzca información útil y estructurada.

Es por tanto, una unidad de datos muy cruda y desestructurada a la que hay que extraer el significado en su procesamiento en las capas inferiores. Por tanto este evento será producido por el sistema y la fuente de entrada del nuestro, a través del **capturador**.

### Capturador

Se trata de un proceso que está siempre escuchando la aparición de nuevos eventos, con el objetivo de capturarlos y pasárselos al siguiente nivel de la arquitectura, el **manejador**. Su carga de trabajo realmente será poca, y consistirá en informar al manejador pasándole el evento junto al instante temporal en el que fué generado.

Por lo tanto el capturador recibe eventos y genera eventos junto a su marca temporal para que los consuma el siguiente nivel, manejador.

### Evento + timestamp

No consiste en una unidad de datos como tal, puesto que sólo consiste en una fecha y hora y el propio evento asociado a ellos.

### Manejador

Recibe un evento y su marca horaria y debe encapsularlo para que el nivel inferior lo analice, es por tanto el nexo de unión entre los datos en bruto y los datos refinados.

Para ello, cuando reciba el evento, en primer lugar lo encapsula junto a su marca horaria en un objeto que hemos llamado Sto. Simple Transfer Object.

Una vez encapsulado, consulta la aplicación fuente del evento, es decir, la aplicación que lo generó, que nos permite establecer nuestro primer filtro y discernir si se trata de

gmail, telegram, facebook... etc.

En función de esto, se llamará a una implementación del analizador u otra, puesto que cada aplicación sigue su propio flujo lógico. Por ello se le comunica un Sto con la información fundamental que necesita.

**Sto**

Es la unidad de datos empleado para comunicar el **manejador** con el **analizador**, se trata de una unidad de datos cuyo objetivo es encapsular la información en unidades contenidas y controladas. Será consumido por el analizador a través de la consulta de su evento y su marca horaria, siendo esta última fundamental para aportar el contexto necesario por el analizador.

**Analizador**

Recibe los citados Stos, objetos sencillos, del manejador, para tratarlos, procesarlos y extraer su significao.

Este procesamiento consisitirá en analizar, en base al modelo correspondiente, el patrón de llegada de los datos y en base a su contenido y el de eventos antecesores y sucesores, aportarle un contexto. Gracias a este modelo construiremos Dtos completos que representarán una unidad de información completa acerca del uso de la app.

**Dto**

Dtos, Data Transfer Objects, será la unidad última de información que genere nuestro sistema, representado así la información completa del usuario, es decir, uno de estos dtos contiene en su interior la interacción completa del usuario con cualquier servicio que le ofrezca el teléfono.

Será por tanto contenido semántico de esta interacción, siendo por ejemplo, un

dto una conversación mantenida por whatsapp con un contacto durante el intervalo de tiempo en el que estuvo activa, una búsqueda en el navegador... en resumen cualquier interacción, contextuada, del usuario con cualquiera de las apps que vamos a monitorizar.

Una vez estos Dtos están listos se les pasa al DB Manager.

**DB Manager**

Capa encargada de la persistencia de los datos que le pasa el analizador. El manager los convierte en modelos para su almacenamiento en una base de datos local en la memoria del teléfono.

En este manager recae una tarea también fundamental para el sistema, puesto que debe realizar un volcado periódico la base de datos a un servidor remoto para su seguridad y almacenamiento para posibles usos. Este funcionamiento de almacenar en local hasta que llegue el momento del volcado se ha tomado teniendo en mente la minimización del número de conexiones con el servicio externo. No podría ser una llamada a la API para persistir la información en la nube cada vez que una unidad de datos esté lista, puesto que la cantidad de información generada al día puede ser ingente.

La idea es almacenar en local hasta que pase un tiempo razonable (por ejemplo, 24 horas, tiempo en el que la base de datos se habrá ido llenando poco a poco) para volcar su contenido, liberar espacio y llevar los datos a la nube para persistir y asegurar la propia información. Así, no se satura la memoria interna del teléfono y la información viaja en bloques, minimizando la comunicación con el servicio externo y ahorrando batería, puesto que la conexión se realiza una vez al día.

### 5.2.2. Servidor Remoto

En este apartado analizaremos la arquitectura del entorno remoto planteado por la arquitectura. Consiste en un servicio en la nube que acepta el volcado que realiza la base de datos del teléfono cada poco tiempo y permite además visualizar los datos allí vertidos. Estamos, por lo tanto, ante una típica arquitectura de servicios web, donde

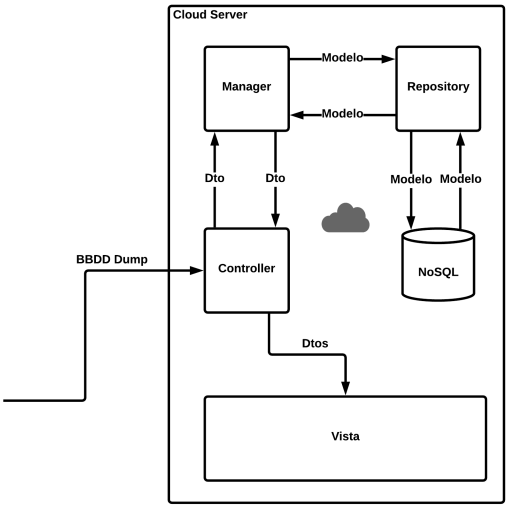


Figura 5.9: Arquitectura de alto nivel. Detalle del entorno cloud

tenemos un controlador, encargado de atender los puntos de entrada de la API, un Manager que realiza las tareas de conversión de objetos y la lógica de negocio, el Repositorio, encargado trabajar contra la base de datos, una base de datos NoSQL y una Vista, que nos permite visualizar los datos almacenados.

#### Controller

Proporciona la interfaz de entrada de la API a través de solicitudes HTTP, así, contaremos con un único método de entrada que aceptará datos en formato JSON, con el contenido en bloques de la base de datos del teléfono, así como los endpoints necesarios para la vista.

Este controller extraerá los datos del json y los convertirá a Dtos, pasándoselos a su vez al manager para que realice la comunicación con las capas inferiores.

**Manager**

Se encarga de transformar los dtos aportados por el controller en modelos que acepte la base de datos, así como realizar la operación inversa y convertir los datos de la base de datos en Dtos de forma que puedan ser invocados por el controlador y ser devueltos a la vista.

Será por tanto el elemento del entorno cloud donde se depositará toda la lógica de negocio, transformaciones de objetos, mezclados, etc.

**Repository**

Capa más próxima a la base de datos, nos ofrecerá un sistema de comunicación del resto de la plataforma con la base de datos, siendo este su único punto de entrada, donde manejaremos modelos acordes al esquema de los documentos almacenados en la DB NoSQL.

**DB NoSQL**

Se trata de nuestra base de datos en la nube, se opta por un sistema NoSQL debido al gran volumen de datos que vamos a manejar. Con el crecimiento de estos datos, surge la necesidad de proporcionar información procesada a partir de grandes volúmenes de datos que tienen unas estructuras horizontales más o menos similares, y este es justo el escenario donde el uso de NoSQL nos brinda la mejor solución.

**Vista**

Punto de salida de nuestra plataforma cloud, proporcionando así una interfaz de consulta de los datos almacenados, generados por la aplicación.

## 5.3. Diagrama de secuencia

Con el fin de explicar cuál es el flujo que recorren los datos y las operaciones que realiza cada componente así como los mensajes que se intercambian entre ellos, se incluye a continuación un diagrama de secuencia que describe el ciclo de vida de un evento desde el instante en el que se dispara hasta el momento en el que se almacena su información.

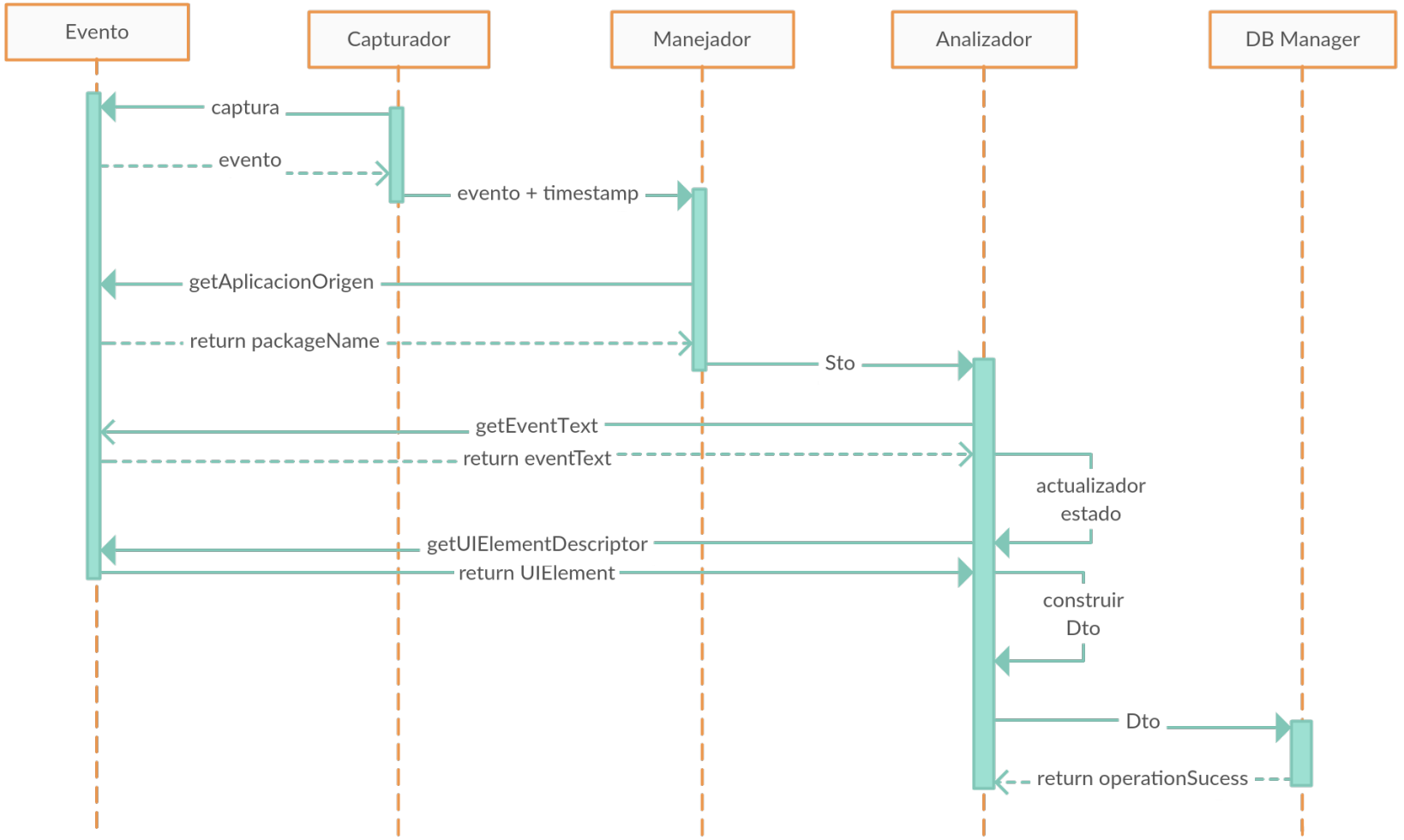


Figura 5.10: Diagrama de secuencia habitual del sistema.

De esta forma, planteamos una arquitectura inicial dividida en capas, donde cada una está pensada para realizar una tarea simple y atómica.

Se ha ideado esta arquitectura con la idea en mente de que sea un proyecto fácilmente ampliable, es decir, que el esfuerzo que conlleve añadir una nueva funcionalidad (por ejemplo, añadir una nueva app a la batería de aplicaciones a escuchar) tenga un impacto mínimo en el código y no implique grandes esfuerzos ni comunicaciones.

Así, si estamos persiguiendo la idea de recolectar un determinado número de aplicaciones lo suficientemente grande como para poder conformar un perfil de usuario completo, veraz y objetivo, para añadir una aplicación al sistema de escucha únicamente deberemos implementar su analizador, puesto que el resto de componentes lo único que hacen es filtrar y conformar la información para esta capa.

Con esta arquitectura conseguimos entonces el objetivo de que añadir aplicaciones para recolectar sus datos sea relativamente sencillo y lo único que hay que hacer es añadir código, sin tener que modificar la implementación existente.

# Capítulo 6

# Diseño

En este capítulo se detalla el resultado de las sucesivas iteraciones sobre la etapa de diseño. A partir de los requisitos se ha estudiado el problema y planteado una arquitectura que nos permitirá guiar la implementación de los casos de uso. Se pretende obtener una arquitectura limpia y organizada de manera que permita la mantenibilidad del proyecto y una modificación mínima y eficaz.

## 6.1. Arquitectura

A la hora de plantear nuestra arquitectura el primer paso será conocer la propia arquitectura de Android, por ello, en la siguiente sección estudiaremos como está planteado el sistema sobre el que nos basamos y en base a ese conocimiento tomaremos unas decisiones de diseño que nos ayudarán a componer la arquitectura final.

### 6.1.1. Decisiones de diseño

Si nos fijamos en la siguiente figura vemos como Android se asienta en una modificación del Kernel de Linux sobre el que se van abstrayendo capas en las que cada una emplea los servicios que provee la capa inmediatamente inferior.

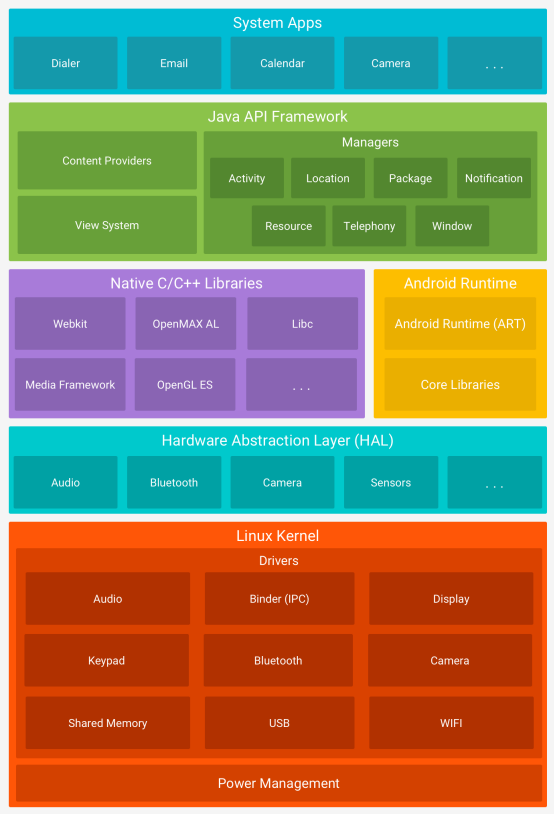


Figura 6.1: Arquitectura por capas de Android.

Así, como base de la arquitectura tenemos una modificación del Kernel de Linux

sobre la cual se implementa una segunda capa, de abstracción del hardware, que será la encargada de manejar y comunicarse con los periféricos del teléfono. Entendemos por periféricos lo habitual en estos entornos; sensores, antenas, emisores radio...). Sobre estas dos capas, que permiten una interfaz de acceso al dispositivo Hardware, tenemos las librerías de bajo nivel y el entorno de ejecución de android.

Si avanzamos un poco más vemos que en las dos últimas capas están relacionadas con una API Java que permite al programador implementar aplicaciones, usando a través de la api, todo la anteriormente expuesto.


Figura 6.2: Márgenes del sistema

Estas dos últimas capas serán en las que se situará nuestra arquitectura. Además será conceptualmente parecida puesto que como hemos visto, Android dispone de un sistema organizado por capas, algo bastante lógico y habitual en los sistemas software, y nuestro objetivo por tanto debe ser separar funcionalidades y agruparlas bajo este patrón.

Teniendo este concepto claro, nos fijaremos en la publicación de Robert C. Martin, Clean Architecture, en la que se plantea el concepto de Arquitectura Limpia,

sin ser más que una serie de condiciones que debe cumplir una arquitectura para que se la considere ”clean”. Nos encontramos entonces con una serie de reglas, cuyo cumplimiento ayuda a diferenciar y dividir el software en capas, obteniendo además un software independiente de elementos externos (como la ui, frameworks y bases de datos), testeable y mantenible.

En esta publicación nos encontramos con que una arquitectura debe partir de las entidades (entities). Estas entidades no son mas que las implementaciones sencillas de clases Java.

Estas clases permitirán instanciar objetos que reprensentarán a los actores principales de la lógica de negocio. Por tanto POJOS (Plain Old Java Object) y Dtos (Data Transfer Objects) se verán incluidos en esta capa.

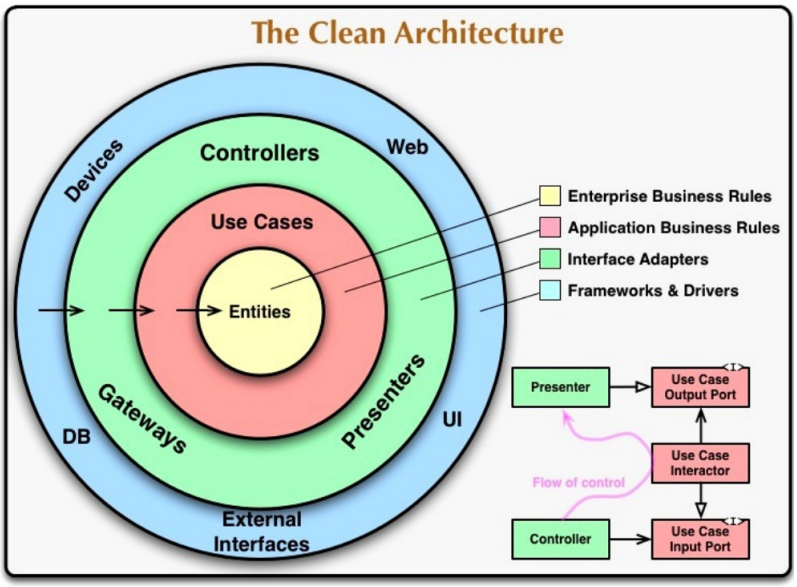


Figura 6.3: Arquitectura clean

Los casos de uso (use cases) están muy relacionados con las entidades, puesto que implementarán la lógica de negocio del sistema, orquestando y orientando el flujo de datos del sistema.

La capa de los Interface Adapters realiza la conversión de los datos de manera que se puedan comunicar los niveles superiores con los casos de uso y entidades (en MVC corresponde a los Controladores, MVVP al presenter... etc).

En la última capa y más externa, Frameworks and Drivers, residen las plataformas y herramientas externas, donde se incluye la interfaz de usuario, web...

### 6.1.2. Arquitectura a bajo nivel

Una vez tenemos claro como está estructurado Android y los servicios que ofrece al desarrollador, debemos plantearnos cómo diseñar una aplicación que sea capaz de implementar estos servicios de accesibilidad para capturar, procesar y extraer la información de los eventos que se generan, deberemos abstraernos del problema y ser capaces de, a partir de una vista general del sistema inferir las capas que finalmente nos guiarán la implementación.

Así pues, en primer lugar situar nuestro sistema dentro del stack tecnológico de Android citado al principio de esta sección.

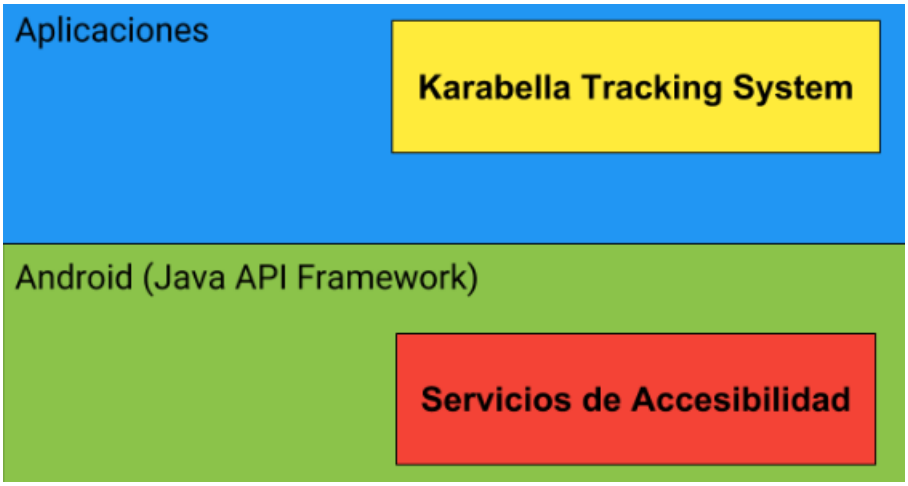


Figura 6.4: Arquitectura del sistema dentro del stack de Android.

Es natural esta clasificación puesto que el proyecto se basa en la implementación de los Servicios de Accesibilidad que Android dispone en de su API, gracias a mecanismos como la herencia y la implementación de interfaces.

Estos servicios de accesibilidad son la solución de Google para asistir y dar soportes a usuarios con diversidad funcional. Su naturaleza y comportamiento es lo que nos permite aprovechar estos servicios para obtener información de lo que hace el

usuario.

Se comporta además como cualquier otro servicio de Android, (es decir, procesos ideados para correr en segundo plano con un ciclo de vida prolongado en el tiempo). El Servicio de Accesibilidad se comporta como listener de Eventos de Accesibilidad, que lanzados por el sistema, son recogidos por la clase que implemente este servicio, donde son capturados y tratados como mejor convenga. Estos eventos serán, entonces, la unidad de información más básica que manejará nuestro sistema, pero a la vez la más compleja, dado que llevan dentro toda la información sobre la cual se construyen el resto de capas.

Los eventos llevan consigo información sobre la interfaz de usuario, por ejemplo los eventos se lanzan cuando entra en foco un elemento (con la descripción del elemento, botones, entradas de texto, nombres de ventanas...). Nuestra solución pasa por extraer la información de esos eventos y procesarla.

Dado que estos eventos pueden ser capturados por Eventos de Accesibilidad, nuestro sistema contará con un servicio que lo implemente y que realiza esta captura, este será un elemento fundamental en la arquitectura puesto que será la base sobre la que se asentarán el resto de elementos.

De esta forma, y siguiendo las líneas definidas, obtenemos una arquitectura detallada en el siguiente diagrama.

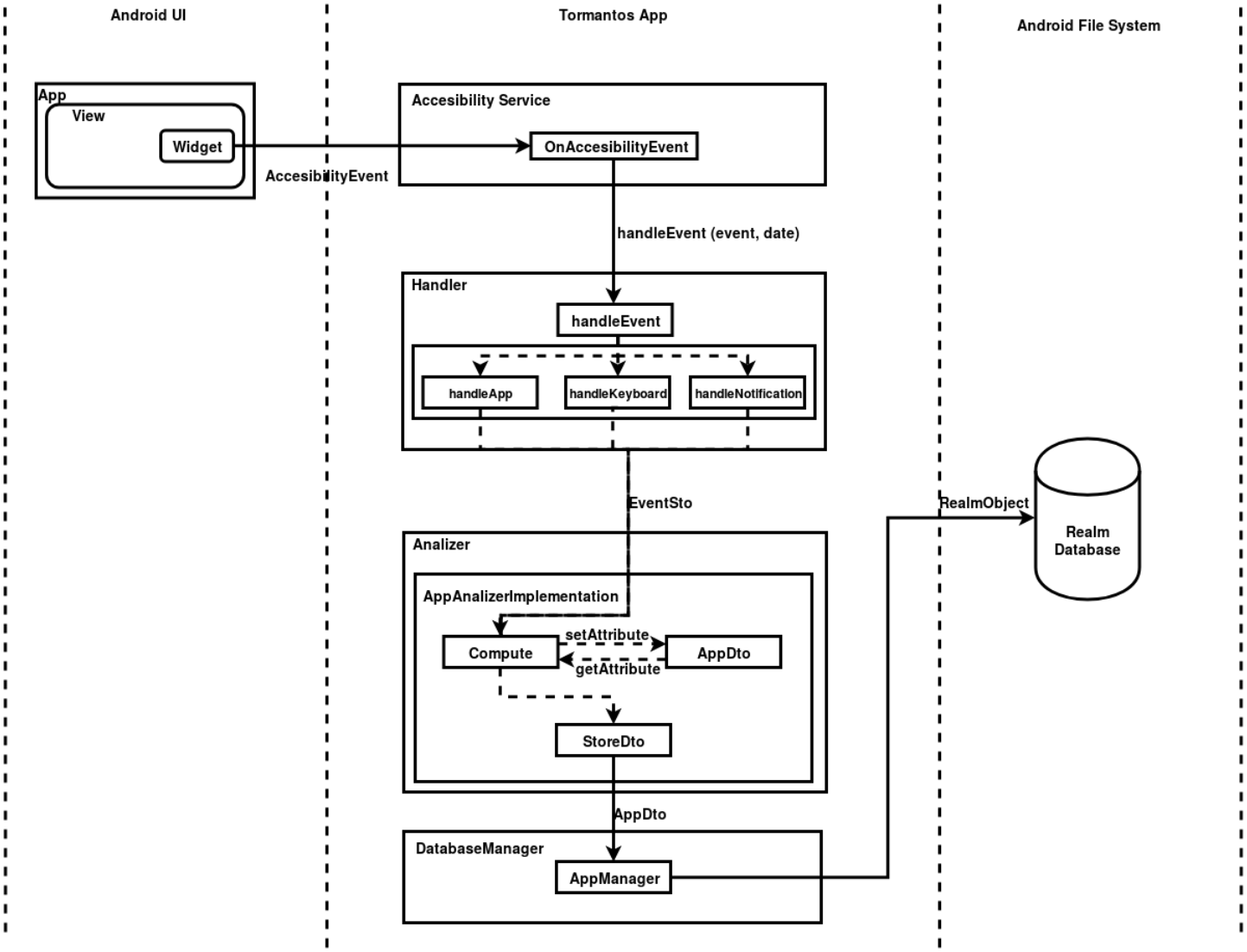


Figura 6.5: Arquitectura general del sistema

Donde la interfaz de usuario (UI) genera eventos de accesibilidad. Estos eventos, como ya se ha comentado, están asociados a cada acción que el usuario realiza con el teléfono sobre la interfaz. Será la implementación del evento de accesibilidad (Accesibility Service) el que nos permita capturarlos.

Este servicio se encargará de capturar los eventos y mandarlos hacia el Manejador de Eventos (Event Handler), este manejador será el encargado de clasificar y procesar los eventos en función de la información que traigan consigo.

Una vez que estos eventos pasen por el manejador, obtendremos como resultado una entidad lista para ser guardada en la base de datos.

Profundicemos un poco más en como sería este manejador y como son los componentes que lo forman.

Como estamos viendo, el servicio de accesibilidad (lo llamaremos simplemente servicio) recibe eventos de la interfaz y los envía al manejador, pero dado que los eventos se generan en gran cantidad y en momentos puntuales tendremos que manejar un caudal de información importante, la comunicación entre el servicio y el manejador se realizará a través de un bus que permitirá al servicio desatender los datos lo antes posible y desentenderse así de lo que ocurre con esos datos y como son tratados.

Este bus lo estará escuchando el manejador, que capturará los elementos para realizar un primer chequeo, y es que en función del nombre de la aplicación (más concretamente su package name), invocará a un *Scraper* encargado de extraer la información relevante del evento que estamos manejando.

En nuestro sistema tendremos una interfaz Analizer, con tantas implementaciones como aplicaciones estemos monitorizando. Procesarán el evento de accesibilidad y

lo encapsularán en una unidad de datos que será pasada a la capa de persistencia, formada por los Managers. Encargados de persistir en una base de datos local los datos generados por el sistema.

Al final nuestra arquitectura a bajo nivel resulta en el siguiente diagrama.

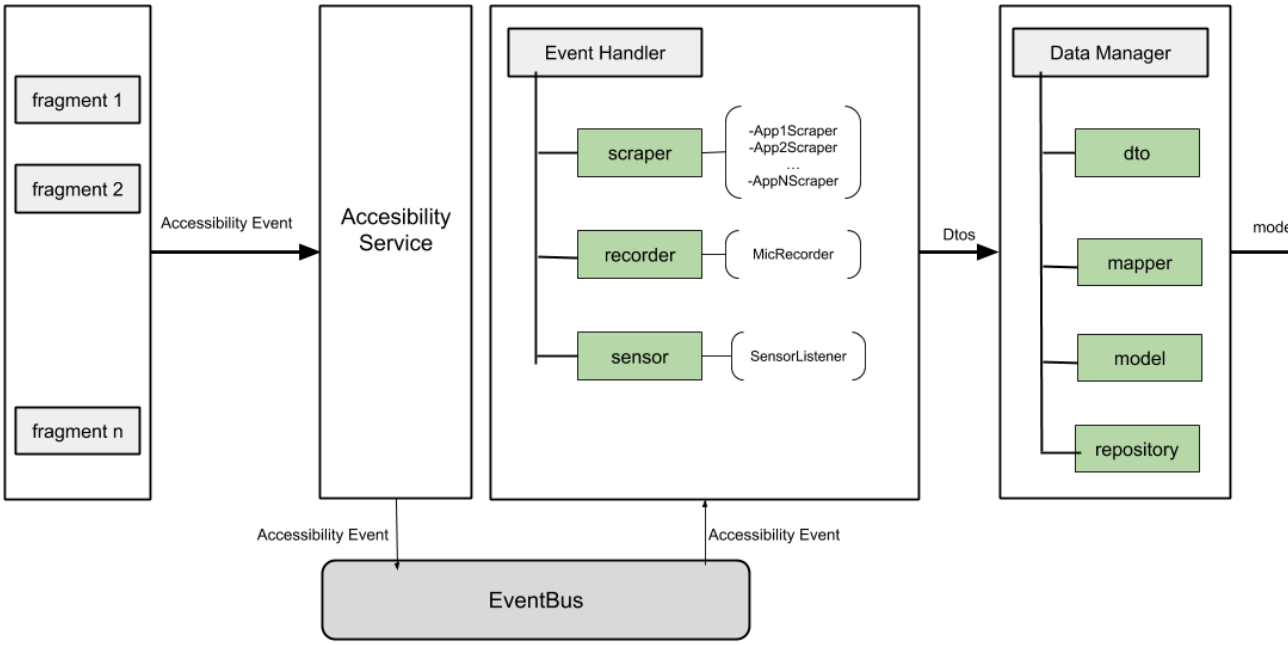


Figura 6.6: Arquitectura detallada

En primer lugar nos encontramos la ui que alimenta al servicio de accesibilidad con eventos. Este servicio se encarga de meterlos en un bus de forma que el manejador pueda recogerlos y computarlos.

Este tratamiento que realiza el manejador no es otro que, en función de la app que lo lanzó llamar a las clases implicadas en su tratamiento.

Expliquemos más despacio en que consite esto. El manejador, como se ha dicho en alguna ocasión, analiza el package name del evento, este package name nos permite identificar el elemento del sistema android que lanzó el evento, ya sean relacionados

con la aplicación en foco en ese momento como los relacionados con el uso de sensores del teléfono.

Los eventos de las aplicaciones serán tratados por el analizador que le corresponda. Estos extraerán la información pertinente y la encapsularán en dtos.

Con los sensores la historia es diferente, puesto que la escucha de los mismos está ligada a la implementación de sendos listeners que recogen la información que van generando. En un apartado diferente entra la escucha del micrófono, puesto que para respetar la privacidad del usuario sólo se lanzará la captura de audio cuando un evento relacionado con la misma sea lanzado. Nos referimos a casos concretos del uso del móvil como comandos de voz, notas de audio, etc.

En la parte del Data Manager nos encontramos a las entidades y a los responsables de mover esas entidades a lo largo de la arquitectura.

Así nos encontramos con el paquete repository, encargado de la persistencia en base de datos de las entidades. Entidades como dtos (para la comunicación de información desde el manejador al data manager) y mappers, que realizarán la conversión de dtos a modelos, entidad aceptada por el repository para la persistencia en la base de datos.

De esta manera nos encontramos con 4 capas separadas por funcionalidad y con independencia lógica entre ellas, puesto que cada una está consagrada a una tarea particular (recibir eventos, computarlos, persistirlos).

La lógica de negocio del sistema se encuentra, sin duda, en la capa correspondiente al manejador, donde deberemos de echar un esfuerzo de implementación considerable para manejar todo el flujo de información.

# Capítulo 7

# Implementación y desarrollo

Durante el siguiente capítulo veremos documentada la lógica que sigue la aplicación, centrándonos en el servicio de escucha en segundo plano, responsable de capturar los datos del usuario en cada una de las aplicaciones que nos permitirán conocer su perfil completo en cada una de sus facetas. Recordemos, comunicación, mensajería, navegación, redes sociales y el plano físico mediante los sensores.

En primer lugar, debemos conocer los objetos y los datos con los que estamos trabajando. A lo largo del proyecto hemos citado en innumerables ocasiones los eventos de accesibilidad, pero el lector aún no conoce su forma, su contenido ni cómo se generan. También se adjuntará una descripción de los objetos Dto donde almacenaremos la información, con el fin de situar al lector dentro del marco en el que se ha realizado la implementación del proyecto.

## 7.1. Accessibility Events

```
D/Helper: [type] TYPE_VIEW_TEXT_CHANGED
[eventId] 16 [class] android.widget.EditText
[package] com.whatsapp [time] 06/07/2018 19:51:25
[text] Hola! Lector

D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class]
[package] com.google.android.inputmethod.latin
[time] 06/07/2018 19:51:25
[text] Se han rechazado las alternativas

D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class]
[package] com.google.android.inputmethod.latin
[time] 06/07/2018 19:51:25
[text] Mostrando teclado español (España) (QWERTY (Ñ))

D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class]
[package] com.google.android.inputmethod.latin
[time] 06/07/2018 19:51:30
[text] El teclado español (España) (QWERTY (Ñ)) está oculto

D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class] com.google.android.launcher.GEL
[package] com.google.android.googlequicksearchbox
 [time] 06/07/2018 19:51:32
 [text] Aplicaciones
```

```
D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class] com.google.android.launcher.GEL
[package] com.google.android.googlequicksearchbox
[time] 06/07/2018 19:51:34
[text] Pantalla de inicio 1 de 1

D/Helper: [type] 16384
[eventId] 16384 [class] android.view.ViewGroup
[package] com.google.android.googlequicksearchbox
[time] 06/07/2018 19:51:35
[text] Pantalla de inicio 1 de 1

D/Helper: [type] TYPE_WINDOW_STATE_CHANGED
[eventId] 32 [class] android.widget.FrameLayout
[package] com.android.systemui
[time] 06/07/2018 19:51:38
[text] Pantalla de bloqueo.
```

El output listado arriba se corresponde con la acción de enviar una cadena de texto a un contacto de Whatsapp y, mediante el botón de *back*, navegar hacia la pantalla principal de la app, llegar a la caja de aplicaciones y volver a la pantalla principal. Una vez ahí se ha bloqueado el dispositivo.

La gran complicación del proyecto consiste en la enorme cantidad de información que generan acciones que tenemos muy automatizadas como usuarios. Es nuestro objetivo entonces consiste en poner orden a ese carro de datos.

Para ello, durante el proceso de implementación, cada vez que nos enfrentamos a una nueva captura, debemos debugguear los servicios generados (con outputs

similares al de arriba) y estudiarla hasta detectar el patrón que se corresponde a la acción que queremos capturar (por ejemplo, la escritura en Whatsapp o una búsqueda Web).

Una vez detectado este patrón, se debe implementar el analizador. En primer lugar, esta implementación consiste en imponer filtros y condicionales que descarten aquellos eventos que no son de nuestro interés. De esta manera, a la capa del analizador llegan únicamente los eventos asociados a la aplicación que queremos escuchar en ese momento.

Una vez hecho esto, se debe implementar el método *compute* del analizador correspondiente. Sendos condicionales consiguen que en función del elemento de la interfaz donde fué originado el evento, consigamos leer la información del ellos y asociarla a los atributos del Dto donde guardaremos la información.

De esta manera, detectando patrones, filtrando eventos y leyendo su información, conseguiremos imbuir información a los objetos dtos, los cuales usaremos para persistir el fruto de nuestra captura.

En el proyecto se incluyen capturas de varias aplicaciones, veamos a continuación tres de ellas que, por su relevancia, consideramos interesantes estudiar en detalle.

## 7.2. Captura de Gmail

En este apartado, y los que siguen, veremos los objetos destinados a guardar la información y los diagramas de actividad de tres aplicaciones clave para el perfilado del usuario, Gmail, Whatsapp y el navegador Chrome. Tienen como objetivo clarificar y documentar el flujo lógico que seguirá cada aplicación, durante el proceso de análisis y extracción del contenido de cada de las aplicaciones propuestas.

### 7.2.1. GmailDto

El objeto donde guardaremos la información de captura se compone de los siguiente atributos;

```
public class GmailDto extends RealmObject {
  @PrimaryKey
    private String id;

    /** The mail sender */
    private String sender;

    /** The mail receivers */
    private RealmList<String> receivers;

    /** Mail subject */
    private String subject;

    /** Mail content */
    private String body;

    /** When the mail was sended */
    private Date timestamp;
```

```
  // -- getters and setters
}
```

En primer lugar nos encontramos un identificador. Esto es necesario en tanto el cuanto el objeto herede de RealmObject. Este RealmObject es propio del framework Realm, una base de datos orientada a objetos que destaca por su rapidez. La veremos más adelante en la sección de librerías de terceros.

Empezando ya con los atributos relevantes para la captura, por un lado tenemos un atributo **sender**, destinado a almacenar el emisor del correo. Se podría pensar que es obvio que el que manda un correo es el usuario del teléfono, pero hoy en día es habitual que una persona tenga varias cuentas de correo, por ello almacenaremos en este atributo cuál de ellas elige.

El atributo **receivers** consiste en una lista (del framework Realm) de Strings. El motivo de que este atributo sea una lista y no un String sin más es sencillo, y es que un correo puede tener varios destinatarios. Almacenaremos como elementos independientes del array cada dirección de correo que el usuario escribe.

El atributo **subject** tiene como destino almacenar el asunto del correo. Lo mismo ocurre con **body**, cuya aplicabilidad no es otra que guardar el cuerpo del mensaje en sí.

Por último no encontramos con un atributo **timestamp** del tipo java.util.Date. Su propósito es aportar un contexto temporal sobre el momento del día en el que el usuario se dedica a mandar correos electrónicos.

### 7.2.2. Diagrama de actividad

En la siguiente figura se documenta el proceso que lleva al registro de un correo mandado por el usuario a través de la aplicación de GMail. Se ha organizado el

diagrama por niveles, desde la capa más externa de la aplicación hasta la capa de persistencia, así se puede observar tanto el camino recorrido por los datos como el procesamiento de la información en cada nivel.

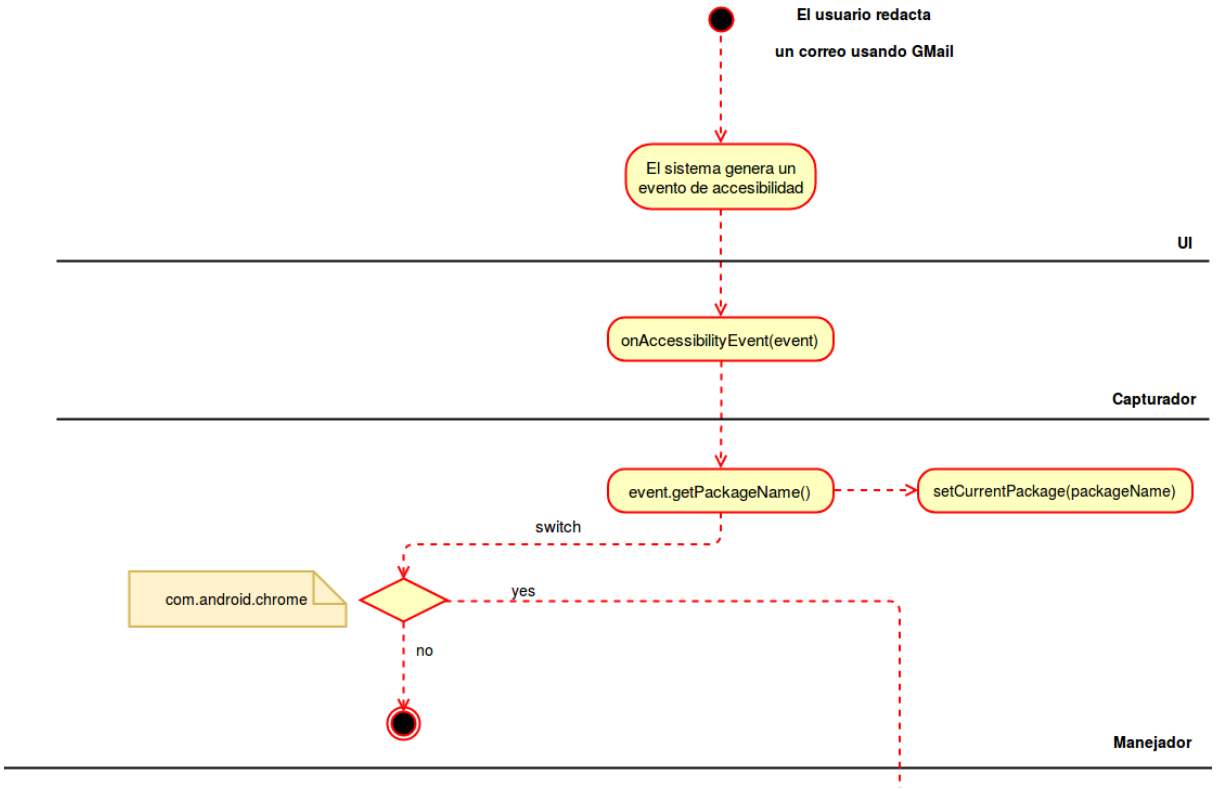


Figura 7.1: Diagrama de actividad de captura de un correo redactado con GMail (1).

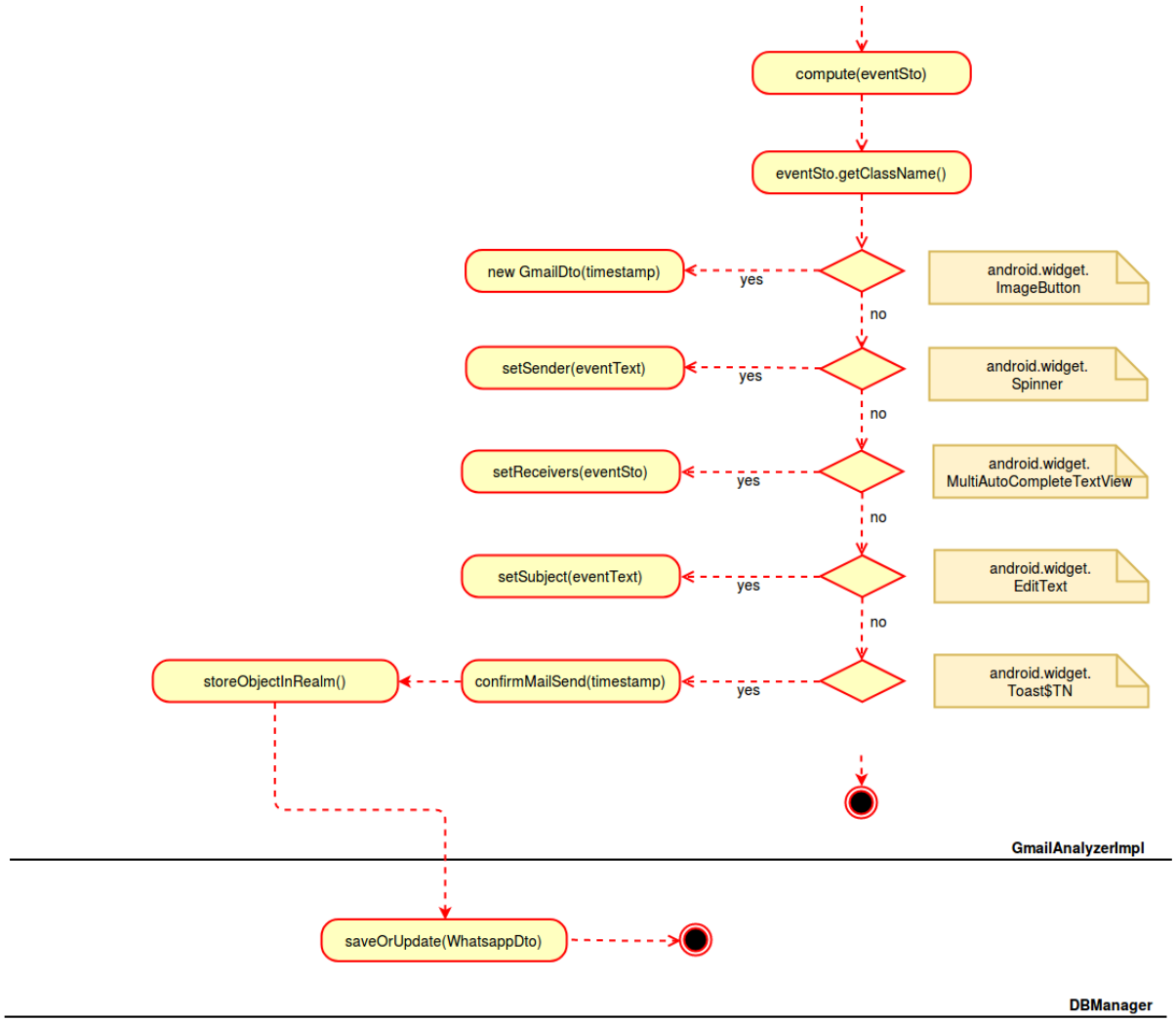


Figura 7.2: Diagrama de actividad de captura de un correo redactado con GMail (2).

Así, podemos desglosar el proceso en los siguientes puntos, explicando que se hace en cada componente.

- **UI**. En primer lugar, el usuario inicia la actividad al lanzar la aplicación de GMail e interactuar con su interfaz. Esta acción provoca que se lance un evento de accesibilidad por cada elemento gráfico que compone la pantalla.

- **Capturador**. Actúa como un listener de eventos de accesibilidad, de esta manera está a la escucha de todos los eventos producidos por el sistema, los cuales recoge y junto a la marca horaria del instante de la captura, se lo transfiere al manejador.

- **Manejador**. Comprueba el nombre del paquete que trae consigo el evento, asociado a la aplicación que le dió lugar. En este caso, si el nombre del paquete de la aplicación coincide con el de GMail, *com.google.android.gm*, invoca al analizador de gmail y le delega el evento.

- **Analizador**. Este es nivel con más carga, puesto que debe comprobar cada evento recibido buscando los componentes de la interfaz conocidos, es decir, aquellos en los que sabemos que se vierte la información que compone el correo. Estos elementos son;

  - *Widget Image Button*. Es el botón de editar que figura en la esquina inferior derecha de la interfaz.

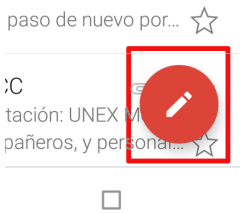


Figura 7.3: Android Image Button
Elemento botón que lanza la creación de un nuevo correo

Al pulsarlo el usuario, el sistema lanza una nueva pantalla en la que redactará el correo. Con lo cual, la pulsación de este elemento de la UI nos inicia el proceso de captura poniendo a nuestro servicio un contenedor,

donde compondremos la información del correo.

Cada vez que se pulse este elemento se instanciará un nuevo GmailDto, con el que se trabajará durante el proceso de captura.

- *Widget Spinner*. Se trata del campo donde aparece el emisor, es decir, el correo del usuario.

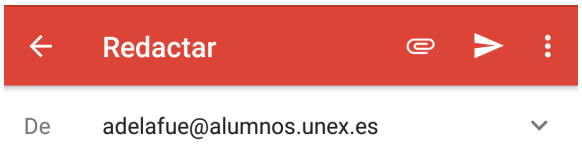


Figura 7.4: Android Widget Spinner

Elemento gráfico donde aparece el emisor.

- *Widget MultiAutoCompleteTextView*. Elemento de la interfaz donde aparecen los receptores, la potencia del componente permite que con las primeras letras del correo del receptor se sugiera la dirección completa.

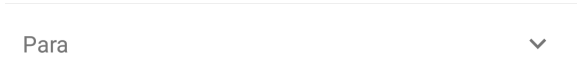


Figura 7.5: Android Widget MultiAutoCompleteTextView

Elemento gráfico donde aparece el receptor/es del correo.

- *Widget EditText*. Campo de texto donde escribir el asunto del correo.

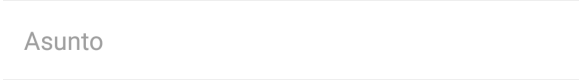


Figura 7.6: Widget Edit Text

Elemento de la interfaz donde escribir el asunto.

- *View*. Vista editable a modo de caja de texto donde se redacta el cuerpo del correo.

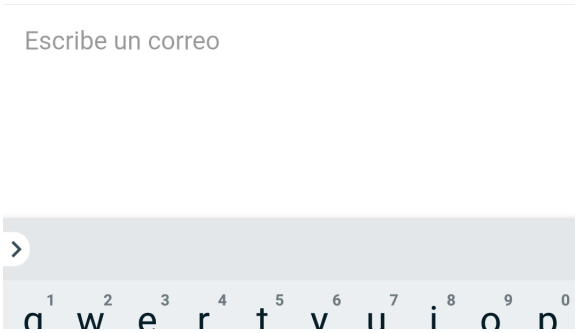


Figura 7.7: Android View

Elemento donde redactar el cuerpo.

Cada uno de estos elementos descritos sobre la interfaz, generan un evento de accesibilidad que el analizador comprueba a través del nombre de la clase del elemento de la interfaz asociado al evento. Si el nombre coincide con alguno de los elementos aquí listados, se extrae el texto del evento y se infla en el GMailDto el atributo correspondiente (emisor, receptor, asunto y cuerpo). Por último, tenemos el elemento *Widget Toast*, típico mensaje que aparece en la mitad inferior de la pantalla que informa de algún suceso.

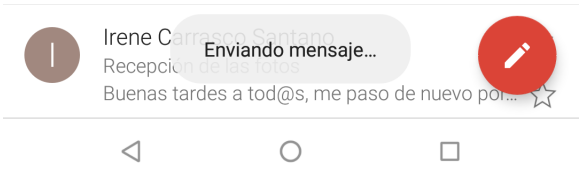


Figura 7.8: Android View

Elemento donde redactar el cuerpo.

En este caso el widget toast nos informa cada vez que se envía un correo, momento en el cual cerramos nuestro Dto y se lo pasamos al manejador de la base de datos para que lo persiste en la colección correspondiente.

- **DBManager**. Capa encargada de la persistencia, se encarga mapear los GMailDto a un modelo y almacenarlo en la base de datos local.

## 7.3. Captura de Whatsapp

Veamos ahora como es la captura de Whatsapp y cómo es la estructura del objeto en el que se asienta.

### 7.3.1. WhatsappDto

Los atributos que componente este objeto son;

```
public class WhatsappDto extends RealmObject {
@PrimaryKey
    private String id;

    /** Name of the contact who the user is talking to*/
    private String interlocutor;

    /** A list of the messages sent, including the timestamp of the send */
    private RealmList<TimestampString> textList;

    /** Describes the timestamp when the user clicked a conversation */
    private Date startTimestamp;

    /** Describes the timestamp when the user abandoned a conversation */
    private Date endTimestamp;

    // -- getters and setters
}
```

En primer lugar, el consabido identificador, necesario para la base de datos.

A este identificado le sigue el atributo **interlocutor**, destinado a almacenar el nombre del contacto con quien el usuario está hablando.

A continuación figura una RealmList con objetos del tipo TimestampString.

```
public class TimestampString extends RealmObject {

    /** String with the content of the text */
    private String text;

    /** Timestamp describing the exact moment when the text was written */
    private Date timestamp;

   // -- getters and setters
}
```

Este tipo de objeto consiste en un String, para almacenar la cadena enviada, y un timestamp para controlar el momento exacto del envío de la misma.

Por último, en WhatsappDto, nos encontramos con dos java.util.Date, **startTimestamp**, para almacenar el momento en el que el usuario abre una conversación, y en **endTimestamp** para, como sospecharán los lectores más hábiles, almacenar el momento exacto en el que esa misma conversación es abandonada.

### 7.3.2. Diagrama de actividad

En la captura de los mensajes enviados con Whatsapp entran en juego dos tipos de eventos. Por un lado, aquellos que pertenecen (ó cuyo nombre del paquete) corresponde con el propio Whatsapp, *com.whatsapp*, y por el otro, los relacionados con el teclado, *com.google.android.inputmethod.latin*.

Jugando con estos dos eventos seremos capaces de recoger con quién habla el usuario, lo que escribe y durante cuanto tiempo permanece en una conversación o en la propia pantalla principal de la aplicación. Veámoslo por partes.

**Eventos de com.whatsapp**

Este tipo de evento, cuyo origen es la propia aplicación de Whatsapp, nos aportará información acerca de los gestos que realiza el usuario dentro de la misma. Gracias a esto, podremos añadir un contexto a la información recogida, puesto que nos permiten obtener datos como el nombre del contacto con el que habla el usuario, a la vez que nos aportan información acerca de la pantalla que está viendo el usuario a cada momento.

A la hora de recoger el texto vertido por el usuario en una conversación determinada la captura se complica, puesto que entran en juego eventos externos, sin los cuales no podríamos dotar de sentido a las cadenas escritas. Veámoslo en detalle, dado que no es un asunto trivial. Expliquemos en primer lugar como se capturan los caracteres escritos.

Cuando escribimos un mensaje en el campo de texto de la aplicación, se generan tantos eventos de accesibilidad como letras pulsadas en el teclado. Estos eventos son originados por Whatsapp, con lo cual seguirán el flujo destinado para su captura, y diferenciando el widget de la UI asociado al mismo, podremos capturar el texto del evento, donde se encuentra nuestra preciada información, las letras que el usuario pulsa.

Pero al mismo tiempo, contamos con un inconveniente, y es que esta generación de contenido es *dummy*, es decir, únicamente recibimos información a chorro, sin saber cuando se envía una cadena, cuando se borra o cuando un mensaje es descartado navegando hacia atrás en la pantalla.

Para darle sentido a este flujo continuo de información deberemos atender los eventos que genera el teclado.

**Eventos de com.google.android.inputmethod.latin**

Estos eventos nos aportan la información de lo que hace el usuario con el teclado a través del texto que muestra a modo de *placeholder*. De esta forma, cuando el teclado se muestra por primera vez o el campo está vacío, el texto asociado al evento generado será *Mostrando teclado español (España) (QWERTY (Ñ))* (en el caso de usar un teclado con la distribución en español).

Pero no es tan sencillo como aparentemente parece, puesto el teclado genera eventos de este tipo cada vez que el usuario hace gestos inesperados, como por ejemplo mandar un emoji, buscar un gif o simplemente cambiar al teclado de símbolos para acceder a los números o símbolos de puntuación.

Manteniendo esto en mente, podemos saber cuándo el usuario acaba de enviar una cadena de texto, dado que cada vez que el campo de texto se queda vacío (recordemos el funcionamiento de la app, cuando enviamos un mensaje se vacía el campo), significa que el usuario ha mandado una cadena. Cada vez que un evento con estas características es generado, se notificará al analizador de WhatsApp mediante la llamada al método *confirmKeyboardInput*, el cual verificará el estado del Dto que contiene la información capturada hasta el momento.

Esta verificación del estado del objeto consiste en chequear la correcta inicialización de las variables, que no se ha capturado una cadena vacía ó que el texto capturado no se corresponde con el *placeholder* que incrusta Whatsapp en su aplicación, que reza *Escribir mensaje*.

Una vez pasado este primer chequeo, se comprueba el array donde almacenamos las cadenas escritas junto a su timestamp. Dado que los eventos, dependiendo de los gestos del usuario, pueden venir duplicados, (por ejemplo cambiar el layout del teléfono, abrir y cerrar el teclado, etc), deberemos comprobar que, en caso de que el

array no esté vacío, la cadena a añadir no exista en el array.

La captura del resto de campos (interlocutor y pantalla principal) es sencilla, como hemos dicho, para esto sólo atenderemos a los eventos generados por Whatsapp. Una vez establecido este primer filtro, deberemos tratar los eventos en función del elemento de la interfaz asociado al mismo, ó dicho en términos de implementación, el *className* del origen del evento. De esta forma tenemos:

- *com.whatsapp.HomeActivity*. Se corresponde a la pantalla principal del Whatsapp. Nos sirve para almacenar en la base de datos cualquier objeto existente que contenga información. Una vez hecho esto, instanciamos una nueva versión del mismo y limpiamos los atributos de la clase relacionados con el interlocutor y el mensaje que escribe el usuario.

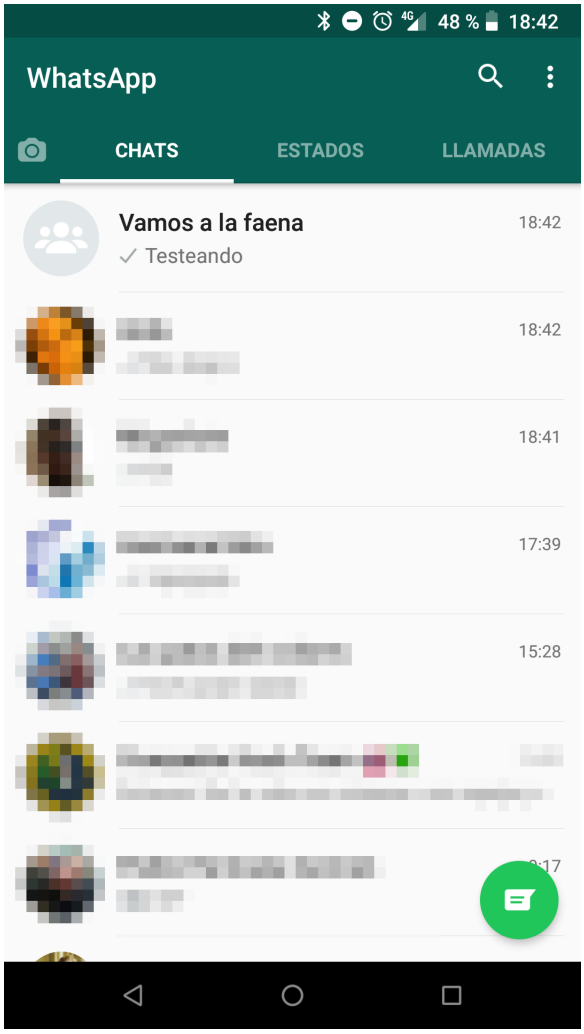


Figura 7.9: Whatsapp Home

Pantalla principal de Whatsapp

- *android.widget.RelativeLayout*. Por si sólo, un relative layout es simplemente una patrón para disponer los elementos en la pantalla, pero si su origen es Whatsapp (recordemos, el nombre del paquete, un filtro que en este momento ya se ha superado), sabremos que el usuario ha pulsado sobre una conversación.

  Además, el mismo evento que nos indica la entrada en una conversación lleva asociado el nombre del contacto, con lo cual para saber con quién está hablando el usuario bastará con extraer el texto del evento.


Figura 7.10: Whatsapp layout de una conversación

Layout de una conversación de usuario

- *android.widget.EditText*. Como es obvio, este evento estará asociado al campo de texto donde el usuario escribe su conversación. Deberemos, por tanto, capturar en todo momento el texto que llevan asociados este tipo de eventos, puesto que ahí estará el contenido real de lo que el usuario está escribiendo a su contacto.

  Este contenido se irá almacenando en una variable que se añadirá al Dto asociado cuando entre en juego un evento de teclado. Con las condiciones

descritas anteriormente, recordemos, eventos con nombre de paquete *com.google.android.inputmethod.latin* y cuyo texto implique la acción de mostrar el teclado limpio, esperando una nueva entrada de texto.

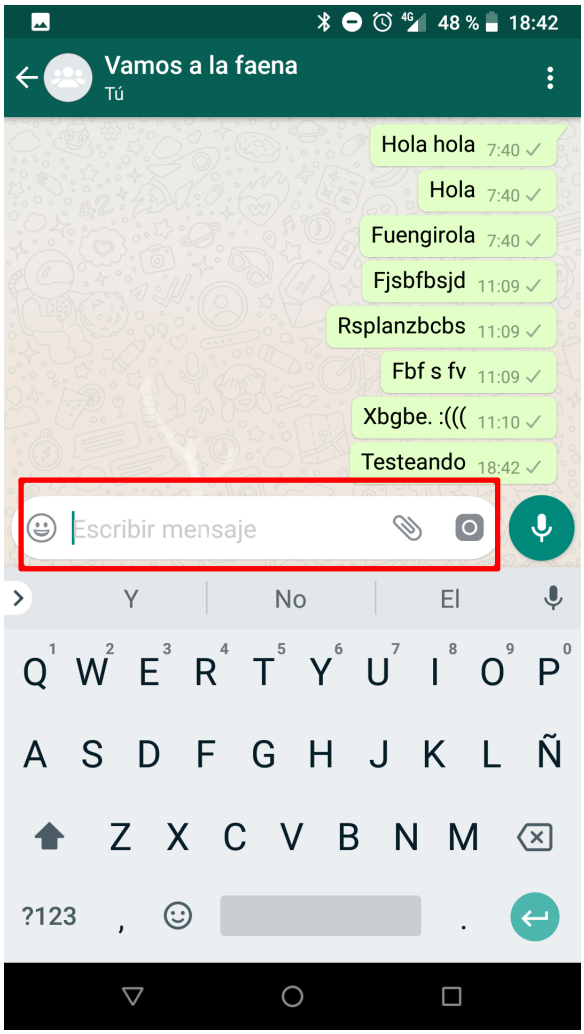


Figura 7.11: Campo de texto de Whatsapp

Campo de texto de Whatsapp

Veamos ahora, mediante un diagrama de actividad, una visualización gráfica del algoritmo de captura descrito anteriormente.

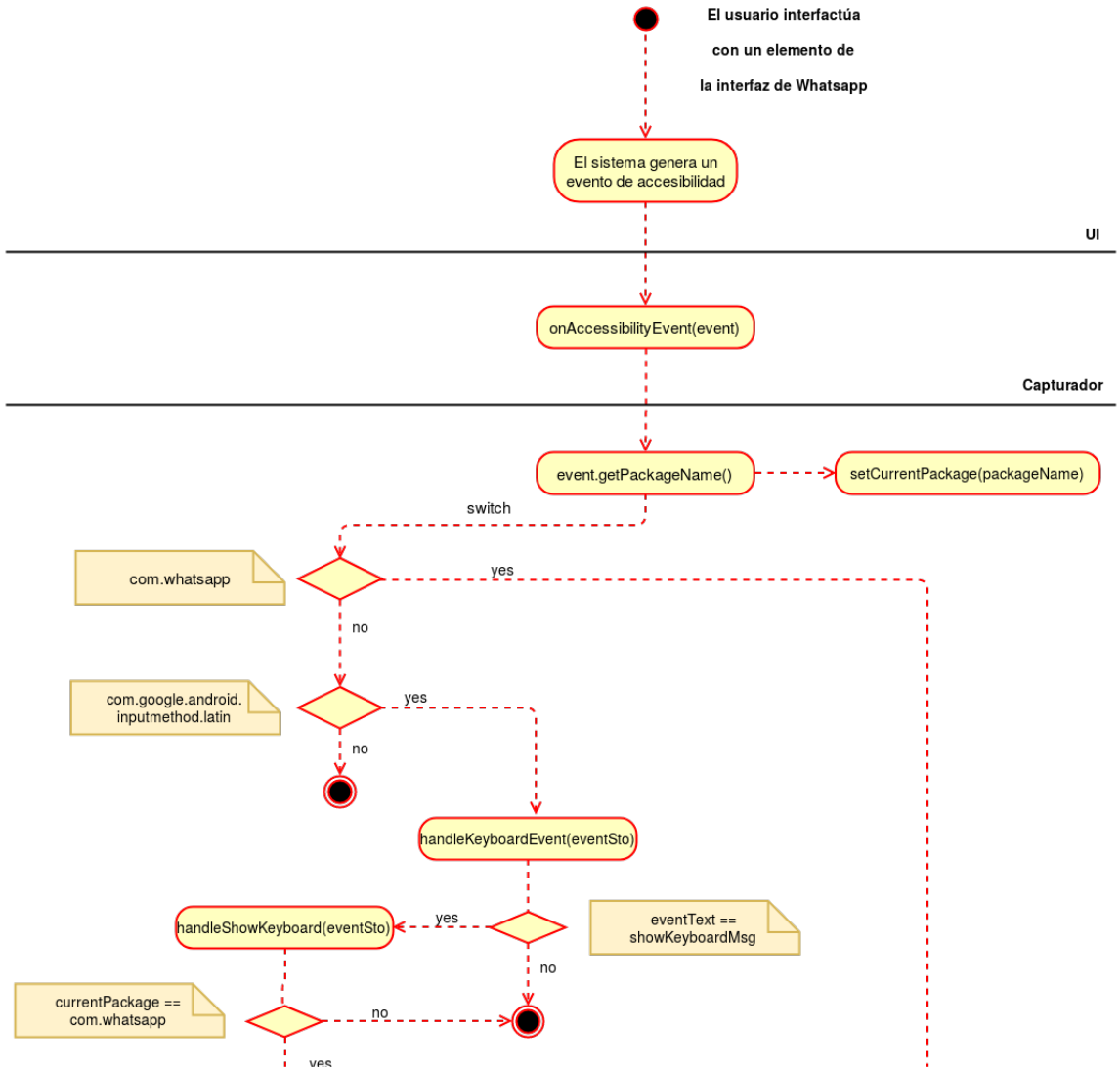


Figura 7.12: Diagrama de actividad de Whatsapp (1)

Diagrama de actividad de Whatsapp (1)

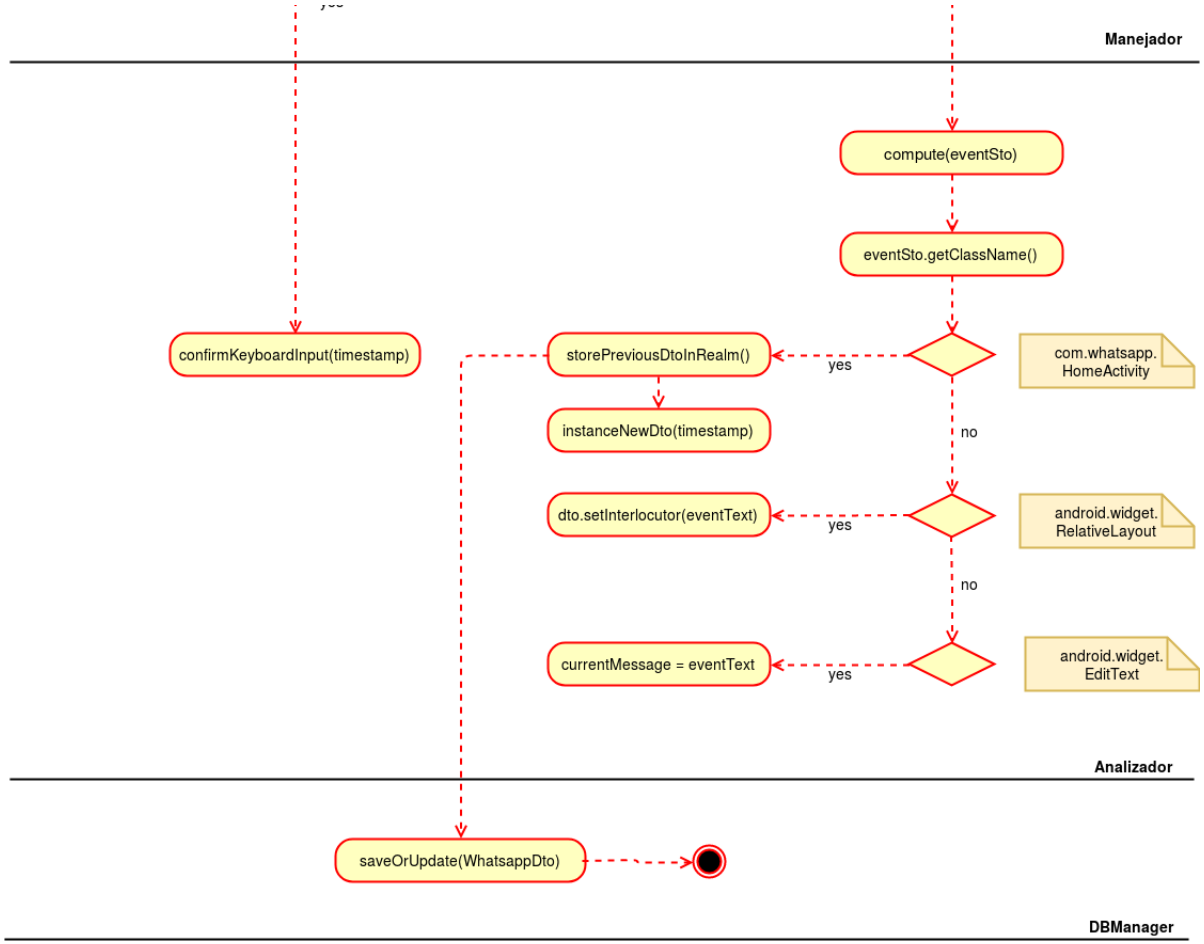


Figura 7.13: Diagrama de actividad de Whatsapp (2)
Diagrama de actividad de Whatsapp (2)

Como vemos, el algoritmo de captura recae en su mayor parte en el WhatsappAnalyzerImpl. Vemos sendas actividades vinculadas a métodos. Analicemos en que consisten estos métodos, puesto que conforman los pilares sobre los que se asientan la caputura.

**Compute(eventSto**

Este método recibe una instancia de la clase EventSto, que como ya hemos visto, incluye el propio evento de accesibilidad, su className y packageName.

Mirando el className construimos un switch, que en función de su valor dirige el flujo de captura hacia un lado u otro.

En caso de coincidir el valor con *com.whatsapp.HomeActivity*, llamaremos al método que chequea y guarda en la base de datos cualquier instancia existente del WhatsappDto.

Si por el contrario nos encontramos con *android.widget.RelativeLayout*, sabremos que el usuario ha entrado en una conversación de Whatsapp, deberemos por tanto setear el valor del interlocutor al correspondiente al texto extraído del evento.

Por último, si estamos tratando con un evento relacionado con *android.widget.EditText*, significará que el usuario se encuentra escribiendo en el campo de texto, y deberemos extraer y almacenar el propio texto del evento.

**storeObjectInRealm()**

Es el método responsable de llamar al DBManager para persistir el objeto en el que estamos guardando la información de captura de una conversación.

En primer lugar, este método asigna el timestamp del momento de captura del

evento al atributo endTimestamp del WhatsappDto, de esta forma añadiremos al contexto del usuario el momento exacto en el que abandona la conversación con su contacto.

Una vez hecho esto, se realiza una comprobación del resto de atributos del dto, incluyendo, que exista el propio objeto (que no sea nulo) y que el interlocutor no sea una cadena vacía. Si estas condicione se cumplen, el objeto será almacenado en la base de datos del teléfono llamando al DBManager.

**instanceNewDto(timestamp)**

El nombre del método clarifica bastante su tarea. Esta consiste en instanciar un nuevo objeto del tipo WhatsappDto y añadirle el timestamp, cuyo origen es el momento en que se capturó el evento, para asignarlo al atributo startTimestamp del nuevo objeto WhatsappDto.

De esta manera aportamos contexto a la captura, puesto que nos permite conocer el momento exacto en el que un usuario abre una conversación. Esto, junto con el endTimestamp nos permitirá averiguar cuánto tiempo pasa el usuario en cada una de sus conversaciones.

**confirmKeyboardInput(timestamp)**

A este método público se le llama directamente desde el manejador de eventos. Esto ocurre cuando se detecta un evento de teclado que coincide con la apertura del teclado ó sencillamente cuando el teclado se vacía después de escribir.

Como se ha explicado al inicio de esta sección, esto se traduce en el hecho de que el usuario ha mandado la cadena que se encontraba escribiendo, por lo tanto el método debe encargarse de chequear la validez de esa cadena y añadirla a la lista de mensajes del WhatsappDto

Gracias a este algoritmo, conseguiremos una completa recolección de lo que hace el usuario dentro del Whatsapp, sabremos con quien habla, lo que dice y durante cuanto tiempo se relaciona con sus contactos. Esta información por lo tanto se denota muy valiosa, ya que Whatsapp es la aplicación más usada para hablar con nuestro alegados, gracias al contexto aportado podremos conocer de que forma se relaciona el usuario con su entorno social más cercano.

## 7.4. Captura de Chrome

La captura de las búsquedas y páginas realizadas Chrome se antoja más sencilla, puesto que sólo necesitamos controlar las cadenas de búsqueda así como la marca horaria en la que fueron realizadas.

### 7.4.1. ChromeDto

El objeto sobre el que construimos la información capturada durante la navegación tiene el siguiente aspecto;

```
public class ChromeDto extends RealmObject {

    @PrimaryKey
    private String id;

    /** String containing the visited urls or search terms */
    private String searchUrl;

    /** Timestamp when the search happened */
    private Date timestamp;

// -- getters and setters
}
```

Obviemos ahora el id de Realm. Tenemos por un lado el atributo **searchUrl**, destinado a almacenar la url de las páginas visitadas o los términos de una búsqueda.

Por otro, el atributo **timestamp**, de tipo java.util.Date, que nos ayudará a situar en un marco temporal las búsquedas del usuario. De este modo persiguiendo nuestro objetivo de lograr un perfil lo más completo del mismo, podremos saber en

qué momento del día el usuario realiza sus búsquedas Web, conociendo además el motivo de esa búsqueda.

### 7.4.2. Diagram de actividad

De nuevo, nos encontramos en una situación en la que necesitamos controlar dos tipos de evento, por un lado los relacionados con la propia aplicación de Chrome, *com.android.chrome*, y por otro, el teclado *com.google.android.inputmethod.latin*.

#### Eventos de com.android.chrome

Estos eventos nos permitirán conocer lo que el usuario escribe en la barra de búsqueda, así como los sitios frecuentes que la propia aplicación muestra cada vez que abrimos una nueva pestaña.

Veamos a que elemento de la interfaz están relacionados los eventos para estos dos flujos de información;

- *android.widget.EditText*. Generado cuando el usuario escribe en la barra de búsuqeda. Por lo tanto, y del mismo modo que ocurría en Whatsapp, deberemos ir almacenando en una variable los caracteres vertidos por el usuario en esa barra de búsqueda. Ya llegará la confirmación por parte de un evento de teclado, que veremos mas adelante, para persistir esa información.

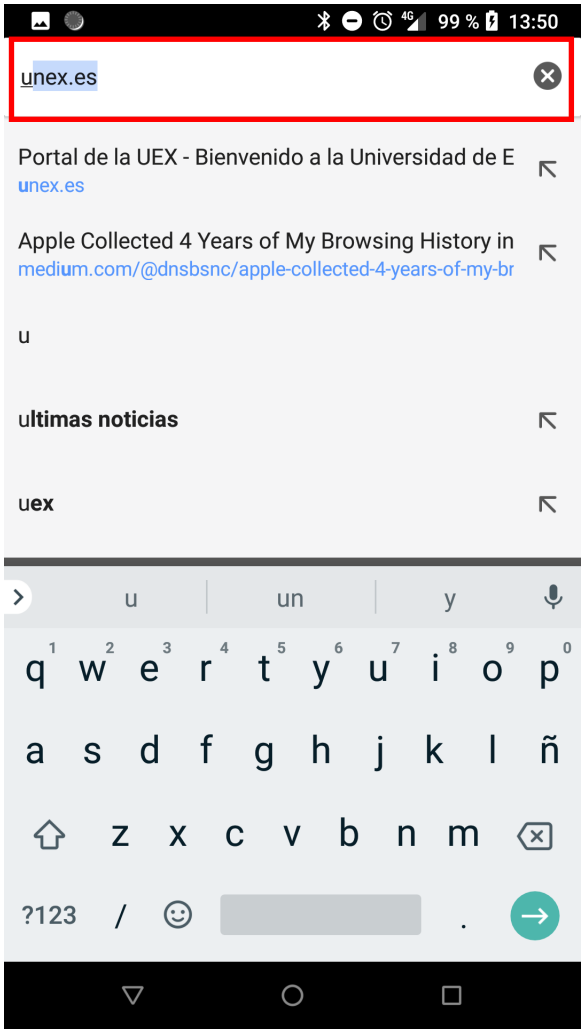


Figura 7.14: Componente edit text de Chrome)

Componente edit text de Chrome

- *android.widget.FrameLayout*. Asociados a la pulsación de una página guardada como sitios frecuentes. Dado que el usuario, este caso, no tiene que escribir nada en la barra de búsqueda, deberemos obtener la página visitada obteniendo el texto de este evento. De esta forma podremos saber que sitio almacenado visita cada vez.

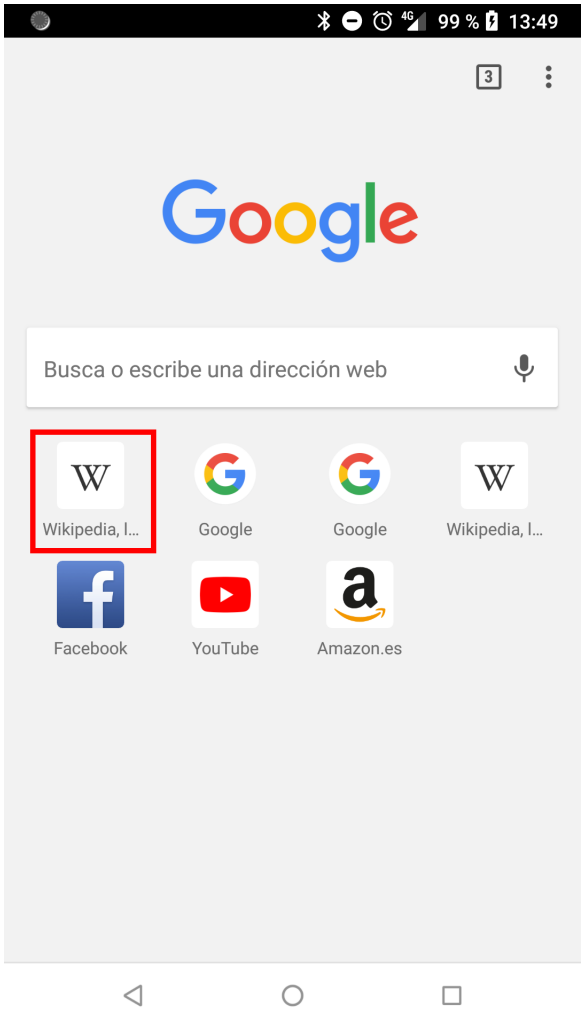


Figura 7.15: Componente frameLayout de sitios frecuentes en Chrome)

Componente frameLayout de sitios frecuentes en Chrome

- *android.view.ViewGroup*. Se ejecutan cuando el usuario pulsa sobre alguna de las sugerencias que va generando la aplicación a medida que el usuario escribe la url o búsqueda. Este flujo debe estar incluido, porque cuando el usuario acepta

alguna de estas sugerencias no se generan, como es obvio, eventos asociados a la escritura, si no que siguen su propio camino. Capturando estos eventos nos aseguramos de que nada se escapa en nuestra recolección de información.

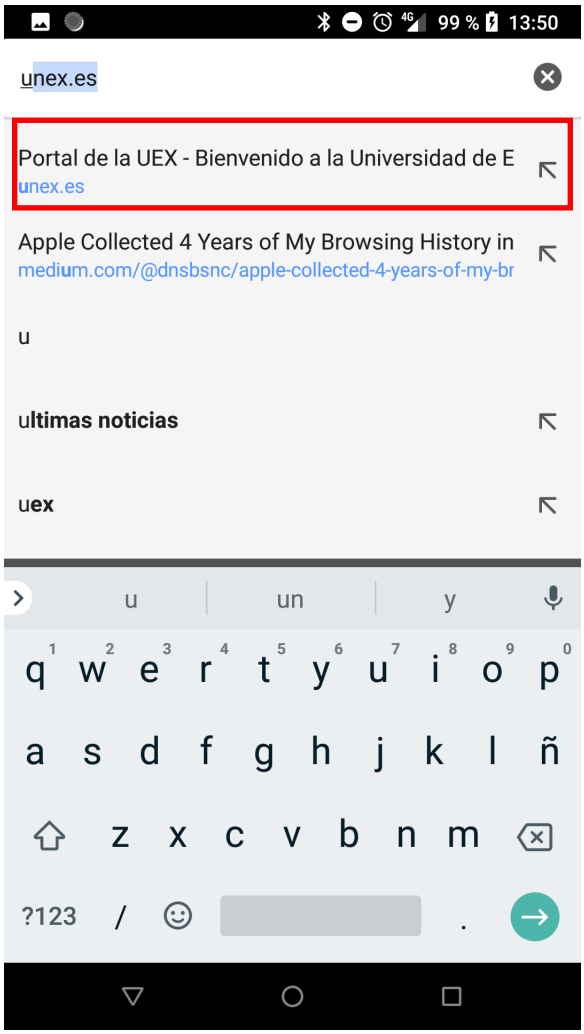


Figura 7.16: Componente view group de sugerencias en Chrome)

Componente view de sugerencias en Chrome

**Eventos de com.google.android.inputmethod.latin**

Usaremos los eventos del teclado en su otra vertiente, puesto que nos interesa ahora saber cuando el teclado se oculta.

Si recordamos como funciona esta app, cada vez que realizamos una búsqueda, el teclado se oculta para poder mostrar la página completa. Esto para nosotros es muy útil, puesto que cada vez que esto ocurre, significa que el usuario ha realizado una búsqueda.

Veamos ahora, mediante un diagrama de actividad, cómo es el flujo que sigue este proceso de captura.

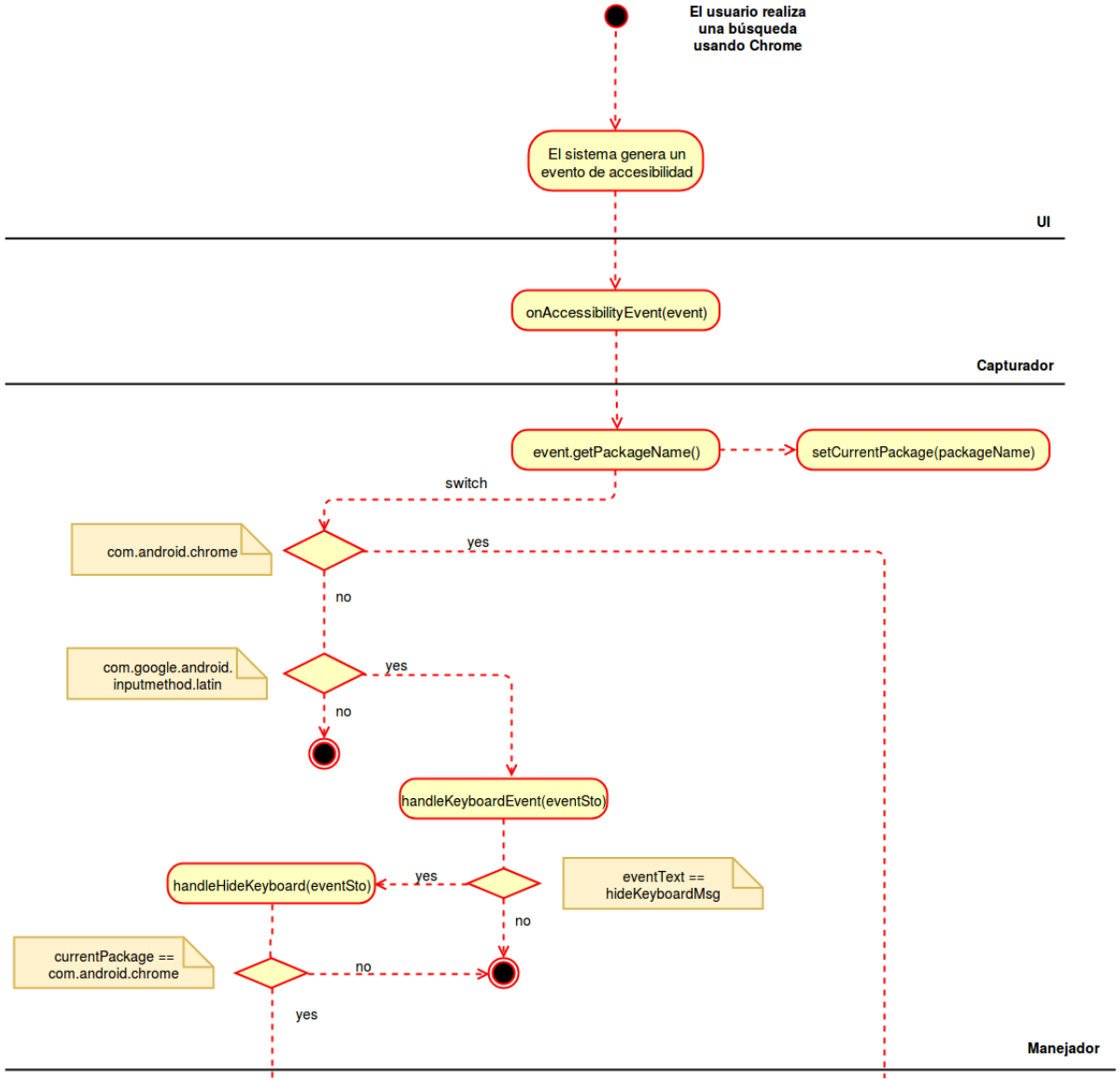


Figura 7.17: Diagrama de actividad de Chrome (1)

Diagrama de actividad de Chrome (1)

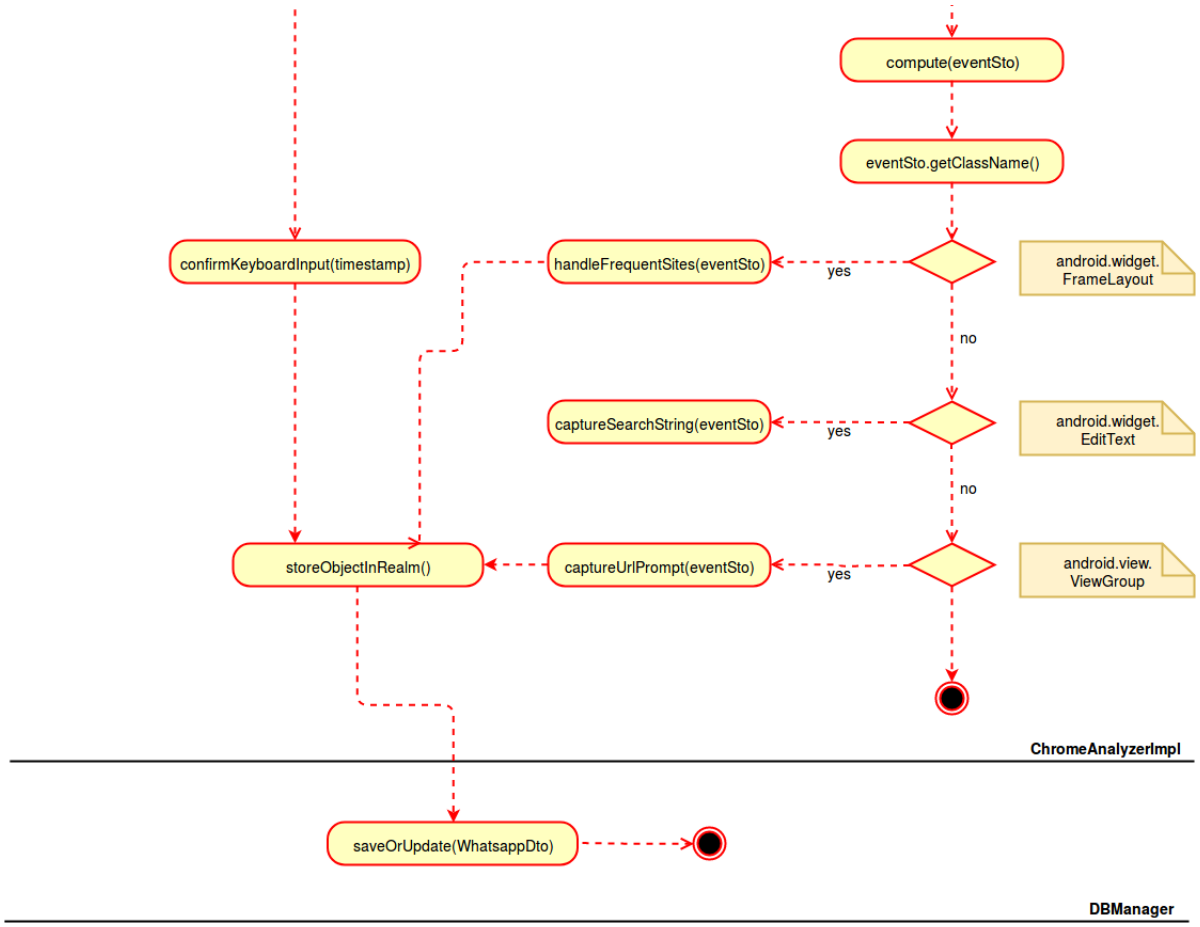


Figura 7.18: Diagrama de actividad de Chrome (2)

Diagrama de actividad de Chrome (2)

De nuevo, la actividad de análisis recae en su mayor parte en el método compute del ChromeAnalyzerImpl. Este compute se encarga, como se puede observar, de, en función del nombre de la clase del elemento de la ui asociado al evento, llamar a diversos métodos para construir el objeto donde alojaremos la información de la captura.

**handleFrequentSites(eventSto)**

Nos ayuda a extraer del evento el texto asociado a la pulsación de un sitio frecuente. En primer lugar, se verifica que la cadena no sea vacía. Si esto es así, se instancia un nuevo objeto del tipo ChromeDto, se le añade el timestamp y el texto de búsqueda (propio texto del evento).

Una vez hecho esto, se invoca al método storeObjectInRealm(), encargado de su persistencia.

**captureSearchString(eventSto)**

Este método, en primer lugar verifica que la cadena de búsqueda sea valida. En concreto, verifica que su longitud es mayor a la unidad. Esto es así porque a veces y de forma aparentemente aleatoria, se generan eventos de accesibilidad que han conseguido superar todos los filtros de cada nivel hasta llegar aquí, pero su contenido es la primer letra de la búsqueda o url.

Seguido de esto, se comprueba que el evento es del tipo *text selection changed* y superado este filtro, se instancia un nuevo objeto del tipo ChromeDto, se le añade la marca horaria, cadena de búsqueda y se deja el objeto preparado para cuando llegue el evento de teclado responsable de confirmar la entrada de texto.

**captureUrlPrompt(eventSto)**

Llamaremos a este método cuando el usuario presione sobre los sitios frecuentes, sugeridos por el propio Chrome.

La primera tarea que debe realizar el método es extraer la url visitada del texto del evento, puesto que este texto en crudo contiene la propia url y además, concatenada, el título de la página con la primera letra en mayúsculas.

Una vez obtenida la url visitada (se puede extraer recorriendo la cadena y buscando el carácter que está en mayúsculas), instanciamos un nuevo ChromeDto, le añadimos el timestamp, la cadena de búsqueda y persistimos en la base de datos.

De esta manera, podremos saber que páginas visita el usuario con su marca horaria. Los hábitos de navegación son un factor determinante a la hora de definir el perfil de usuario, puesto que una persona cuando busca en la web se refleja a sí misma, sus dudas, inquietudes y sus intereses. Cabe decir que en el proyecto se incluye también la captura de la barra de búsqueda de Google, por estar incluida en la mayoría de versiones de Android, y el navegador Firefox, al ser el segundo más asentado en el mercado.

## 7.5. Bibliotecas de terceros

### 7.5.1. Realm

Realm es una base de datos cuya implementación para Android nos ofrece una alternativa a SQLite, destactando por su rapidez y facilidad de uso.

Se trata de una base de datos orientada a objetos, facilitando así el mapeo objeto relacional a través del código, incluyendo de esta manera las facilidades de un ORM de forma nativa.

Para añadirla su dependencia basta con añadir en el build gradle (Project):

```
classpath "io.realm:realm-gradle-plugin:5.1.0"
```

En este caso se está usando exactamente la versión 5.1.0. Deberemos añadir en el build gradle (App):

```
apply plugin: 'realm-android'
```

# Capítulo 8

# Conclusiones y trabajos futuros