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

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

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, como su propio nombre indica, 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 features que usa 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 hablaba Nuria Ruiz (@pantojacoder), 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, cual 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.

Con esta idea en mente, 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 y enmascararla bajo la apariencia de una aplicación desarrollada con la intención de que el usuario tenga un punto de vista objetivo del grado de adicción a su terminal, mostrando estadísticas de uso, aplicaciones más frecuentes, etc.

A pesar de ser una idea con un uso potencialmente malicioso, se confía ciegamente en la labor de investigación de los profesionales del laboratorio y que el uso de los datos serán únicamente con propósitos de investigación. Además de lo atractivo que resulta

jugar con los aparentes límites de la tecnología, capturando información que circula libremente sin que nadie le preste atención.

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 de dividirá en varias etapas. La primera será estudiar las herramientas existentes en el y averiguar mediante que técnicas consiguen capturar, o más bien, leer, la pantalla del usuario, puesto que este es nuestro principal objetivo, registrar los cambios que ocurren en pantalla cuando el usuario interactúa con el teléfono.

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

Estado del Arte

Al final, el propósito que se persigue es enterarnos de los cambios realizados en el teléfono por parte del usuario, veamos las diferentes aproximaciones a este problema.

2.1. Traceview

Consiste en inspeccionar los logs generados por el sistema, con propósito de depuración en su mayor parte. – a ampliar.

2.2. Talkback

Lector de pantalla de google que emplea los servicios de accesibilidad que provee la plataforma para usuarios con discapacidades funcionales. – a ampliar.

Capítulo 3

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.

3.1. Requisitos

3.1.1. Requisitos funcionales

El sistema deberá:

- Registrar cada aplicación que lanza el usuario, con información suficiente que permita identificarla.
- Aportar un contexto temporal de cuando las aplicaciones son ejecutadas, esto es, fecha y hora exactas.
- Incluyendo lo anterior, registrar el uso dado a aplicaciones relacionadas con, comunicación, mensajería instantánea, búsqueda web,redes sociales y por último sensores.
- 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, Facebook e Instagram.
- Los sensores que escucharemos serán el emisor de 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 el objeto de búsqueda.
- El tiempo invertido en la aplicación.
- En **Búsqueda de google** el objeto de búsqueda.

Aplicaciones de redes sociales

- En **Twitter** capturar, el nombre de los perfiles visitados.
- Los los *retuits* realizados.
- Los *me gusta* dados.
- El contenido de los mensajes directos.
- Interlocutor de esos 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.
- Los *me gusta* dados.
- En **Instagram** capturar, el nombre de los perfiles visitados.
- Los comentarios hechos en dichos perfiles.
- Las subidas de fotos y su descripción.
- Los *me gusta* dados.
- Los mensajes directos.
- El interlocutor de esos mensajes directos.

Sensores

- 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.

3.1.2. 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.

Capítulo 4

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.

4.1. Diagramas de Casos de Uso

Si nos fijamos en como está planteado el proyecto, no tendremos un usuario como tal, si no que nos aprovecharemos de los movimientos de este para registrar de forma pasiva lo que hace.

Plantearemos entonces los casos de uso desde dos puntos de vista, el primero será aquel en el que usuario será el actor interaccionando con el teléfono y se dispara la recogida de información.

El segundo punto de vista en el que nos basaremos será aquel donde el actor será la propia recogida de información, empleando para ello sucesivos casos de uso que conformarán la lógica de negocio general del sistema.

Aquí tenemos los casos de uso del usuario entonces;

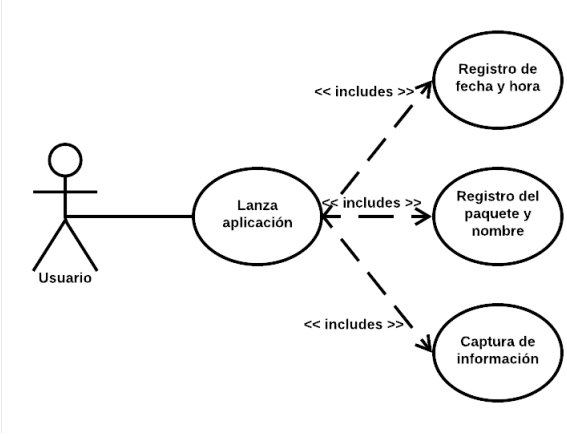

Figura 4.1: Casos de uso próximos al usuario.

Siendo los casos de uso desde el punto de vista de la captura de información los siguientes. Si nos centramos en el apartado de aplicaciones de comunicación;

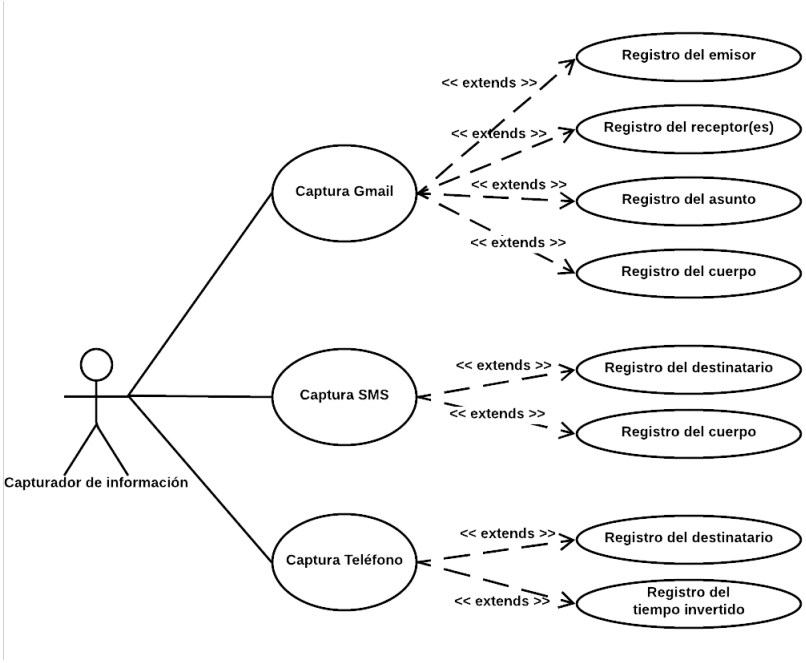

Figura 4.2: CU Apps de comunicación.

En el grupo de mensajería instantánea.

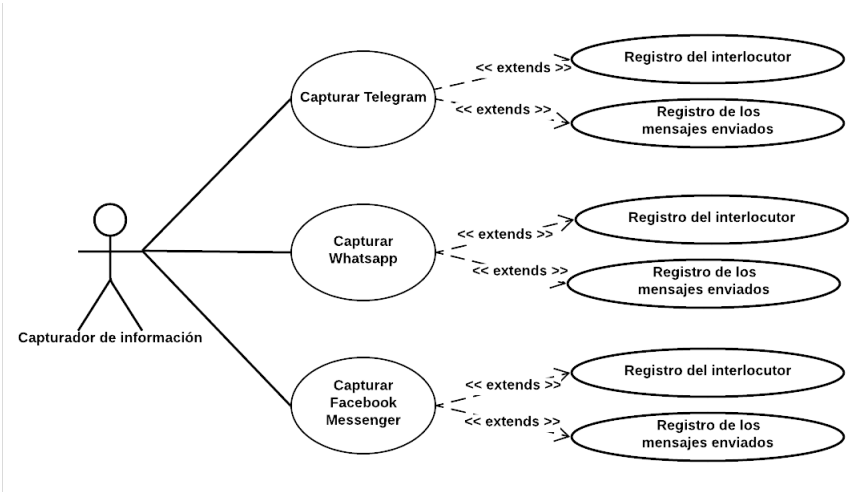

Figura 4.3: CU Apps de mensajería.

Aplicaciones de navegación y consulta de la web.

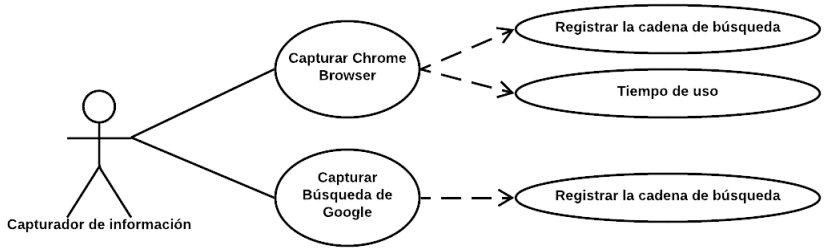

Figura 4.4: CU Apps de navegación web.

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

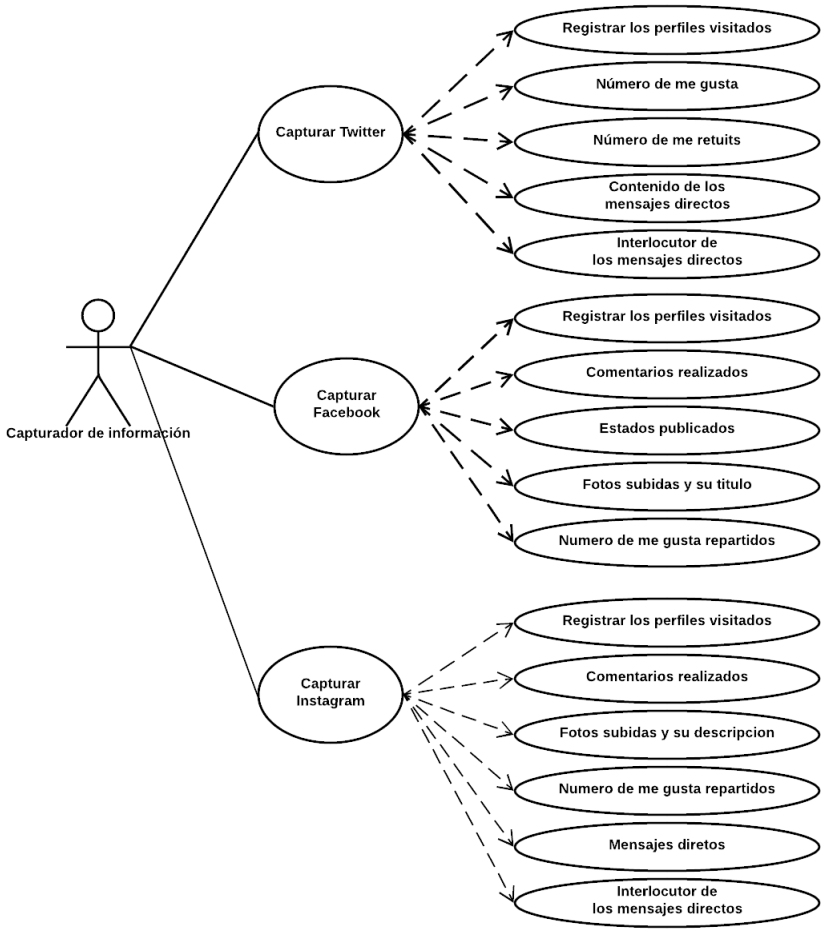

Figura 4.5: CU Apps RRSS.

Por último, en el apartado de los sensores, donde el actor será el capturador de la información de los sensores;

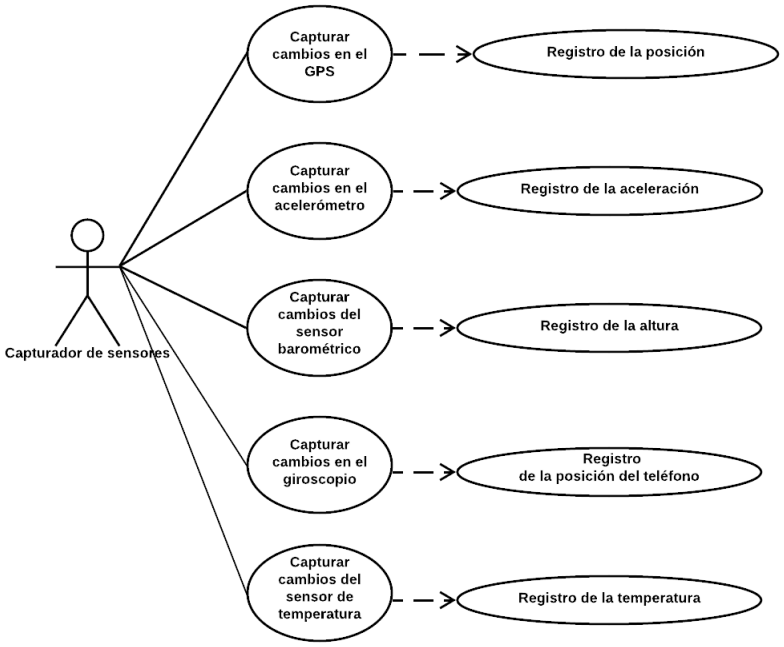

Figura 4.6: CU Sensores.

4.2. Arquitectura de alto nivel

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

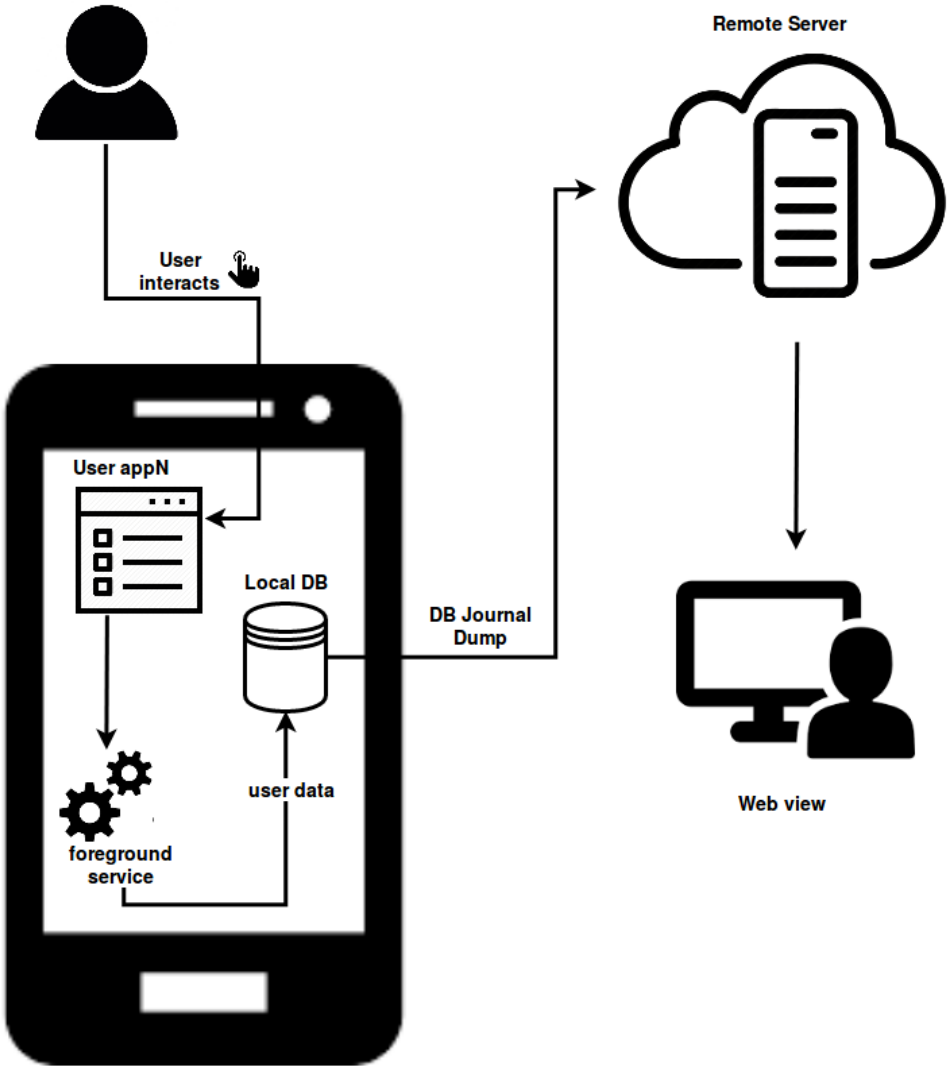

Figura 4.7: Arquitectura de alto nivel.

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.

4.2.1. 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.

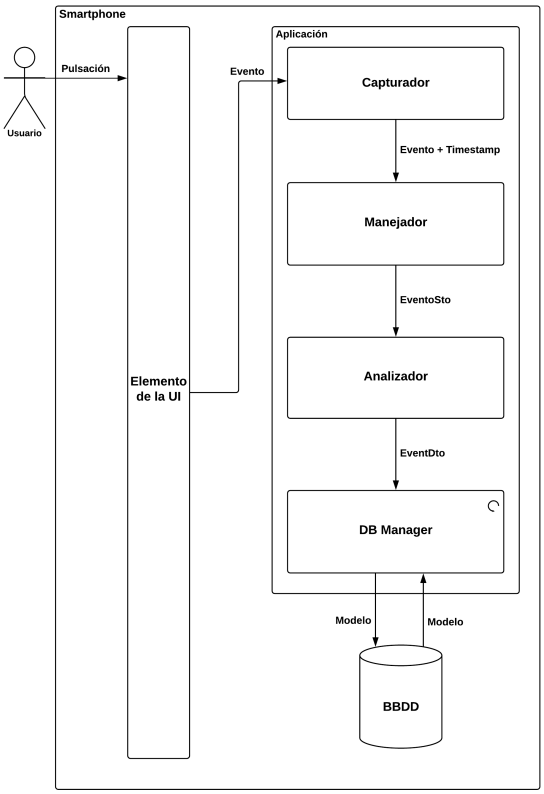

Figura 4.8: Arquitectura de alto nivel. Detalle de la parte móvil.

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, que lo aporta la relación que existe entre los eventos al estar asociados a través de un árbol, representando la vista que está viendo el usuario.

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 decír, 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 en 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 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.

4.2.2. 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

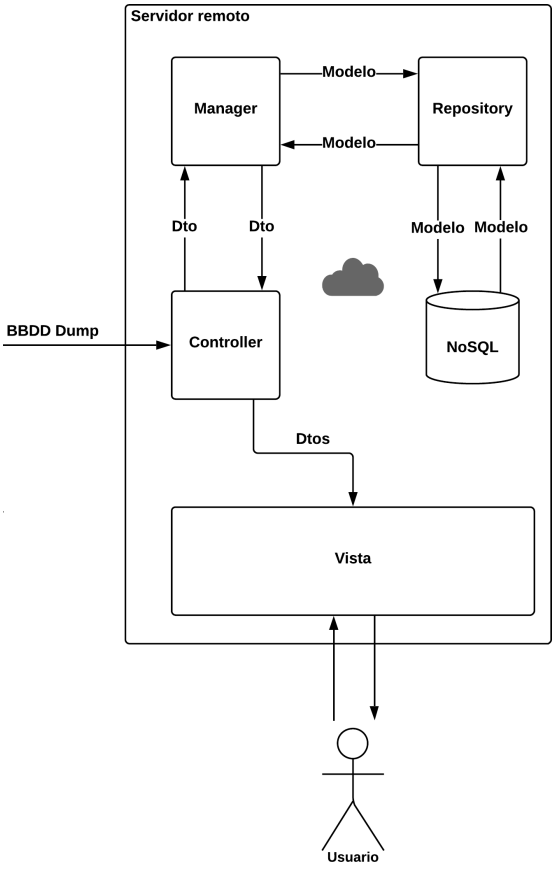

Figura 4.9: Arquitectura de alto nivel. Detalle del entorno cloud

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.

4.3. Diagrama de secuencia

Con el fin de explicar cual 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.

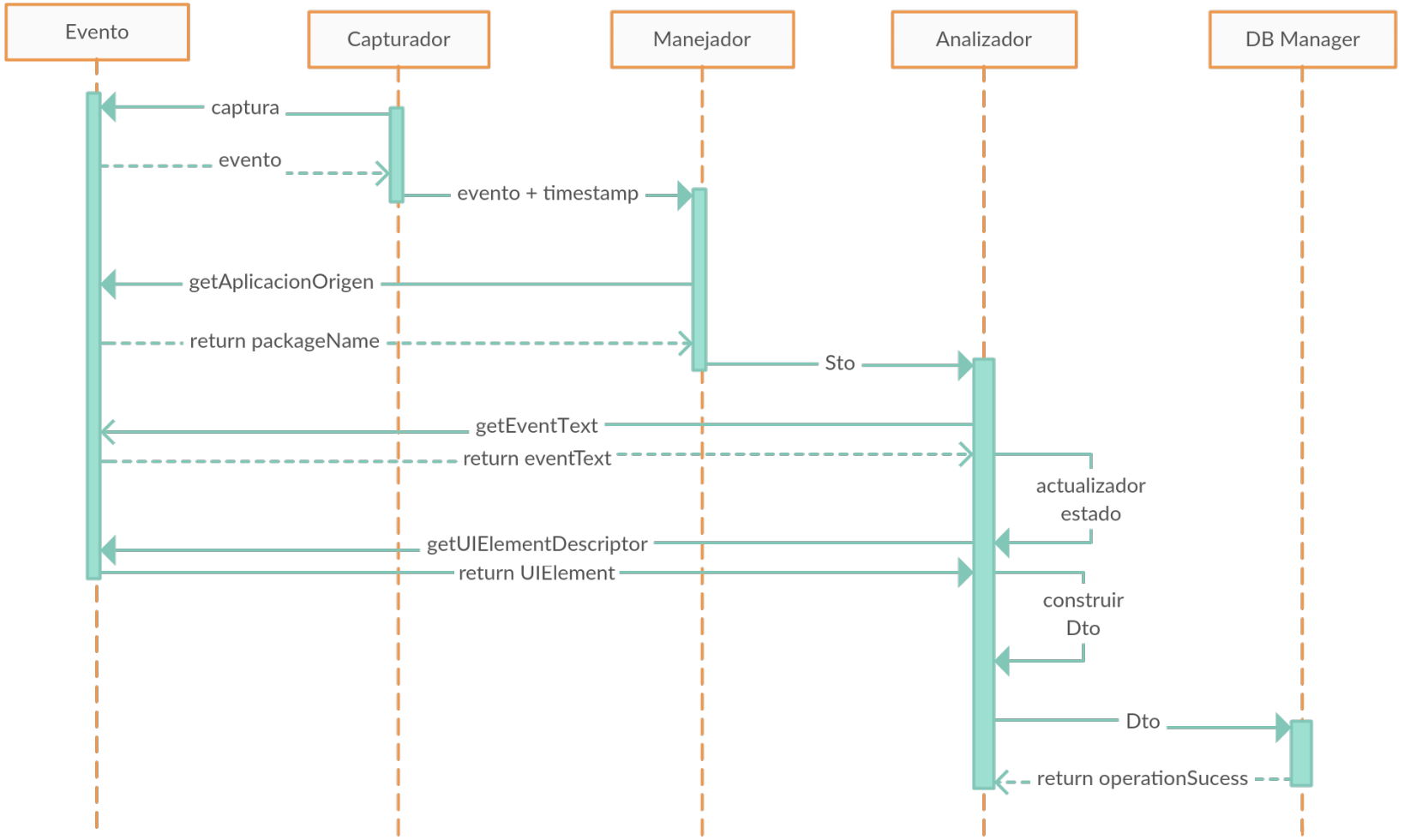

Figura 4.10: Diagrama de secuencia habitual del sistema.

4.4. Diagramas de actividad

En esta sección veremos varios diagramas de actividad destinados a clarificar y documentar el flujo lógico que seguirá la aplicación, durante el proceso de análisis y extracción del contenido de cada de las aplicaciones propuestas.

4.4.1. Gmail

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 el camino recorrido así como el procesamiento de la información en cada nivel.

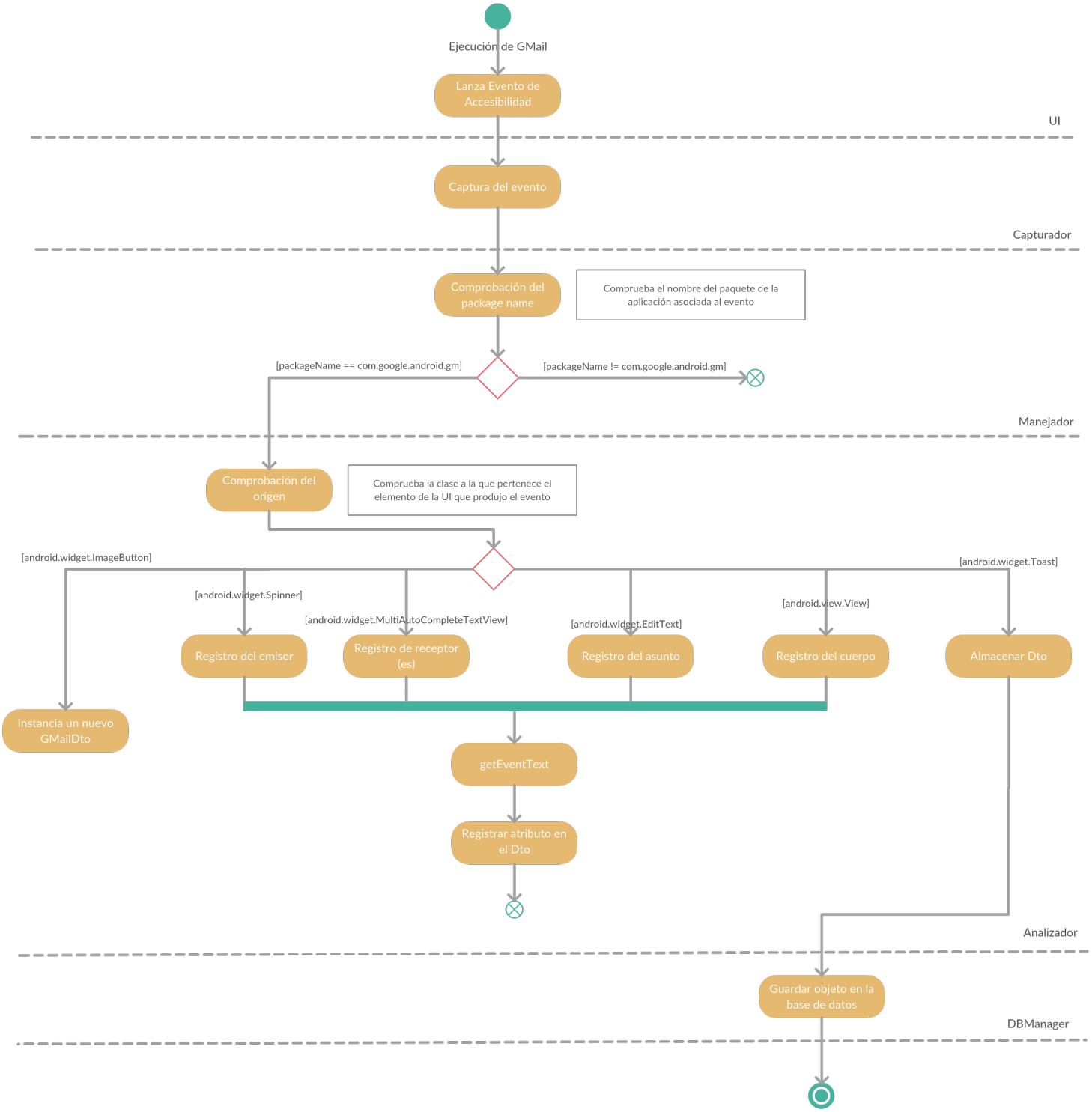

Figura 4.11: Diagrama de actividad de captura de un correo redactado con GMail.

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.

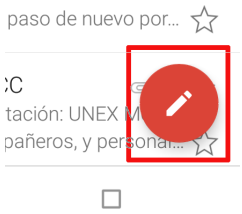

Figura 4.12: Android Image Button
Elemento botón que lanza la creación de un nuevo correo

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.

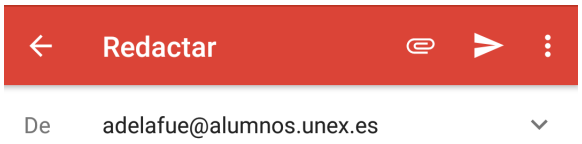

Figura 4.13: Android Widget Spinner

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 4.14: Android Widget MultiAutoCompleteTextView

Elemento gráfico donde aparece el receptor/es del correo.

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

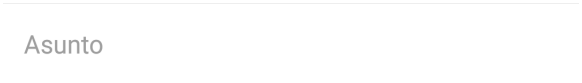

Figura 4.15: Widget Edit Text

Elemento de la interfaz donde escribir el asunto.

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

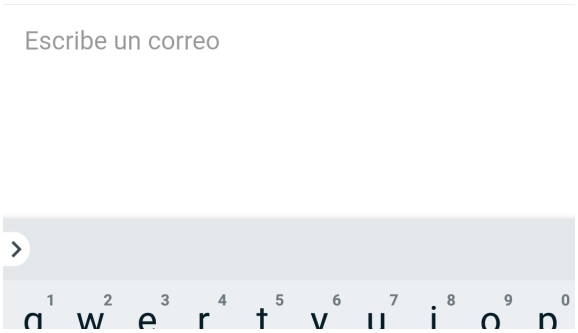

Figura 4.16: Android View

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.

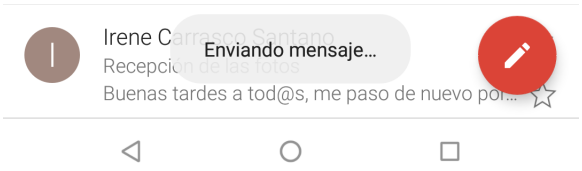

Figura 4.17: Android View

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.

Capítulo 5

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.

5.1. Arquitectura

A la hora de plantear nuestra arquitectura el primer paso es conocer la propia arquitectura de Android.

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.

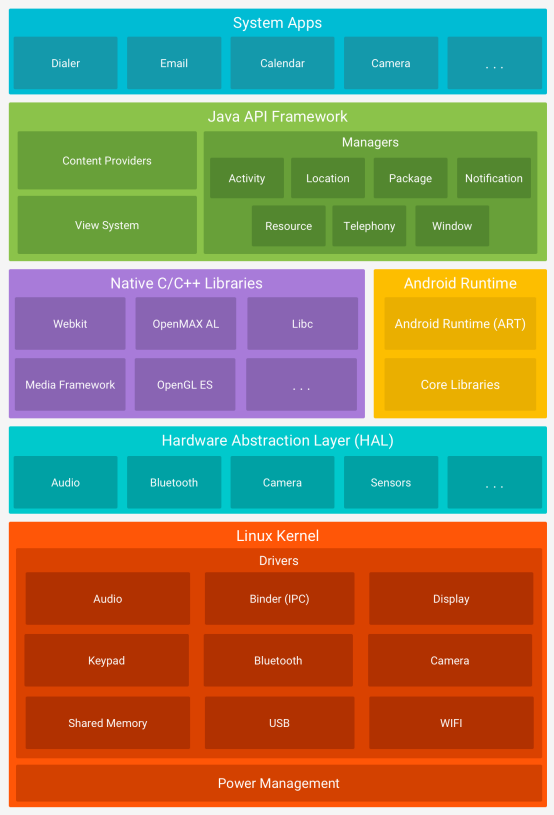

Figura 5.1: Arquitectura por capas de Android.

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 5.2: 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 lógica 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.

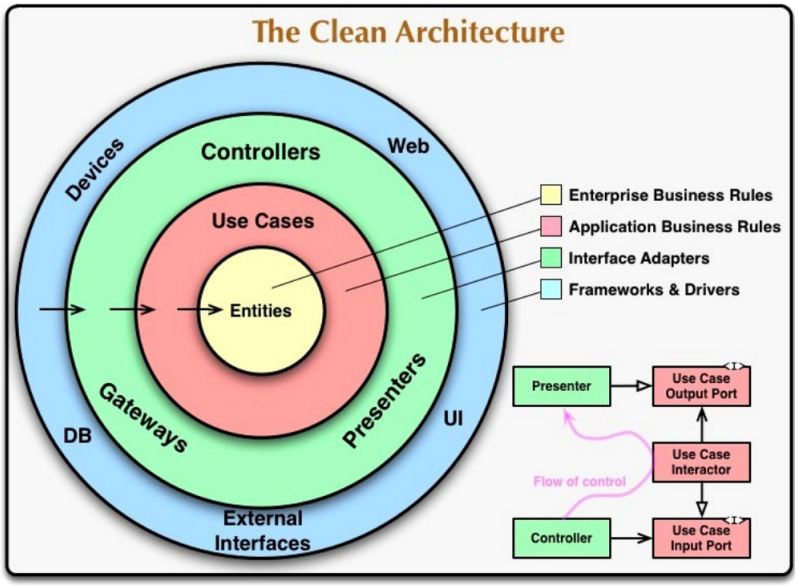

Figura 5.3: Arquitectura clean

Los casos de uso (use cases) están muy relacionados con las entidades, puesto que implementarán la lógica de negocio del sistema, puesto que deben orquestar y orientar

todo 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...

Si nos planteamos esto en los términos del proyecto, es decir, una aplicación que sea capaz de meterse en los servicios de accesibilidad que provee 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.

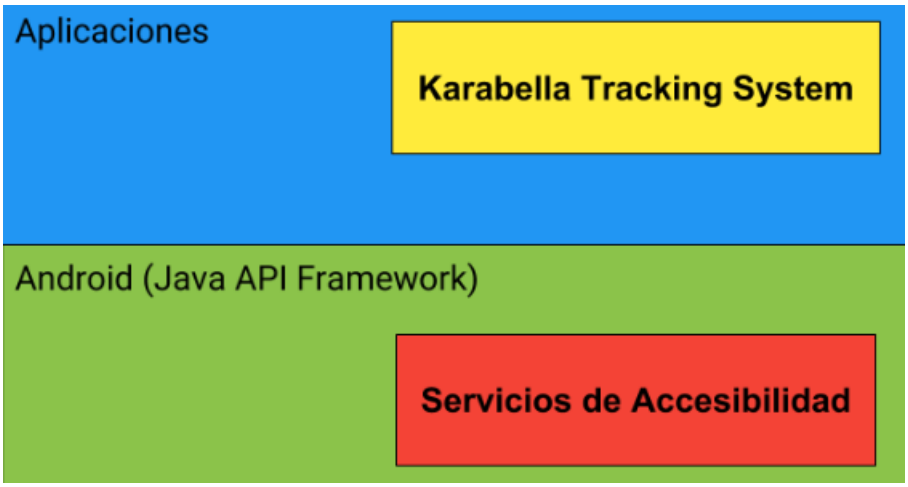

Figura 5.4: Arquitectura del sistema dentro del stack de Android.

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). Los Servicios de Accesibilidad se comportan al final como listeners de Eventos de Accesibilidad, que lanzados por el sistema. Estos Eventos de Accesibilidad son la entidad que manejan los servicios de accesibilidad.

Los eventos llevan consigo información sobre la interfaz de usuario, por ejemplo los eventos se lanza 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.
Así, en un primer vistazo, podemos ver la arquitectura usando un grano gordo del siguiente modo.

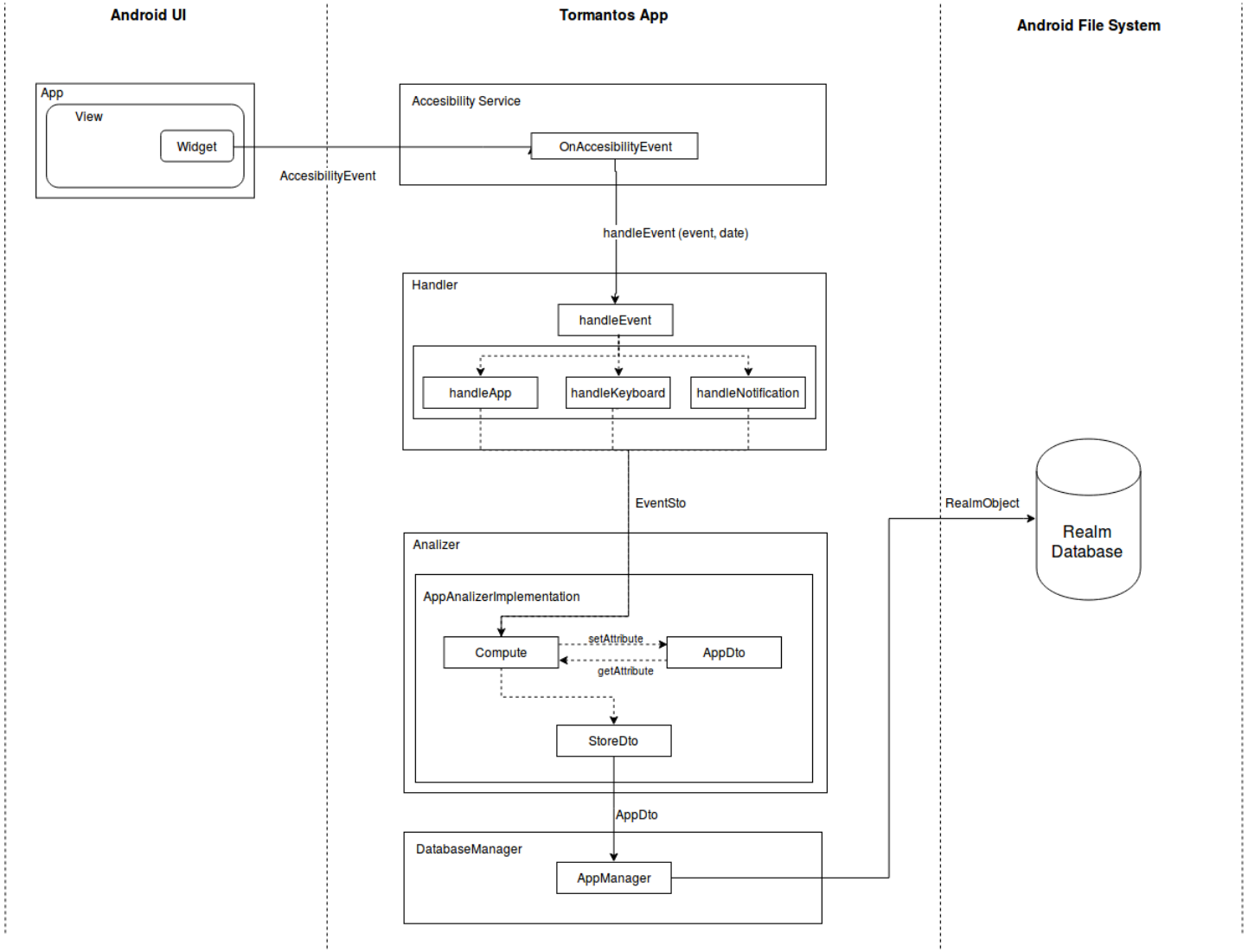

Figura 5.5: Arquitectura general del sistema

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.

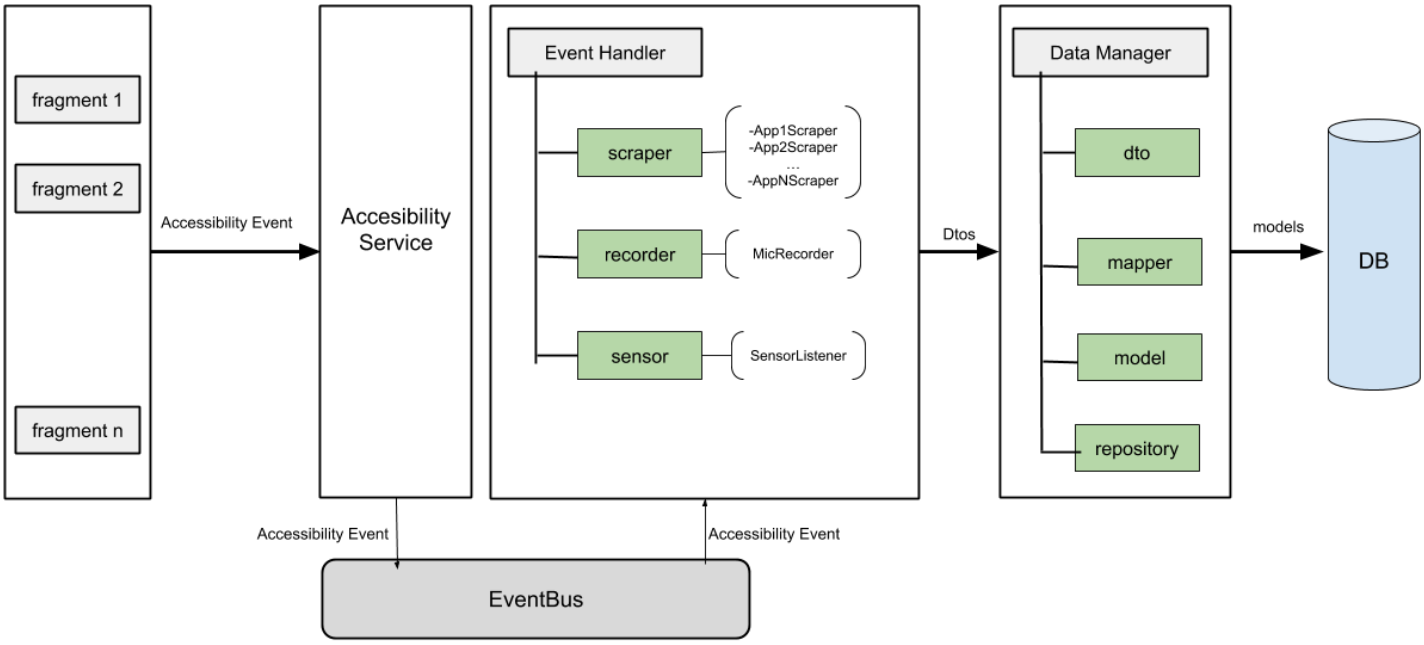

Figura 5.6: Arquitectura detallada

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 6

Implementación y desarrollo

6.1. Bibliotecas de terceros

6.1.1. 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 7

Conclusiones y trabajos futuros