UNIVERSIDAD PRIVADA ANTENOR ORREGO

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA DE COMPUTACION Y SISTEMAS

"ARQUITECTURA ADAPTADA PARA EL DISEÑO DE APLICACIONES MÓVILES EN ANDROID"

TESIS PARA OBTENER EL TITULO PROFESIONAL DE INGENIERO DE COMPUTACION Y SISTEMAS

LINEA DE INVESTIGACION :

ARQUITECTURA ORIENTADA A SERVICIOS EN APLICACIONES WEB

TESISTA(S):

CABRERA ABANTO, Alan Andres

CUEVA CHAMORRO, Diego Alonso

ASESOR:

PIMINCHUMO FLORES, Jorge Luis

TRUJILLO-PERU, 2015

TESIS PARA OBTENER EL TITULO PROFESIONAL DE INGENIERO DE COMPUTACION Y SISTEMAS

TITULO:

"ARQUITECTURA ADAPTADA PARA EL DISEÑO DE APLICACIONES MÓVILES EN ANDROID"

DESARROLLADO POR:

Br. Diego Alonso Cueva Chamorro Tesista	Br. Alan Andres Cabrera Abanto Tesista

APROBADO POR:

Ing. Armando Javier Caballero Alvarado Presidente Nº CIP: 149181	Ing. Henry Antonio Mendoza Puerta Secretario Nº CIP: 139568
Ing. Walter Manuel Cueva Chavez Vocal Nº CIP: 139607	Ing. Jorge Luis Piminchumo Flores Asesor Nº CIP: 137153

PRESENTACIÓN

Señores Miembros del Jurado:

De conformidad a lo estipulado en el Reglamento de Grados y Títulos de la Universidad Privada Antenor Orrego, ponemos a vuestra consideración el trabajo de Investigación Titulado: "ARQUITECTURA ADAPTADA PARA EL DISEÑO DE APLICACIONES MÓVILES EN ANDROID" a fin de ser evaluado.

Este trabajo es el resultado de nuestra experiencia laboral adquirida en el desarrollo de aplicaciones móviles, que se plasma en una arquitectura de diseño que esperamos permita facilitar y mejorar el diseño de más aplicaciones móviles desarrolladas para la plataforma Android.

Esperamos que el presente trabajo logre cubrir las expectativas que tienen al respecto, excusándonos de antemano por los errores involuntarios incurridos en el desarrollo del mismo.

Br. Diego Alonso Cueva Chamorro

Br. Alan Andres Cabrera Abanto

DEDICATORIA

DEDICACTORIA DE ALAN CABRERA

A mi familia, que me han instado y brindado todo lo necesario para llegar hasta donde estoy, siendo el soporte necesario para mi desarrollo.

A mis amigos y amigas, quienes siempre han estado en los momentos en que los he necesitado, y en lo que no también.

DEDICATORIA DE DIEGO CUEVA

A mis padres Alicia y Alfredo, por ser los mejores padres que cualquiera podría soñar. Por su cariño, confianza y apoyo infinito. Me queda esta vida y la siguiente para agradecerles y aspirar a ser la mitad de bueno que ustedes son.

A mis hermanas Livia y Mónica, a quienes amo y admiro desde siempre.

A mi pequeño ejército de ángeles, que desde el cielo me cuidan, me protegen y guían cada uno de los pasos que doy.

AGRADECIMIENTO

A DIOS, por mostrarnos día a día que con humildad paciencia y sabiduría todo es posible.

A JORGE PIMINCHUMO FLORES, por la confianza, la motivación y el apoyo que nos ha brindado como asesor de este trabajo de tesis.

RESUMEN

“ARQUITECTURA ADAPTADA PARA EL DISEÑO DE APLICACIONES MÓVILES EN ANDROID”

Por:

Bach. Cabrera Abanto, Alan Andres

Bach. Cueva Chamorro, Diego Alonso

La presente tesis brinda una arquitectura para el diseño de aplicaciones móviles en la plataforma Android y la aplica en la implementación de una aplicación para taxis.

Para este estudio se realizó la investigación de los fundamentos de la plataforma móvil Android, tecnologías móviles, conceptos de arquitectura de software y el lenguaje de modelamiento unificado.

Para la presentación de la arquitectura desarrollada se hace uso de varias vistas arquitectónicas con las cuales se muestra sus diferentes perspectivas, utilizando para ello diagramas realizados con el lenguaje de modelamiento unificado. Asimismo se presentan los criterios utilizados en la elaboración de la arquitectura y los componentes utilizados para diferentes escenarios.

Finalmente se aplica la arquitectura presentada al desarrollo de una aplicación móvil en Android para la gestión de servicios, seguimiento y control de unidades de una empresa de taxis.

ABSTRACT

“ADAPTED ARCHITECTURE FOR DESIGN OF MOBILE APPLICATIONS IN ANDROID”

by:

Bach. Cabrera Abanto, Alan Andres

Bach. Cueva Chamorro, Diego Alonso

This thesis provides an architecture for the development of mobile applications on the Android platform and applied in the implementation of an application for taxis.

For this study investigating the fundamentals of the Android mobile platform , mobile technologies , software architecture concepts and language of unified modeling was performed.

For the presentation of the architecture developed multiple architectural views are used with which their different perspectives shown , using diagrams made with unified modeling language is made. The criteria used in the development of architecture and components used for different scenarios also presented .

Finally the presented architecture is applied in the development of a mobile application on Android for service management, monitoring and units control in a taxi company.

INDICE GENERAL

Pág.

INDICE DE TABLAS

INDICE DE ILUSTRACIONES

INTRODUCCION

Android es un sistema operativo dirigido a equipos móviles. El propietario de la plataforma es Google y es actualmente la plataforma con mayor crecimiento a nivel mundial. Es la primera plataforma completa, abierta y gratuita para móviles (Conder & Lauren, 2011) . En la actualidad existen más de 1,2 millones de aplicaciones disponibles y diariamente, con muchas nuevas aplicaciones de diferentes tipos siendo publicadas diariamente en Google Play, la tienda de aplicaciones Android de Google.

El desarrollo de aplicaciones en Android es relativamente sencillo y se cuenta con abundante documentación en Internet de cómo aprovechar el potencial de los teléfonos móviles, sin embargo programar en Android es conceptualmente diferente a programar para otros entornos, como realizar una página web (Gargenta, 2011). Pocos desarrolladores tienen un conocimiento profundo de la plataforma y de los diferentes componentes, propios de Android, que brinda para la creación de aplicaciones, las ventajas y desventajas de usa un componente u otro en temas como rendimiento o robustez.

Además de esto, la mayoría de la literatura disponible sólo muestra aplicaciones incompletas, o ejemplos enfocados en alguna tarea específica, mas no muestran una vista completa o una arquitectura de cómo estructurar una aplicación mediana o grande, los componentes a utilizar y cómo realizar la interacción entre ellos. Es responsabilidad de cada equipo definir la estructura que utilizará el proyecto en desarrollo.

Esto lleva muchas veces a errores o confusiones al momento de realizar el diseño e implementación una aplicación en Android, causando problemas como demoras en el desarrollo, aplicaciones que presentan errores en tiempo de ejecución o no tienen una buena performance, especialmente cuando son ejecutadas en equipos de gama baja.

Siendo el desarrollo de nuevas de aplicaciones de todo tipo en la plataforma Android una tendencia en la actualidad, no existe una arquitectura definida que indique cómo estructurar estos proyectos. Es importante contar con dicha arquitectura como base pues facilita el desarrollo de las aplicaciones, ahorrando tiempo en diseño y construcción. Lo

que nos lleva a plantearnos la pregunta ¿Es posible crear una arquitectura común para aplicaciones Android?

Por lo tanto se planteó el siguiente problema:

- ¿Cómo mejorar el diseño de las aplicaciones móviles en Android?

Y para resolver dicho problema se plantea la siguiente hipótesis:

- Una arquitectura adaptada mejorará el diseño de las aplicaciones móviles en Android.

Como objetivo general tenemos:

- Proponer una arquitectura adaptada para el desarrollo de aplicaciones móviles en Android

Siendo los siguientes objetivos específicos:

- Realizar una investigación bibliográfica sobre los temas de materia de estudio del presente proyecto.
- Formular la propuesta de arquitectura para el diseño de las aplicaciones móviles en Android.
- Desarrollar un caso de prueba para evaluar la arquitectura propuesta.
- Realizar la contrastación de la hipótesis.

La presente investigación se encuentra estructurada de la siguiente manera:

En el Capítulo I: Fundamento Teórico. Se explica el conocimiento teórico utilizado en la elaboración de la investigación como conceptos de las tecnologías móviles, las características y lenguaje de programación de la plataforma Android, conceptos de arquitectura de software y el lenguaje de modelamiento unificado.

En el Capítulo II: Descripción de la Arquitectura Formulada. En este capítulo se mostrará la arquitectura desarrollada, utilizando varias vistas arquitectónicas para la presentación de sus diferentes perspectivas, utilizando para ello diagramas realizados con el lenguaje de modelamiento unificado.

En el Capítulo III: Caso. Se presenta el caso de una aplicación móvil bajo la plataforma Android para la gestión de servicios, seguimiento y control de unidades de una empresa de taxis, en la cual se hace uso de la arquitectura presentada.

En el Capítulo IV: Discusión. Comprobación de los supuestos sobre la base de los cuales se emprendió este trabajo de investigación son aceptables una vez confrontados con la realidad en la cual se pretende aplicar. Se evalúan los resultados obtenidos de aplicar los medios de contrastación de la hipótesis presentada.

En el Capítulo V: Conclusiones. Se exponen las conclusiones del trabajo realizado.

En el Capítulo VI: Recomendaciones. Se exponen las recomendaciones del trabajo realizado.

En el Capítulo VII: Referencias Bibliográficas. Se indican las fuentes utilizadas para la realización de la investigación

En el Capítulo VIII: Anexos. Se adjuntan al trabajo de tesis los anexos correspondientes.

CAPITULO I: FUNDAMENTO TEORICO

1.1. Teoría

1.1.1. Sistema

Un sistema es una matriz de componentes que colaboran para alcanzar una meta común, o varias, al aceptar entradas, procesarlas y producir salidas de una manera organizada. (Oz, 2006)

No todos los sistemas tienen una sola meta. A menudo un sistema está formado por varios subsistemas con metas secundarias, todas las cuales contribuyen a alcanzar la meta principal. Los subsistemas pueden recibir entradas y transferir salidas a y de otros sistemas o subsistemas.

Los sistemas son cerrados o abiertos dependiendo de la naturaleza del flujo de información en el sistema. Un sistema cerrado es independiente y no tiene conexión con otros; nada entra de otro sistema, nada sale hacia otro sistema. Un sistema abierto se comunica e interactúa con otros sistemas.

1.1.2. Sistema de Información

Se denomina aplicación web a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. Existen aplicaciones como los

webmails, wikis, weblogs, tiendas en línea y la propia Wikipedia que son ejemplos bien conocidos de aplicaciones web.

Es importante mencionar que una página Web puede contener elementos que permiten una comunicación activa entre el usuario y la información. Esto permite que el usuario acceda a los datos de modo interactivo, gracias a que la página responderá a cada una de sus acciones, como por ejemplo rellenar y enviar formularios, participar en juegos diversos y acceder a gestores de base de datos de todo tipo.

1.1.3. Aplicaciones Móviles

Las aplicaciones móviles son relativamente aplicaciones simples que realizan una tarea específica para el usuario. Algunas aplicaciones deben ser instaladas en el dispositivo para realizar estas tareas, algunas veces interactúan con el hardware del dispositivo, incluyendo la cámara, el dispositivo de almacenamiento, el sensor GPS, etc. Las aplicaciones móviles también pueden ser utilizadas para construir flujos de trabajo específicos del usuario mediante la orquestación de piezas de lógica personalizada y componentes nativos, como el Servicio de Mensajes Cortos (SMS) del subsistema, el programa de correo electrónico predeterminado, y el navegador preinstalado.

Las aplicaciones móviles son aplicaciones de software, por lo que las prácticas conocidas de buen diseño y desarrollo se pueden aplicar. Sin embargo, las aplicaciones móviles se ejecutan en los dispositivos móviles, y estos dispositivos son significativamente diferentes de las computadoras portátiles y computadoras de escritorio.

1.1.3.1. Aspectos Críticos del Software de Aplicaciones Móviles

Las aplicaciones mobiles son software diseñado para ser ejecutado en dispositivos especiales. Esta aparente simple factor impone un numero de restricciones en el desarrollador y origina unas pocas

cuestiones para los cuales nuevos patrones y prácticas son requeridas.

1.1.3.1.1. El Modelo de Interacción

La diferencia clave entre una aplicación desktop y una móvil es que el usuario de una aplicaciones móvil moderna usara sus dedos para seleccionar cualquier contenido disponible. Usar los dedos para seleccionar el ítem de un menú es una muy natural y placentera experiencia para el usuario, pero no si el ítem a seleccionar es demasiado pequeño.

Un dedo, en efecto, nunca sera preciso como el puntero de un mouse o un stylus. ¿Y que hay acerca de escribir texto entonces? El dispositivo puede tener un teclado físico o dependiendo del dispositivo un teclado virtual que emula al físico, tipear en un dispositivo móvil es definitivamente problemático. Si el teclado es proveido mediante software, entonces este terminará ocultando gran parte de la interface de usuario. Algunas veces el teclado oculta algún botón que el usuario tiene que seleccionar para continuar con la operación. Todo esto es molesto y deben abordarse a nivel de diseño, manteniendo prácticas de usabilidad.

1.1.3.1.2. El Modelo de Presentación

Las pantallas sensibles al tacto fuerzan a incrementar el tamaño de cualquier contenido seleccionable como los botones principales y los enlaces, no podemos simplemente hacer a cada botón un gran enlace, el total del tamaño de la pantalla de un dispsitivo mobil es mucho más pequeño que la pantalla de una laptop.

El efecto combinado de estos dos factores conduce a un replanteamiento completo de la interfaz de usuario de una aplicación móvil. Una aplicación mobil es diseñada a partir

de unos muy específicos casos de uso, cada uno de estos buscan ejecutarse en la pantalla principal. En la implementación de cada caso de uso el numero de pasos que el usuario requerirá deberá ser el minimo.

La presentación de los datos también está sujeta a la nueva normativa. El desplazamiento es muy aconsejable, y se hace verticalmente en su mayor parte. Sin embargo, las formas de desplazamiento horizontal están subiendo, sobre todo en Windows Phone con la vista panorámica. Esta es una diferencia importante y notable con las aplicaciones de escritorio y web. Una pauta ampliamente aceptada dice que el desplazamiento horizontal debe evitarse por completo en las páginas web y minimizarse en las interfaces de usuario de escritorio. Además, en el móvil, el gesto de deslizar horizontal es una forma aceptada para voltear a través de bloques distintos pero relacionados de contenido. Mientras que un desplazamiento es sólo un gesto de introducir nuevos contenidos, pero relacionado, en general, transmite la idea de que se puede navegar a través de los contenidos de las páginas. Como desarrollador, desplazarse es una excelente manera de ahorrar un valioso espacio en la pantalla sin sacrificar la sensación de continuidad en la experiencia de la presentación.

1.1.3.1.3. El comportamiento de la aplicación

El estado de la pantalla no es el único recurso que se limita en un dispositivo móvil. El poder de procesamiento, es decir, la potencia de la CPU y la unidad de procesamiento gráfico asociado (GPU) no es comparable a lo que es típico en la computación de escritorio; Por lo tanto, se utiliza para la memoria y almacenamiento local. En particular, algunos dispositivos también impiden (o al menos dificultan) el acto

de montar una tarjeta de memoria SD adicional sobre el dispositivo.

Esto significa que el consumo de memoria se debe mantener bajo control estricto, y la optimización de los algoritmos es más importante que nunca. Además, por su naturaleza inherente, un dispositivo móvil es interrumpible y es utilizado por los usuarios de una manera que requiere una lógica multitarea fuerte. Multitarea, sin embargo, los conflictos con la falta de capacidad de procesamiento y memoria. El efecto neto es que todas las aplicaciones se deben crear desde cero en torno a la idea de que se pueden poner en un estado de reposo y hasta se quitan de la memoria cuando el sistema recupera sus recursos.

Por último, las aplicaciones móviles funcionan con energía de la batería y están sujetos a los caprichos de la conectividad.

Debido a que el usuario podría estar viajando cuando se trata con el dispositivo, la conectividad puede aparecer y desaparecer en cualquier momento, y también podría cambiar su calidad como el usuario se mueve de una zona Wi-Fi a 3G en roaming.

El comportamiento real de las aplicaciones móviles, por lo tanto, depende de una variedad de temas que la mayoría de los desarrolladores nunca enfrentó antes.

1.1.3.1.4. Las preocupaciones de seguridad para software móvil

En los últimos 15 años de desarrollo web, arquitectos y desarrolladores han aprendido mucho acerca de las amenazas a la seguridad de los datos y las aplicaciones y la forma de

combatirlos. En particular, hemos aprendido que es imprescindible el uso de contraseñas largos hechos de tanto letras mayúsculas y minúsculas, números y símbolos.

Pruebe a escribir una contraseña tan fuerte en un dispositivo móvil. Usted tiene que cambiar varias veces entre las diferentes distribuciones de teclado; al final, todos los usuarios podrían optar por una contraseña simple, y los desarrolladores de aplicaciones optarían por los PIN de números solamente. Todo esto es eficaz, pero se reduce el nivel de seguridad.

Además, tenga en cuenta que un dispositivo móvil es mucho más fácil robar que un ordenador portátil y también es mucho más fácil de perder. Además, un dispositivo móvil a menudo se presta a otras personas, aunque sólo sea por algo sencillo y rápido como una llamada telefónica. Todos estos comportamientos comunes aumentan el riesgo de ataques y pérdida de datos.

El reto es encontrar una buena mezcla de medidas que mantienen la seguridad en un nivel lo suficientemente alto sin hacer vida de los usuarios mucho más difícil. La seguridad móvil, sin embargo, es un nuevo campo de investigación de la misma manera que la seguridad web era en la década de 1990.

1.1.3.2. Plataformas

1.1.3.1.5. Múltiples plataformas

El ecosistema móvil está poblado por varias plataformas diferentes, cada uno de los cuales tiene su propio más o menos conjunto único de características y capacidades. Las plataformas más populares hoy en día son iPhone, iPad,

Android, Windows Phone y BlackBerry. La lista de las plataformas, sin embargo, no termina aquí.

Otras plataformas que es probable que encuentre o que considerar son Symbian, Windows Mobile, Meego, Bada, QT, y webOS. Y cuando usted comienza a buscar en el uso de dispositivos de tableta, la gama de plataformas que puede que tenga que tener en cuenta crece aún más, ya que hay variaciones en tabletas específica de las plataformas antes mencionadas, incluyendo Android Honeycomb, BlackBerry PlayBook, y el próximo Windows 8.

Cada plataforma tiene su propio sistema operativo, su propia interfaz de programación de aplicaciones (API), y su propio conjunto de directrices de programación. A menudo, cada plataforma móvil requiere estar escritos en un lenguaje de programación específico, como Java, Objective-C, C # o C + +.

¿Significa esto que usted debe desarrollar su aplicación a partir de cero para cada una de estas plataformas?

Francamente, muy pocas aplicaciones (por ejemplo, proveedores de contenido) deben abordar todas estas plataformas. Más típicamente, las aplicaciones se dirigen a un subconjunto de no más de tres o cuatro de ellos. Si es crucial para su negocio para llegar a la mayor audiencia posible, incluso aquellos que se ejecuta en dispositivos de gama baja, entonces es posible que desee mirar a HTML-específicamente HTML5 para construir un sitio web optimizado para dispositivos móviles (es decir, un sitio móvil web). Sitios móvil web son a menudo la primera opción que usted debe considerar cuando la orientación de múltiples plataformas es una verdadera necesidad de la empresa. Sin embargo, no

están libres de problemas en el dispositivo, ya sea. Al final, la construcción de un sitio móvil puede ser considerablemente más complicada que la construcción de un sitio web.

1.1.3.1.5.1. Desarrollando para IOs:

En estos días, muchos directores de tecnología (CTO) están luchando para definir la estrategia ideal para integrar el desarrollo móvil en sus plataformas. Una razón por la que el desarrollo de soluciones móviles es problemático es que aún hoy en día, la palabra móvil a menudo sólo significa "iPhone y iPad" para muchos ejecutivos.

El Software móvil tiene muchas facetas, una de las más convincente es, probablemente, es una aplicación para el iPhone. Tenemos ejecutivos que están impresionados por una aplicación novedosa de iPhone que han visto y ahora quieren replicarla.

Tenemos directores de programas impresionados por la cantidad de trabajo (y dinero) que se necesitaría para proporcionar una solución móvil integral que llegue a una amplia audiencia. Y luego tenemos a los usuarios móviles.

Los usuarios de móviles sólo tienen que utilizar sus dispositivos; disfrutan de aplicaciones y sitios con tal de que sean fáciles de usar, están obligando de alguna manera, y responden a una necesidad específica, tal vez una necesidad no profesional, pero sigue siendo una necesidad.

Los usuarios de móviles no tienen demandas explícitas, pero son extremadamente selectivos y no particularmente indulgentes. En el espacio móvil, los usuarios ya no son víctimas de las manías y obsesiones de los desarrolladores.

En el desarrollo móvil, el usuario es el rey, y las aplicaciones se hacen para agradar al usuario.

En este contexto, iOS-el sistema operativo móvil a cargo de iPhone, iPod Touch, e iPad, contribuyó significativamente hacia el establecimiento de unas normas de facto para la tecnología móvil, facilidad de uso y capacidades de aplicación. Hoy en día, una solución móvil a veces consiste únicamente en un sitio móvil. Sin embargo, cada vez que alguien va más allá del nivel de un sitio móvil, entonces se construye una aplicación nativa de iOS.

1.1.3.1.5.2. Desarrollando con Android

Nos guste o no, uno de los factores clave para el éxito de los iOS es sin duda que se trata de un ambiente cerrado. Hay sólo unas pocas configuraciones de hardware y un sistema operativo. Ambos aspectos están estrictamente controladas por Apple. Como usuario, usted recibe una notificación de actualizaciones del sistema directamente de Apple; Como desarrollador, usted tiene sólo un homólogo con quien hablar (y a veces pelear).

En concreto, esto significa que, como desarrollador, no enfrenta el problema de la adaptación de la interfaz de usuario y la lógica de presentación a diferentes resoluciones y hardware diferente o, al menos, sólo hay tres opciones para tratar con: iPhone, iPod Touch , y IPAD. La mayor parte del tiempo, todo esto se reduce a dar algunas aplicaciones (ni siquiera todas las aplicaciones) con una interfaz de usuario diferente en el iPad para que puedan tomar ventaja de la pantalla más grande. Al hacerlo, sin embargo, en realidad está empezando otro proyect, con un alto grado de reutilización de código. Dentro de la misma aplicación, que rara vez se

necesita para adaptarse a diferentes configuraciones de la interfaz de usuario y el hardware.

En iOS, usted todavía puede distinguir un par de versiones diferentes del sistema operativo, y posteriormente unos niveles variables de la interfaz de programación de aplicaciones (API). Esto significa que una aplicación escrita para iOS 4, no podrán ejecutarse en un iPod Touch bastante antiguo, pero, francamente, hay pocas diferencias más allá de este punto. El panorama iOS definitivamente no está fragmentado.

En Android, te enfrentas a una situación completamente diferente.

Google comenzó a Android como un proyecto de código abierto y, a diferencia de Apple, Google no obliga a una versión única y no modificable del sistema operativo y la configuración del hardware. Como resultado, la gama de diferentes dispositivos Android que su aplicación puede tener que ejecutar en él es bastante amplia. Esto hace que el desarrollo de Android sea desafiante y potencialmente más costoso.

1.1.3.3. Aplicaciones Móviles vs Sitios Móviles

La era moderna de la tecnología móvil comenzó con el lanzamiento del primer iPhone de Apple en el verano de 2007.

Los Moviles (Mobile) ha sido anunciada como la próxima gran cosa durante una década. Desde finales de 1990, se ha pronosticado en varias ocasiones como el vehículo para una gran innovación que va a afectar a nuestros estilos de vida. Aunque la adopción de dispositivos móviles ha sido siempre más allá de una masa crítica significativa, la

previsión inicial se hizo realidad hace tan sólo unos años, comenzando con el lanzamiento del iPhone en 2007.

Inevitablemente, la publicidad sobre telefonía móvil siempre ha estado ligada a (y sesgados por) el uso de un teléfono inteligente con su propio conjunto de aplicaciones nativas. Pero al principio, cuando primero se hacían previsiones sobre la próxima revolución móvil, móvil significaba algo completamente diferente; significaba tener sitios web optimizados para dispositivos móviles.

Hoy en día, los sitios y aplicaciones móviles nativas son muy a menudo enfrentadas entre sí, al igual que el programa de mano de un combate de boxeo sensacional. ¿Es esta una representación correcta de la realidad? Está poniendo una "contra" en otro un duelo justo? Vamos a ver.

Para anticipar: La respuesta rápida es que ambos sitios y aplicaciones móviles nativas tienen plena razón de existir. El enfoque de cabeza a cabeza es sobre todo una interpretación forzada de sus diferencias. Es muy común encontrar dos opciones (sitios móviles y de aplicaciones nativas) implementados en una estrategia móvil de clase empresarial.

1.1.3.1.6. No es cuestión insustancial

Las aplicaciones nativas y sitios móviles representan diferentes maneras de implementar una solución móvil (en su totalidad o en parte). El análisis y la comprensión de los pros y los contras de cada enfoque es definitivamente una buena cosa, pero acercándose a los temas como la contraposición entre dos enfoques mutuamente excluyentes no tienen sentido. Ser un defensor de uno de los campamentos está muy bien; prefiriendo un enfoque sobre el otro también está bien; tomar una decisión de negocios puro y duro no es nunca una cuestión de preferencia. Si se entiende la mecánica de las

aplicaciones nativas y sitios móviles, siempre hay una mejor ruta a tomar.

1.1.3.1.7. Un falso dilema -Pero los verdaderas diferencias

Si escribe palabras clave como "apps nativas frente a los sitios móviles" en cualquier motor de búsqueda, obtendrá un montón de enlaces, muchos de los que discuten las ventajas y desventajas de cada método de acuerdo con la visión del autor en particular. La mayoría de los argumentos tienen sentido, y recomiendo que lea el contenido de algunos de los primeros enlaces que Google y Bing devuelven en respuesta a esa búsqueda.

El punto real, sin embargo, es diferente: la comparación de los sitios móviles vs aplicaciones nativas es generalmente una falsa dicotomía, pero los sitios y aplicaciones móviles nativas representan dos opciones válidas para una solución móvil. Existen verdaderas diferencias entre los sitios y aplicaciones móviles nativas, y la comprensión de ellos es la clave para la localización de los pilares de su solución móvil.

1.1.3.1.8. Centrarse en la pregunta correcta

Contrastando sitios de telefonía celular con aplicaciones nativas no es la forma correcta de acercarse a este tema, ya que el enfoque intenta encontrar una respuesta dura a una pregunta irrelevante. Para los desarrolladores que leen esto, el tema no es muy diferente a una lluvia de ideas si JavaScript Object Notation (JSON) o XML es preferible cuando se construye una aplicación distribuida. Durante el proceso de creación de la aplicación, es probable que haya un momento en que los arquitectos o desarrolladores necesitan tomar una decisión en cuanto a si se debe emplear JSON o XML, pero las preguntas más críticas para resolver son, por ejemplo, cómo diseñar la interfaz pública de sus servicios [es decir, de

estado representacional de transferencia (REST) vs simple Object Access Protocol (SOAP)]; qué protocolos de transporte deben participar; si la seguridad y transacciones importantes; y así sucesivamente.

Volver a móvil: Si "aplicaciones nativas vs sitios móviles" no es la cuestión más relevante, entonces ¿qué es?, es necesario centrarse en los productos, servicios, y el público para el que se está construyendo. A continuación, usted detallar los objetivos y se centrarse en el presupuesto, los recursos y el cronograma.

Cuando todo esto está claro, puede pasar a la parte arquitectónica del proyecto. En ese momento, probablemente descubrirá que tiene una sola opción que queda.

En cualquier caso, existen algunas diferencias verdaderas entre una aplicación nativa y un sitio móvil. Las siguientes secciones se centran en los principales rasgos de ambos.

1.1.3.1.9. Los principales rasgos de las aplicaciones nativas

Una aplicación nativa debe jugar según las reglas del sistema operativo en el que se encuentra. Esto significa que es objeto de restricciones (por ejemplo, las capacidades limitadas de almacenamiento y capacidad de procesamiento). Por otro lado, tiene la ventaja de ser capaz de utilizar la ubicación y los servicios integrados tales como los servicios de notificación de inserción, servicio de mensajes cortos (SMS), y la cámara, utilice todas las capacidades de la interfaz de usuario táctil, y se basan en un mecanismo de instalar / actualizar sin problemas.

Muchas personas tienden a justificar su estancia en la "aplicación nativa" con el presunto hecho de que los usuarios prefieren las aplicaciones nativas. En general, usted debe

tomar esto como una opinión arbitraria. Aunque una aplicación nativa (o falta de ella) sin duda puede tener un impacto en la comercialización de su marca, en términos de funcionalidad y características, aquí están las principales características:

- Una aplicación nativa tiene la mejor oportunidad para integrarse bien con el dispositivo y utilizar hardware y servicios de software integradas.
- Una aplicación nativa está mucho menos expuesto a la latencia de la red.
- Una aplicación nativa puede ser capaz de trabajar en equipos totalmente desconectados.
- Una aplicación nativa es un programa en el mismo recinto, por lo que no requiere que los usuarios lidien con URLs.
- Una aplicación nativa normalmente ofrece una experiencia de usuario nativa.
- Los usuarios descargan aplicaciones nativas como un paquete que lo abarca todo en una sola solicitud de red.
- Una aplicación nativa se debe crear para cada plataforma móvil que tiene la intención de apoyar.

1.1.4. Arquitectura de Software

Los grandes sistemas siempre se descomponen en subsistemas que proporcionan algún conjunto de servicios relacionados. El proceso de diseño inicial que identifica estos subsistemas y establece un marco para el control y comunicación de los subsistemas se llama diseño arquitectónico. El resultado de este proceso de diseño es una descripción de la arquitectura del software (Sommerville, 2005).

El diseño arquitectónico es la primera etapa en el proceso de diseño y representa un enlace crítico entre los procesos de ingeniería de diseño y de requerimientos. El proceso de diseño arquitectónico está relacionado con el establecimiento de un marco estructural básico que identifica los principales componentes de un sistema y las comunicaciones entre estos componentes. Constituye el estilo arquitectónico que tendrá el sistema, la estructura y las propiedades de los componentes que ese sistema comprende, y las interrelaciones que tienen lugar entre todos los componentes arquitectónicos del sistema, nos proporciona una visión global del sistema a construir (Pressman, 2002).

Un modelo de arquitectura del sistema es una descripción compacta y manejable de cómo se organiza un sistema y cómo interoperan sus componentes. La arquitectura del sistema es a menudo la misma para sistemas con requerimientos similares y, por lo tanto, pueden soportar reutilización del software a gran escala. La arquitectura del sistema afecta el rendimiendo, solidez, grado de distribución y mantenibilidad de un sistema (Bosch, 2000). Constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes.

El resultado del proceso de diseño arquitectónico es un documento de diseño arquitectónico. Éste puede incluir varias representaciones gráficas del sistema junto con texto descriptivo asociado.

El modelo de diseño arquitectónico y los patrones arquitectónicos contenidos dentro son transferibles. Esto es, los estilos y patrones de arquitectura pueden ser aplicados en el diseño de otros sistemas y representados a través de un conjunto de abstracciones que facilitan la descripción de la arquitectura de un modo predecible.

Toda arquitectura de software debe describir diversos aspectos del software, como por ejemplo el lenguaje de programación a usar, los tipos de datos que se requieren, etc. Generalmente, cada uno de estos aspectos se describe de

una manera comprensible, utilizando distintos modelos o vistas. Es importante destacar que cada uno de ellos constituye una descripción parcial de una misma arquitectura, y es deseable que exista cierto solapamiento entre ellos. Cada paradigma de desarrollo exige diferentes tipos de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas fundamentales en cualquier arquitectura:

- La vista estática: describe qué componentes tiene la arquitectura.
- La vista funcional: describe qué hace cada componente.
- La vista dinámica: describe cómo se comportan los componentes a lo largo del tiempo, y cómo interactúan entre sí.

Las vistas o modelos de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes, tales como los diagramas de estado, los diagramas de flujo de datos, etc. Estos lenguajes son apropiados únicamente para un modelo o vista.

Afortunadamente, existe cierto consenso en adoptar UML (Unified Modeling Language, lenguaje unificado de modelado) como lenguaje único para todos los modelos o vistas. Los modelos informales y las notaciones tales como UML (Bass, Clements, & Kazman, Software Architecture in Practice, 2002) son las notaciones más comúnmente usadas para la descripción arquitectónica. El UML es el lenguaje gráfico para modelado de sistemas de software más conocido y utilizado actualmente, se usa para visualizar, especificar, construir y documentar un sistema de software. Algunos de los diagramas que propone el lenguaje UML son:

- Los diagramas de estructura, que enfatizan los elementos que deben existir en el sistema modelado, como las clases, componentes, objetos, etc.
- Los diagramas de comportamiento, que se enfocan en lo que debe suceder en el sistema modelado, como por ejemplo las actividades, casos de uso y estados.

- Los diagramas de interacción, los cuales pueden considerarse como parte de los diagramas de comportamiento. Estos muestran el flujo de control y de datos entre los elementos del sistema modelado, así como la colaboración e interacciones entre ellos, entre otras cosas.

1.1.4.1. Líneas de producto

Uno de los objetivos en el desarrollo de cualquier sistema, es la reutilización de código o componentes de software, con el objetivo de reducir la carga de trabajo, el tiempo de desarrollo y los costos. Una arquitectura de software representa una inversión de tiempo y esfuerzo considerable y es diseñada por expertos; de tal forma que es deseable maximizar los recursos de la inversión tratando de reutilizar ésta en varios sistemas. Éste es el objetivo de una línea de producto de software (Bosch, 2000).

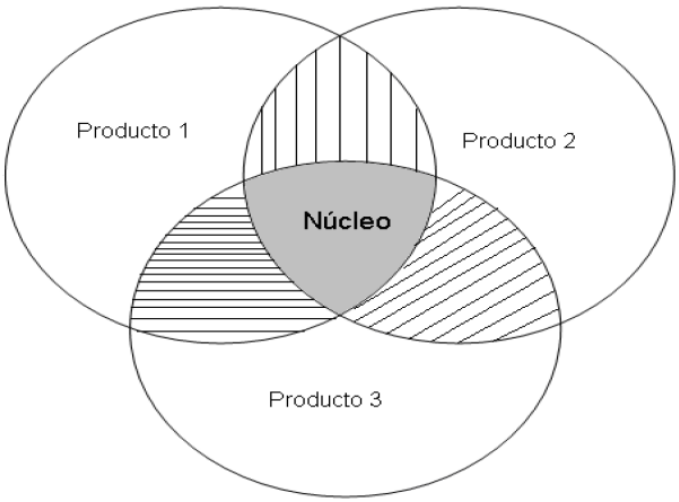


Ilustración 1: Líneas de producto

1.1.4.1. Elementos de una Línea de Productos

La idea de reutilizar elementos para generar diversos productos ya es usada de forma exitosa en la industria manufacturera por fabricantes de productos

(electrónicos, automotrices), los cuales diseñan un conjunto de elementos base, o núcleo, que posteriormente se reutiliza para crear nuevos productos. La intersección de los elementos comunes de los distintos productos caracteriza al núcleo. En el caso particular de la industria de software, el desarrollo de líneas de producto de software requiere de técnicas de ingeniería de software y administración de desarrollo de sistemas, para crear una colección similar de sistemas de software similares entre ellos, y que comparten un núcleo de software común en su producción. En el software, el núcleo común frecuentemente engloba las decisiones arquitecturales. Crear una línea de producto de software exitosa depende de una estrategia coordinada que involucra ingenieros de software, equipos técnicos, la estructura y políticas de la organización en cuestión (Bass, Clements, & Kazman, Software Architecture in Practice, 2002).

1.1.4.2. Papel de la arquitectura de software en la especificación de un sistema

El buen diseño de una arquitectura es esencial para lograr componentes de software reutilizables, es decir que se utilicen en varias aplicaciones. En este sentido la arquitectura de software impacta en los siguientes aspectos:

- Define la estructura de un sistema: al diseñar componentes con comportamiento bien específico, el arquitecto define responsabilidades y las tareas más importantes de cada componente. En este sentido, cada componente tiene asignado un rol en la aplicación final. También es importante destacar que se deben minimizar las dependencias entre componentes generando interfaces que faciliten un bajo acoplamiento entre ellos. De esta forma al remplazar algún componente los cambios están bien identificados y las dependencias son mínimas evitando la propagación de errores entre componentes.

- Especifica mecanismos efectivos de comunicación entre componentes: debido a que la arquitectura está compuesta por múltiples componentes, es necesario crear o utilizar mecanismos de control e intercambio de datos entre los componentes. No obstante, existe un conjunto de patrones o estilos arquitectónicos ya probados y validados en cierto dominio de aplicaciones que utilizan mecanismos de comunicación bien definidos. El arquitecto puede entonces apoyarse en algún estilo arquitectónico predefinido (por ejemplo Cliente-Servidor, el cual define los mecanismos de comunicación entre componentes de forma adecuada) para plantear la arquitectura. También es muy importante destacar que la elección de un estilo arquitectónico particular tiene impacto directo en determinados requerimientos no funcionales.
- Satisface los requerimientos no funcionales (NFR´s Non Functional Requirements) de la aplicación final: Este tipo de requerimientos definen como se deben cubrir o alcanzar los requerimientos funcionales de la aplicación final. Estos requerimientos se detallan más adelante.
- Crea un medio efectivo de comunicación entre todos los involucrados: Cuando se identifican los elementos estructurales de la arquitectura (componentes) y las propiedades externas y visibles (comunicación de los componentes, características de desempeño, recursos compartidos, entre otros.) se está en la posibilidad de crear un modelo con diferentes vistas, las cuales especifican diferentes niveles de descripción de la complejidad de la arquitectura. Sin embargo cada involucrado tiene determinado conocimiento del sistema por lo que utilizará la vista que le proporcione la información de su interés de la forma más clara y sencilla.

1.1.4.2. Ventaja de la arquitectura de software

Las ventajas principales que se obtienen al realizar la arquitectura de un sistema son:

- Reducción de los riesgos técnicos y relativos a la calidad asociados a la implementación del sistema: se identifican componentes y patrones arquitectónicos validados y probados los cuales sirven de entrada al diseño del sistema.
- Representa una decisión temprana de diseño: una vez definida la arquitectura, es posible generar prototipos para explorar las consecuencias de las decisiones de diseño.
- Es una abstracción transferible hacia otros sistemas: se puede reutilizar el diseño de una arquitectura en sistemas que exhiban requerimientos no funcionales similares. De esta forma se reduce el tiempo y los costos para realizar nuevos sistemas.
- Facilita el análisis más preciso de costos y estimación de tiempos: el cálculo de costos y el cálculo de tiempos para los proyectos de software es una actividad muy importante en la construcción de un sistema; ya que permite estimar los recursos necesarios (humanos, tecnológicos, etc.) para la implementación. Al plantear una arquitectura para el sistema se puede conocer mejor la complejidad del sistema y de esta forma estimar los recursos de forma más precisa.

1.1.4.3. Vistas de Arquitectura de Software

Una vista de la arquitectura es una manera de describir los aspectos o elementos de la arquitectura que son relevantes a los elementos que se intenta abordar. Las vistas permiten hacer una descripción de una arquitectura de software. Una vista está conformada por un conjunto de modelos representando la arquitectura, es común utilizar UML para realizar estos modelos.

Varios modelos han sido propuestos para crear la documentación de una arquitectura, realizando separación de intereses. Cada modelo describe un conjunto de vistas e identifica de qué debe encargarse cada una. Los

diferentes modelos cubren el mismo dominio de arquitectura de software, incluyendo elementos organizacionales, de negocio y tecnológicos, cada uno con sus ventajas y desventajas.

Es posible combinar las vistas de los diversos modelos y determinar un conjunto que se adapte mejor a una arquitectura que se desee describir.

1.1.4.2.1. Modelo 4+1 de Kruchten

Este modelo consiste de múltiples vistas concurrentes que permiten satisfacer las necesidades de diferentes interesados de forma separada. El modelo incluye 5 vistas.

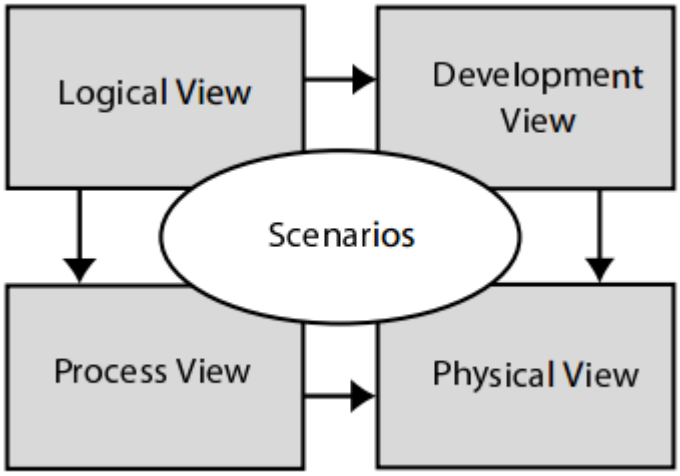


Ilustración 2: Modelo de vistas 4+1 de Kructhen

El modelo es genérico así que las vistas pueden ser descritas utilizando diferentes notaciones o métodos de diseño. el modelo incluye además detalles de cómo las vistas están relacionadas y el proceso de desarrollo de un conjunto completo de vistas.

Kruchten previó un proceso iterativo para diseño arquitectónico, empezando por la descripción de escenarios críticos. El arquitecto puede entonces identificar abstracciones clave del dominio del problema y modelarlo en la vista lógica.

Las clases lógicas pueden ser mapeadas a módulos y paquetes en la vista de desarrollo, y las tareas y procesos en la vista de procesos. Finalmente, los procesos y módulos pueden ser mapeados a hardware en la vista física. En cada iteracción subsecuente escenarios adicionales son modelados, siguiendo esta secuencia, hasta que la arquitectura se torne estable. Esto usualmente ocurre cuando no son descubiertas nuevas abstracciones, subsistemas, procesos o interfaces.

Las vistas no son totalmente independientes, la vista de escenarios tiene la función de unir y relacionar las otras 4 vistas.

1.1.4.2.2. Modelo SEI

Este modelo ha sido bien documentado y revisado, detallando y describiendo los procesos para la elección de vistas y documentación de la arquitectura. El proceso de documentación consiste en documentar las vistas relevantes y cualquier información adicional que correspnoda a más de una vista. Sin embargo, algunas vistas son muy complejas para ser mostradas en ona representación, así que puede ser dividida en un número de paquetes.

El modelo SEI incluye plantillas para los contenidos de una vista. Consiste en la representación básica, catálogo de elementos, diagrama de contexto, guía de variabilidad, background arquitectónico, información no técnica y la relación con otras vistas.

La lista de estilos y vistas que pueden ser usados para representar el sistema se basa en la estructura del mismo sistema.

1.1.4.2.3. Modelo de 4 vistas de Siemens

Este modelo es el resultado del estudio de las prácticas industriales de arquitectura de software. Los autores encontraron que las estructuras usadas para diseñar y documentar una arquitectura de software caen en 4 categorías básicas, las cuales ellos llaman estructuras conceptual, modular, de ejecución y de código. Cada categoría atiende los intereses de diferentes grupos.

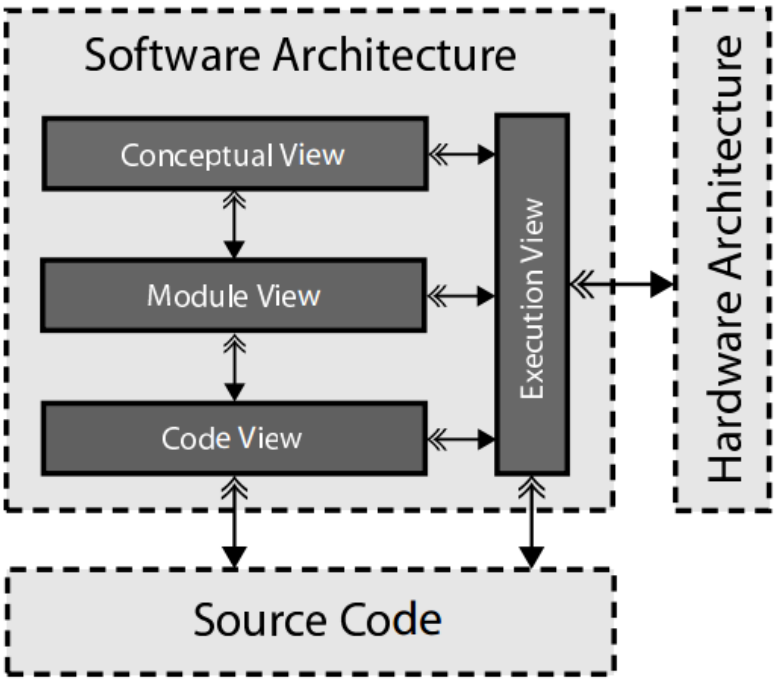


Ilustración 3: Modelo Siemens de 4 vistas

Adicionalmente, encontraron que cuando las categorías son brindadas separadamente, la complejidad de implementación de los sistemas decrece y se mejora la reutilización y la reconfiguración. Para separar las categorías, proponen que la arquitectura de software debería ser documentada utilizando cuatro vistas, cada una de las cuales atiende diferentes categorías de estructuras

Las cuatro vistas de este modelo presentan bajo acoplamiento. El flujo de diseño sigue la información pasada entre vistas empezando por el modelo conceptual. El feedback resulta de probar las vistas y su conformidad con los requerimientos no funcionales del sistema.

1.1.4.2.4. Especificación para Descripción Arquitectónica Rational (ADS)

La especificación para descripción arquitectónica (ADS) es una expansión del modelo 4+1 para permitir la descripción de arquitectura más complejas, como empresariales, e-business, sistemas embebidos y sistemas no otro tipo, no de software. Fue definida por primera vez en Noviembre de 2000 y actualmente es parte del curso de Rational "Principles of Architecting Software Systems.

Muestra una definición formal de la evolución de los requerimientos y utiliza la notación UML cuando es posible.

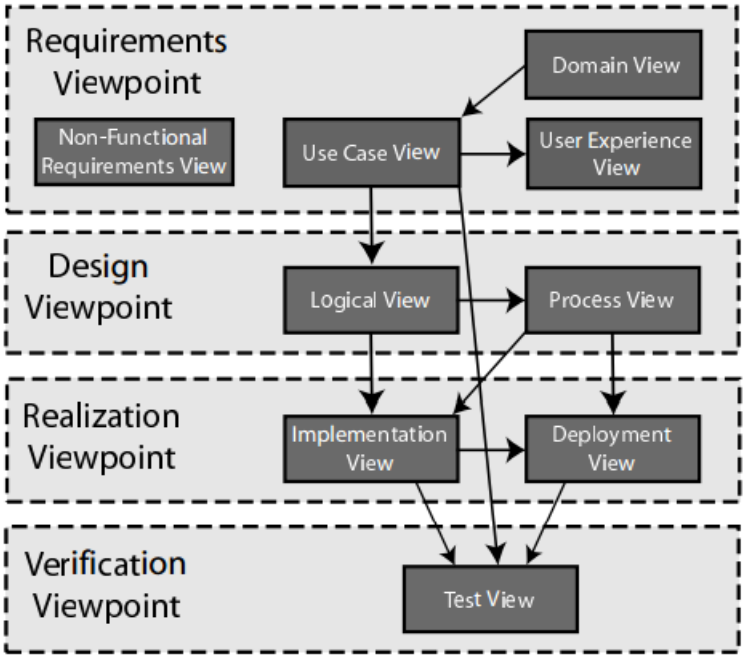


Ilustración 4: Modelo ADS

Las vistas del modelo 4+1 han sido renombradas parcialmente y se definieron 4 nuevas vistas. La vista de escenarios se convierte en la vista de Casos de Uso, la vista de desarrollo se convierte en la vista de Implementación y la vista física se convierte en la vista de Despliegue. Las nueve vistas han sido agrupadas en cuatro bloques

1.1.4.4. Arquitectura de aplicaciones Android

Existen investigaciones o proyectos en los que, directa o indirectamente, se presenta formas de estructurar las aplicaciones móviles desarrolladas bajo la plataforma Android, aprovechando de la mejor manera los componentes que brinda y facilitando tareas de diseño y desarrollo de dichas aplicaciones. Podemos mencionar las siguientes:

1.1.4.4.1. Android REST client

Google, en su conferencia anual de desarrolladores Google I/O del año 2010, presentó 3 diseños de arquitecturas bases, muy similares entre ellas, que sugiere utilizar en el desarrollo de aplicaciones móviles en Android que realicen invocaciones a servicios web REST. En ella tocan temas como los componentes a utilizar, la estructura y los mecanismos de comunicación entre dichos componentes.

Esta arquitectura se concentra principalmente en el modelo de intercambio de datos, específicamente cuando se utilizan web services del tipo REST, esta es una decisión fundamental desde el punto de vista de arquitectura de una aplicación (Dobjanschi, 2010).

Google presentó 3 modelos para realizar invocaciones REST:

Uso del Service API

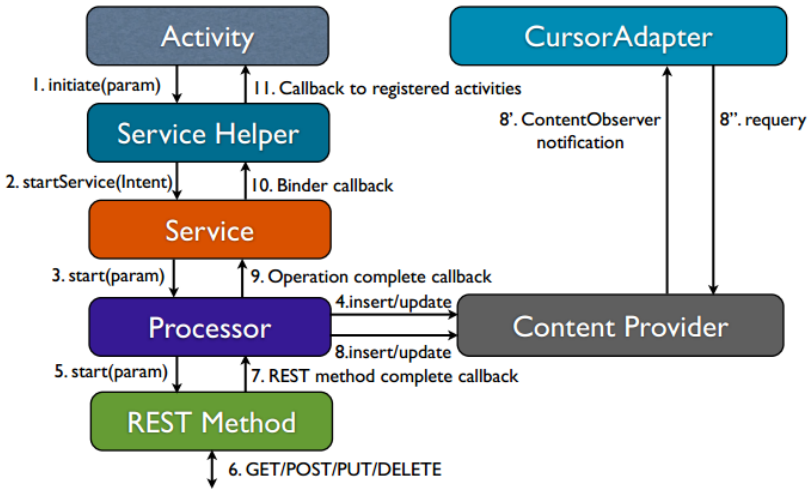


Ilustración 5: Uso del Service API

Esta solución está basada en el código de las plataformas Android 1.x así que no usa mecanismos introducidos en nuevas versiones. Desde la base de la arquitectura el componente REST se encarga de la creación, envío y recuperación de objetos por el protocolo HTTP. Especifica tipos de dato e implementa una conexión, así como un parser. La data puede ser enviada en diferentes formatos como JSON o XML.

El segundo componente (Processor) es responsable de sincronizar la memoria local con la fuente externa (la que usa el componente REST). Agregando flags a cada objeto transferible el desarrollador puede tener una descripción del estado actual de cada objeto particular en cuanto a la sincronización. El processor almacena los datos utilizando un componente ContentProvider el cual mayormente utiliza una base de datos, pudiendo usarse también la memoria externa.

El siguiente componente es el Service, el cual se ejecuta en el contexto de la aplicación. Este Service inicia tareas largas (Processor) en background y es independiente del ciclo de vida del Activity, así que todas las peticiones pueden ser manejadas aún si el usuario decide dejar el Activity o algún eveno llama su atención (como un email entrante). El servicio puede usar el Intent API para recibir intents de la capa superior y delegarlos al componente Processor. Después de que la tarea es completada notifica a la capa superior llamando al servicio adecuada mediante un callback. En este caso el Service es un componente sin estado así que no necesita almacenar su data persistentemente. Más importante, en cuanto el Service termina su tarea debería ser terminado.

Implementar una capa intermedia con un singleton que exponga un API simple es un patrón común en el desarrollo de aplicaciones multicapa. El rol del ServiceHelper es sólo transmitir peticiones y data entre Activities y el Service. Sin embargo, es importante reducir el número de peticiones innecesarias, para resolver esto el ServiceHelper puede preguntar al Service en ejecución si la operación correspondiente está pendiente y evitar duplicados.

El último componente en esta arquitectura es el Activity, el cual simplemente invoca al ServiceHelper cuando precisa de una operación REST. Registra ciertos listener que son invocados por el ServiceHelper cuando termina una operación REST.

Uso del ContentProvider API

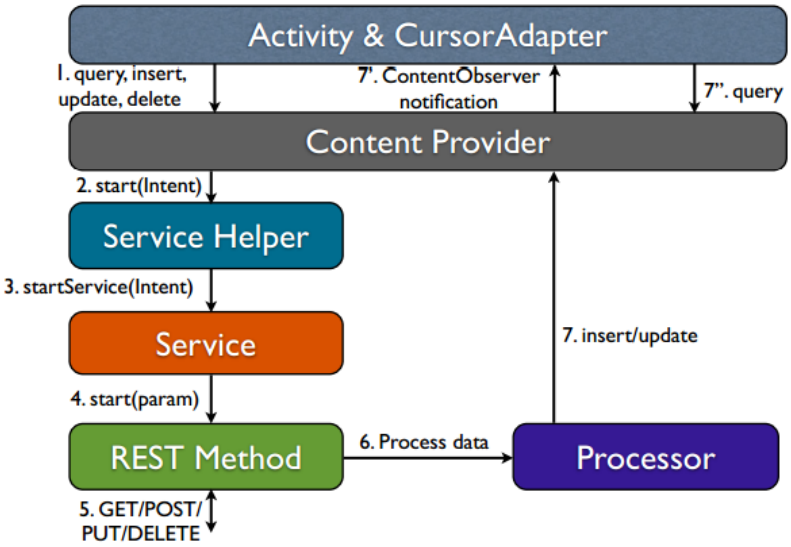


Ilustración 6: Cliente REST – Uso del ContentProvider API

Esta solución no introduce nuevos componentes con respecto a la anterior, simplemente utiliza al máximo el ContentProvider y su correspondencia exacta con las operaciones REST estándares (en pares: querys al ContentProvider - REST GET, Insert - POST, Delete - DELETE, Update - PUT). En esta solución los Activity invocan al ContentProvider apropiado el cual ejecuta sus tareas, dejando la data en un estado de transición. Sin embargo, antes de terminar, invoca al ServiceHelper y realiza la petición de una nueva operación REST. Desde este momento todo funciona exactamente igual que en la primera solución. El camino de retorno puede acortarse llamando al Processor directamente después de que termina el método REST. En este puno no tiene sentido pasar el mensaje de retorno a través del Service y ServiceHelper, pero en cambio el Processor puede invocar directamente al ContentProvider para cambiar los datos o simplemente actualizar su estado

actual. Tan pronto como el ContentProvider detecta una modificación en sus datos invoca a todos sus ContentObservers con un mensaje de notificación. El Activity o cualquier componente que dependa de la data devuelta en un cursor del ContentProvider pueden llamar al método requery y refrescar la data presentada al usuario.

Como esta solución cambia la responsabilidad del ServiceHelper y Service al ContentProvider, también es su trabajo encargarse de errores en la red. Cuando un método REST falla el ContentProvider debe mantener esto en memoria y reintentar la petición.

Uso del ContentProvider API y un SyncAdapter

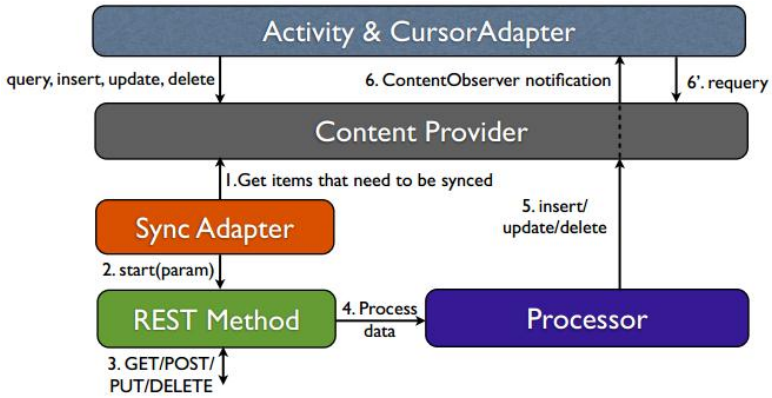


Ilustración 7: Cliente REST – Uso del ContentProvider API y un SyncAdapter

La tercera solución hace uso de un SyncAdapter, mecanismo introducido en Android 2.0. Aún cuando está presente desde hace varias versiones, muchos desarrolladores aún no conocen bien el concepto y sólo una fracción de ellos lo utiliza en sus aplicaciones.

El SyncAdapter reemplaza al Service y el serviceHelper. Un Activity invoca al ContentProvider apropiado y ejecuta las

peticiones instantáneamente, posiblemente dejando algunos registros en un estado de transición. Antes de terminar un método requestSync es invocado en un ContentResolver, añadiendo la petición en una cola del SyncManager. Cuando el SyncManager decide ejecutar la petición lanza el método onPerformSync. Usualmente implementaciones del SyncAdapter requerirían consultar al ContentProvider por todas las filas que necesitan ser sincronizadas, crear una nueva petición y ejecutar el componente REST pasándolo como parámetro. Después de que termina la operación de red, se inicia la sincronización apropiada, teniendo ambas respuestas del servidor remoto y el acceso local al ContentProvider. Los Activity son notificados de los cambios por listeners de ContentObservers.

1.1.4.4.2. RoboSpice

RoboSpice es una librería para Android enfocada en facilitar la ejecución de tareas largas de manera asíncrona. Brinda una forma de estructurar estas tareas para facilitar el desarrollo. Es en parte una implementación del diseño presentado por Google en el Google I/O 2010 para clientes REST, sin embargo puede ser utilizado para una variedad de casos mayor. Una librería muy utilizada actualmente en diversas aplicaciones.

Robospice se basa en una arquitectura robusta para realizar sus invocaciones REST (Nicolas, Robospice Wiki, 2014).

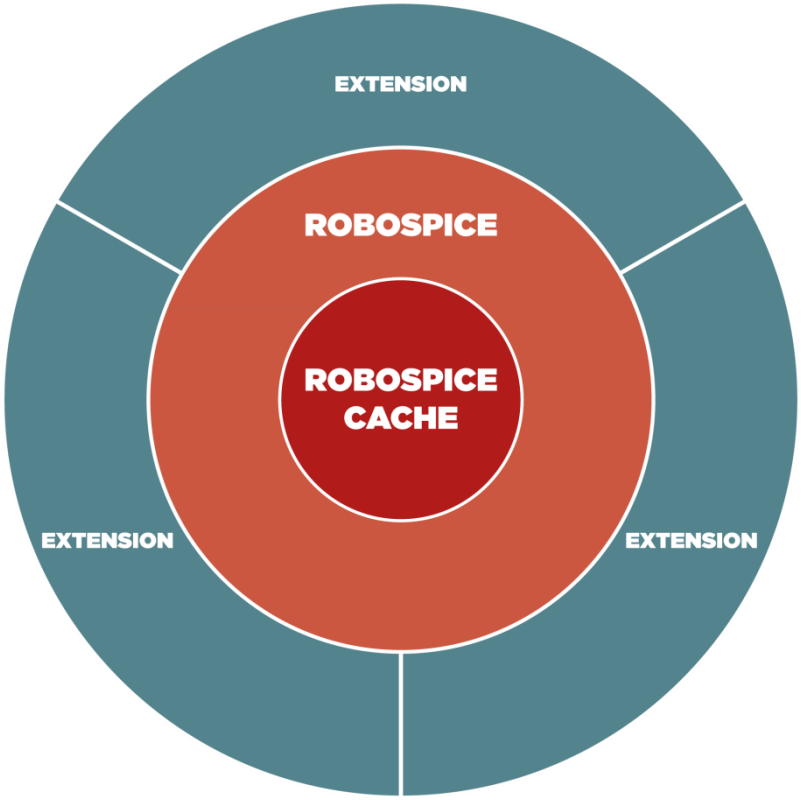


Ilustración 8: Arquitectura de RoboSpice

La arquitectura de esta librería se basa en módulos. El módulo de RoboSpice Cache es el núcleo de la librería. Este módulo se encarga de la persistencia de los POJOs. Cuenta con las clases ObjectPersister, que se encarga de grabar y cargar data y CacheManager, que mantiene una lista de instancias de ObjectPersister, a los que delega las tareas de persistencia. El ObjectPersisterFactory se encarga de crear instancias de ObjectPersister según sean requeridas.

El módulo RoboSpiceCore es el núcleo conceptual de la librería. Contiene todo lo relacionado a ejecutar peticiones, uso de Services para procesar las peticiones y obtener los resultados y errores. Como la librería está pensada para tareas largas en segundo plano se decidió utilizar Services. Usa la clase SpiceRequest como base para realizar las tareas en segundo plano, la cual se ejecutará en el

Service. También está el RequestListener, el cual permite notificar sobre el procesamiento de las peticiones. Y SpiceService, que es el Service que puede realizar los procesos, responsable también de encolar las peticiones. Por último está el SpiceManager, el objeto que se encarga de ejecutar las peticiones.

También estan las extensiones, que son módulos adicionales que tiene el framework para temas como persistencia o uso de librerías Spring.

1.1.4.4.3. Arquitectura Yamba

En el libro Learning Android, escrito por Marko Gargenta, además de brindar una explicación de los componentes básicos de la plataforma Android, guía paso a paso el desarrollo de una aplicación de mediano tamaño que funciona como cliente de Twitter. El autor presenta sin llegar a mucho detalle una estructura para la aplicación que va desarrollando, la cual aunque no es muy completa puede ser utilizada como base para la implementación de nuevas aplicaciones (Gargenta, 2011).

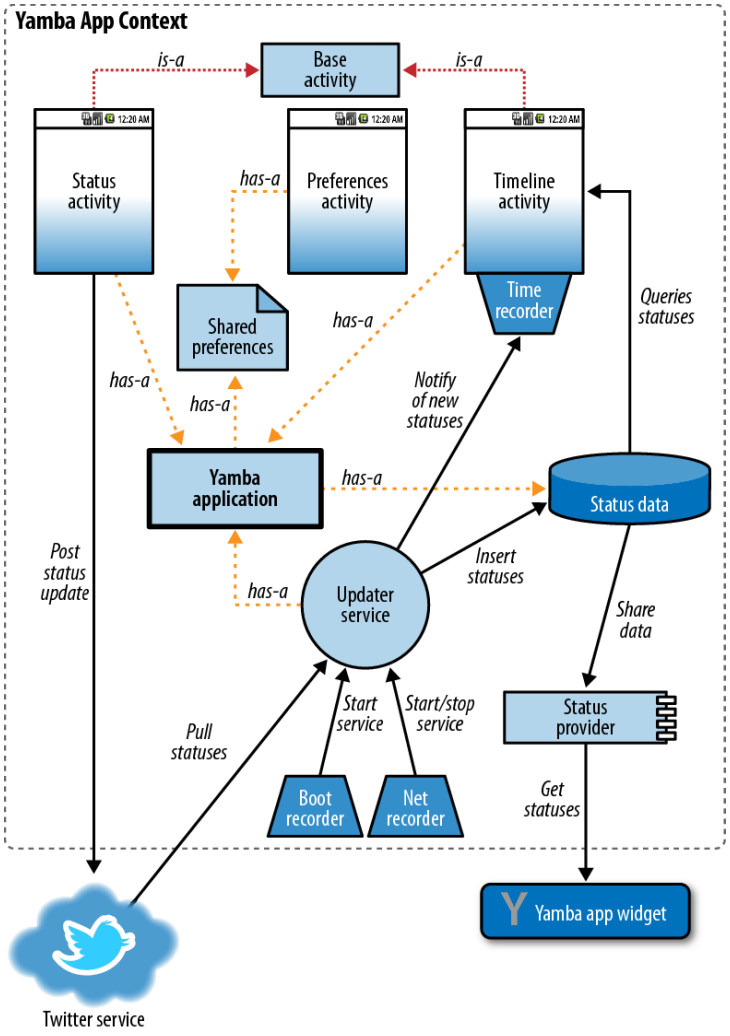


Ilustración 9: Arquitectura de la aplicación Yamba

Podemos resaltar algunas decisiones de diseño tomadas para el desarrollo de esta aplicación como el uso de un componente Service para realizar las invocaciones al servicio de Twitter, ya que las peticiones que utilizan la red no deberían realizarse en el hilo principal de la aplicación ni estar amarradas a un Activity. Este service se encarga de actualizar los datos locales con lo obtenido a través de la red y envía una notificación a los Activity cuando obtiene una respuesta. También se hace uso de la clase YambaApplication, la cual extiende de Application que cada aplicación Android tiene y sirve para almacenar variables

globales durante todo el ciclo de vida de la aplicación. Asimismo se cuenta con un ContentProvider para utilizar datos de la aplicación desde un Widget o desde otras aplicaciones.

1.1.5. Evaluación de Arquitectura de Software

La evaluación de una arquitectura de software puede ser realizada en una o más fases del proceso de desarrollo. Se pueden utilizar para comparar e identificar fortalezas y debilidades en diferentes alternativas arquitectónicas durante las fases tempranas de diseño. Pueden además ser usadas para evaluar sistemas existentes antes de su mantenimiento futuro o mejoramiento del sistema, así como para identificar problemas. Los métodos de evaluación de arquitectura de software pueden ser divididos en 4 categorías principales (Martensson, 2006). Los métodos en las categorías pueden ser usados independientemente pero también pueden ser combinados para evaluar diferentes aspectos de la arquitectura si se requiere.

1.1.5.1. Categorías de Evaluación

1.1.5.1.1. Evaluaciones basadas en experiencia

Las evaluaciones se basan en experiencias previas y conocimiento del dominio de desarrolladores y consultores. Las personas con experiencia pueden decir si la arquitectura de software será lo suficientemente buena.

1.1.5.1.2. Evaluaciones basadas en simulación

Se basan en una implementación a alto nivel de alguno o todos los componentes en la arquitectura de software. La simulación puede luego ser usada para evaluar la calidad de los requerimientos como performance y validez de la arquitectura. La simulación puede también ser combinada con el prototipado, de manera que los

prototipos puedan ser ejecutados en el contexto esperado del sistema completo.

1.1.5.1.3. Modelamiento matemático

Utiliza pruebas y métodos matemáticos para evaluar principalmente requerimientos de calidad tales como performance y comportamiento de los componentes en la arquitectura. El modelamiento matemático es similar a la simulación y puede combinarse con esta para estimar de forma más precisa el performance de los componentes del sistema.

1.1.5.1.4. Evaluaciones basadas en escenarios

Este método intenta evaluar un atributo de calidad particular creando un escenario que fuerce una descripción concreta del requerimiento de calidad. Los escenarios son luego utilizados con la arquitectura de software y las consecuencias son documentadas.

1.1.5.2. ¿Por qué evaluar?

Mientras antes se detecte un problema en un proyecto de software es mejora. El costo de corregir un error encontrado durante la fase de requerimientos o de diseño temprano es órdenes de magnitud menor que corregir ese mismo error durante las pruebas. La arquitectura es el producto de la fase de diseño temprano, y sus efectos en el sistema y el proyecto son profundos.

Una arquitectura inadecuada causará un desastre en el proyecto. Los objetivos de eficiencia no serán logrados, los objetivos de seguridad fallarán. El cliente estará impaciente porque las funcionalidades deseadas no están disponibles y el sistema es muy difícil de modificar. Los cronogramas y los presupuestos explotarán mientras el equipo trata de corregir los problemas. Meses o años después los

cambios que pudieron haber sido anticipados o planeados serán rechazados por ser muy costosos.

La evaluación de la arquitectura es relativamente barata para evitar el desastre. La arquitectura puede ser evaluada cuando aún se trata de una especificación en papel e incluye la ejecución de una serie de experimentos simples. Por hacer una analogía, si se está construyendo una casa no se procedería sin ver cuidadosamente los planos antes de empezar la construcción.

1.1.5.3. ¿Qué cualidades se pueden evaluar?

No se puede simplemente ver una arquitectura y saber si se cumplirán todos los atributos de calidad. Una arquitectura no determina estrictamente todas las cualidades de un sistema. Por ejemplo la usabilidad. La usabilidad es la medida en la que el usuario puede utilizar el sistema efectivamente. La usabilidad es un atributo importante para varios sistemas de información, pero la usabilidad está en gran medida en base a la interfaz de usuario. En el diseño de sistemas modernos, aspectos particulares de la interfaz de usuario tienden a ser encapsulados en pequeñas áreas de la arquitectura. Obtener los datos de la interfaz de usuario y hacerlos fluir por el sistema para que se realize la tarea necesaria es ciertamente un asunto de la arquitectura, como la habilidad de cambiar la interfaz de usuario. Sin embargo, muchos aspectos de la interfaz de usuario no son considerados arquitectónicos por ser decisiones confinadas a limitadas áreas del sistema.

Pero otros atributos de calidad sí caen en el campo de la arquitectura. Por ejemplo, se puede considerar los siguientes atributos:

- Rendimiento: El rendimiento se refiere a la respuesta del sistema - el tiempo requerido para responderá múltiples eventos o al número de eventos procesados en un intervalo de tiempo. Los atributos de calidad suelen ser expresados por el número de transacciones por unidad de tiempo o la cantidad

de tiempo que toma ejecutar una transacción en el sistema. Las medidas de rendimiento suelen ser citadas como benchmarks.

- Robustez: La robustez es la habilidad del sistema de seguir operando a través del tiempo. La robustez es usualmente medida por tiempo hasta falla.
- Disponibilidad: La disponibilidad es la proporción de tiempo que el sistema está funcionando. Se mide por el tiempo entre fallas, como también por qué tan rápido puede volver a operar después de un evento de falla.
- Seguridad: La seguridad es la medida de la habilidad del sistema para resistir a intentos no autorizados de uso y denegación de servicios mientras sigue funcionando correctamente para usuarios autorizados. La seguridad es categorizada en términos de los tipos de amenazas que puede afrontar el sistema.
- Mantenibilidad: La mantenibilidad es la habilidad de hacer cambios a un sistema rápidamente y con bajo costo. Se mide utilizando cambios específicos como benchmarks y grabando qué tan costosos son esos cambios.
- Portabilidad: La portabilidad es la habilidad de un sistema para ejecutarse bajo diferentes entornos computacionales. Estos entornos pueden ser de hardware, software o una combinación de ambos. Un sistema es portable en la medida que todas las asunciones sobre un ambiente particular están confinados a un componente. Si portar a un nuevo sistema requiere cambios la portabilidad es un tipo de mantenimiento.
- Funcionalidad: La funcionalidad es la habilidad de un sistema en hacer la tarea que tiene destinada. Realizar una tarea requiere que muchos o la mayoría de componentes de un sistema trabajen de manera coordinada.

- Variabilidad: La variabilidad es qué tan bien la arquitectura puede ser expandida o modificada para producir nuevas arquitecturas que difieran de una manera específica.
- Integridad conceptual: La integridad conceptual es la visión que unifica el diseño del sistema en todos los niveles. La arquitectura debería hacer cosas similares de manera similar. La integridad conceptual se ejemplifica en una arquitectura que muestra consistencia, tiene un número pequeño de datos y mecanismos de control, usa un número pequeño de patrones para hacer su trabajo.

1.1.5.4. Escenarios

Los atributos de calidad forman la base para la evaluación de una arquitectura, pero simplemente nombrar los atributos no es suficiente para juzgar una arquitectura. Constantemente se escriben requerimientos como "El sistema debe ser robusto", pero sin una elaboración esto está sujeto a interpretación y malas interpretaciones. Lo que uno puede pensar por robusto otro puede considerarlo diferente. El punto es que los atributos de calidad no son cantidades absolutas, ellos existen en el contexto de objetivos específicos. Por ejemplo "El sistema es robusto con respecto a un tipo específico de falla". Es importante que el arquitecto entienda exactamente en qué circunstancias el sistema debe ser robusto (Bass, Recommended Best Industrial Practice for Software Architecture Evaluation, 1997).

En un mundo perfecto los requerimientos de calidad de un sistema estarían completos y no estarían especificados ambiguamente. Sin embargo los documentos de requerimiento suelen no estar escritos o estar escritos pobremente; o pueden no estar presentes al momento de empezar la arquitectura.

Los stakeholders pueden ayudar a escribir los requerimientos para la evaluación. El mecanismo usado es el de escenarios.

El hecho que los atributos de calidad estén relacionados a un contexto ha llevado a la adopción de escenarios como una manera descriptiva de especificar y evaluar los atributos de calidad dentro de un contexto. Los escenarios brindan un modo de caracterizar qué tan bien una arquitectura particular corresponde a las demandas establecidas por esos escenarios. Un escenario es una oración corta describiendo la interacción de un stakeholder con el sistema. Un usuario describiría el uso del sistema para realizar una tarea; estos escenarios son similares a los casos de uso. Un stakeholder de mantenimiento describiría el hecho de hacer un cambio al sistema. Un escenario de desarrollador involucraría el uso de la arquitectura para construir un sistema o predecir su performance. Un escenario de cliente puede describir la reutilización de la arquitectura para diferentes productos.

Cada escenario está asociado con un stakeholder en particular. Cada escenario además consigue un atributo particular de calidad, pero en términos específicos.

No todos los escenarios tratan temas arquitectónicos; por ejemplo, un escenario de portabilidad puede tener implicaciones arquitectónicas pero también puede depender de aspectos más específicos como a nivel de código.

1.1.5.5. Métodos de evaluación de la arquitectura

Los principales métodos para la evaluación a nivel de arquitectura de atributos como eficiencia y mantenibilidad son los siguientes (Martensson, 2006):

1.1.5.5.1. ATAM

ATAM es un método de análisis organizado alrededor de la idea que los estilos arquitectónicos son el principal determinante de los atributos de calidad. Basado en los objetivos de atributos de calidad de usa ATAM para analizar cómo los estilos arquitectónicos ayudan a lograr estos objetivos. Los pasos que tiene son los siguientes:

- Presentar ATAM: El método es descrito a los stakeholders (típicamente representantes del cliente, equipos de arquitectura, representantes de usuarios, mantenedores, administradores, etc.)
- Presentar los objetivos de negocio: El gerente de proyecto describe qué objetivos de negocio motivan el esfuerzo de desarrollo y los objetivos arquitectónicos, por ejemplo contar con alta disponibilidad.
- Presentar la arquitectura: El arquitecto describe la arquitectura propuesta, enfocándose en cómo cumple con los objetivos de negocio.
    - Identificar los enfoques arquitectónicos: Los enfoques arquitectónicos son identificados por el arquitecto, pero no son analizados.
    - Generar el árbol de utilidad de los atributos de calidad: Los factores de calidad que comprometen el funcionamiento del sistema (eficiencia, disponibilidad, seguridad, mantenimiento, etc) son llevados a nivel de escenario y priorizados.
    - Analizar los enfoques arquitectónicos: Basado en los factores de alta prioridad identificados, los enfoques arquitectónicos que consiguen estos factores son analizados. Durante este paso los riesgos, puntos de

sensibilidad y puntos de compensación son identificados.

- Lluvia de ideas y priorización de escenarios: Basado en los escenarios generados por el árbol de utilidad, un conjunto de escenarios son tomados. Este conjunto de escenarios es priorizado.
- Analizar los enfoques arquitectónicos: En este paso se considera los escenarios con alta priorización como casos de prueba para el análisis arquitectónico.
- Presentación de resultados: Basado en la información recolectada el equipo presenta lo que descubrió a la asamblea de stakeholders y potencialmente escribe un reporte.

1.1.5.5.2. SAAM

SAAM es un método para realizar un análisis arquitectónico basado en escenarios. SAAM fue desarrollado originalmente para permitir la comparación de soluciones arquitectónicas. Los pasos de SAAM son:

- Describir la arquitectura candidata: La arquitectura candidata deberá estar descrita con una notación arquitectónica que sea bien entendida por quienes estén envueltos en el análisis. Esta descripción debe indicar los elementos de cómputo y data, así como la relación entre componentes.
- Desarrollo de escenarios: Se desarrollan escenarios que ilustren los tipos de actividades que el sistema debe soportar y los tipos de cambios que se anticipa que se darán en el tiempo. En el desarrollo de escenarios, es importante capturar todos los usos importantes del

sistema. Estos escenarios representan las tareas relevantes para los diferentes roles.

- Desarrollar la evaluación de escenarios: Para cada escenario, determinar si la arquitectura puede ejecutarlos directamente o se requiere un cambio. Por cada escenario indirecto se listan los cambios necesarios en la arquitectura.
- Revelar la interacción de escenarios: Diferentes escenarios pueden necesitar cambios al mismo componente. En tales casos se dice que los escenarios interactúan. Determinar las interacciones es el proceso de identificar escenarios que afectan un conjunto de componentes comunes.
- Evaluación general: Finalmente, se pesa cada escenario en términos de importancia relativa y se obtiene un ranking general. El peso elegido reflejará la importancia relativa de factores de calidad que el escenario manifiesta.

1.1.5.6. Herramientas de Software para la Evaluación de Atributos de Calidad

1.1.5.6.1. Android Testing Framework

El Android testing framework, parte integral del entorno de desarrollo de Android, provee de una arquitectura y poderosas herramientas que permiten probar cualquier aspecto de la aplicación a cualquier nivel.

Consta de las siguientes funciones:

- La suite de pruebas de android se basa en JUnit. Es posible usar JUnit para realizar pruebas de una clase que no usa librerías de android o usar la versión de JUnit para Android para probar los componentes de Android.
- Las extensiones de Android de JUnit proveen clases de prueba específicas para cada componente. Estas clases proveen métodos de ayuda para crear objetos mock y métodos que pueden ayudar a controlar el ciclo de vida de un componente.
- Las suites de prueba están contenidas en paquetes de prueba que son similares a los paquetes de la aplicación, así que no es necesario aprender un nuevo conjunto de herramientas o técnicas para diseñar y construir pruebas.
- Las herramientas del SDK para construir pruebas están disponibles en diversos IDEs. Estas herramientas obtienen información del proyecto bajo prueba y usan esta información para crear automáticamente la estructura y archivos del paquete de pruebas.

1.1.5.6.2. DDMS

Android incluye una herramienta para debugging llamada Dalvik Debug Monitor Server (DDMS), la cual provee servicios a nivel de puertos, captura de pantalla en el dispositivo, información sobre hilos y estado de la memoria en el teléfono, logcat, información de estado, realización de llamadas o envío de SMS, simulación de localización y demás.

Permite analizar la data de performance generada de forma gráfica. DDMS permite analizar un proyecto android ejecutándose en un emulador o que esté conectado a un dispositivo real.

Traceview es una de las herramientas que incluye y es un visor gráfico que muestra los logs creados por la aplicación. Mediante Traceview es posible medir el performance de la aplicación y encontrar errores.

También incluye el Allocation Tracker, que permite visualizar las locaciones de memoria en tiempo real, para poder evitar pérdidas de memoria.

1.1.5.6.3. SonarQube

SonarQube (conocido anteriormente como "Sonar") es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

Entre sus funciones están las siguientes:

- Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software.
- Aunque pensado para Java, acepta otros lenguajes mediante extensiones.
- Se integra con Maven, Ant y herramientas de integración continua como Atlassian Bamboo, Jenkins y Hudson).
- SonarQube cuenta con un plugin que permite realizar el análisis de proyectos en Android, a través de Android Lint

1.1.5.6.4. Selendroid

Selendroid es una herramienta para automatización de pruebas, el cual puede ser usado para aplicaciones nativas en Android así como aplicaciones híbridas. Los casos de prueba son escritos usando el API de Selenium 2. Selendroid puede ser usado en emuladores o en dispositivos reales y puede ser integrado como un nodo dentro del Selenium Grid para pruebas de escala y paralelismo.

Entre sus características están:

- No se requiere modificar la aplicación bajo prueba para automatizarla.
- Pruebas de web móvil utilizando la aplicación nativa de webview.

- El concepto es el mismo para automatizar aplicaciones nativas o híbridas.
- Soporte para gestos.
- Selendroid puede interactuar con múltiples dispositivos (emuladores y dispositivos reales) al mismo tiempo.
- Selendroid soporta hot plugging.
- Incluye un Inspector para simplificar el desarrollo de casos de prueba.

1.2. Metodología

1.2.1. UML

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Ilustración 10: Logo de UML

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa: Booch, OMT y OOSE. UML ha puesto fin a las llamadas “guerras de métodos” que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

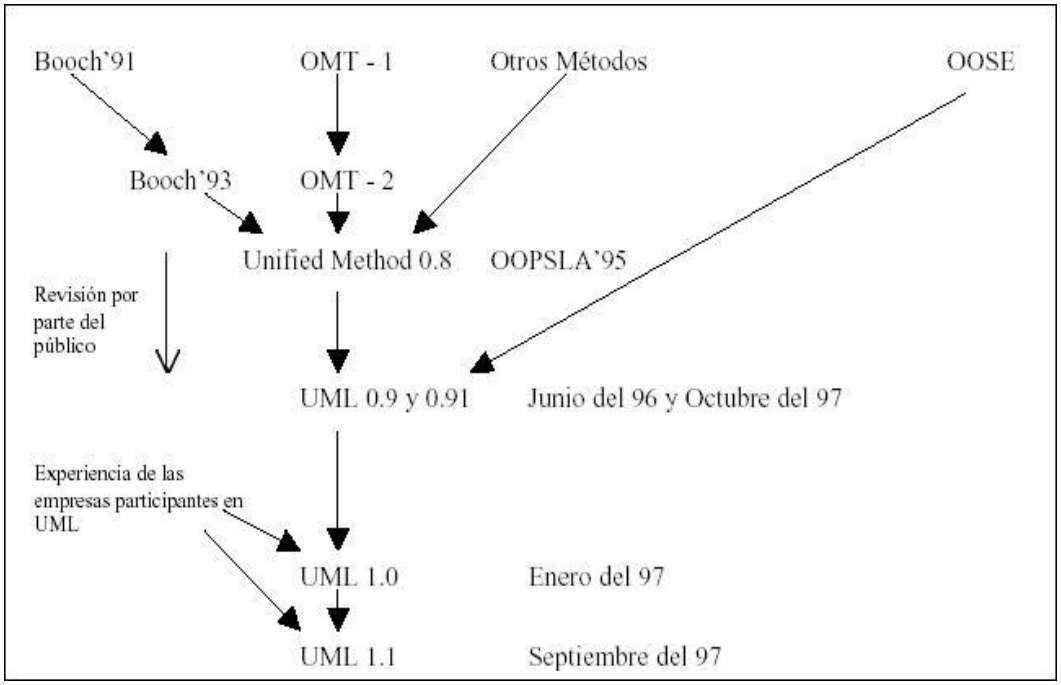


Ilustración 11: Historia de UML

El objetivo principal cuando se empezó a gestar UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. En la figura 2 se puede ver cuál ha sido la evolución de UML hasta la creación de UML 1.1.

Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación. En este curso se sigue el proceso propuesto por Craig Larman[Larman99] que se ajusta a un ciclo de vida evolutivo e incremental dirigido por casos de uso.

En la parte II de este texto se expone la notación y semántica de UML, mientras que en la parte III se presenta el proceso de desarrollo orientado a objetos de Larman, que se sirve de los modelos de UML que se han visto anteriormente.

1.2.1.1. Notación UML

En esta parte se verá cómo se representan gráficamente en UML los conceptos principales de la orientación a objetos.

1.2.1.1.1. Modelos

Un modelo representa a un sistema software desde una perspectiva específica. Al igual que la planta y el alzado de una figura en dibujo técnico nos muestran la misma figura vista desde distintos ángulos, cada modelo nos permite fijarnos en un aspecto distinto del sistema.

Los modelos de UML con los que vamos a trabajar son los siguientes:

- Diagrama de Estructura Estática.
- Diagrama de Casos de Uso.
- Diagrama de Secuencia.
- Diagrama de Colaboración.
- Diagrama de Estados.
- Diagrama de Paquetes

1.2.1.1.1.1. Diagramas de Estructura Estática

Con el nombre de Diagramas de Estructura Estática se engloba tanto al Modelo Conceptual de la fase de Diseño de Alto Nivel como al Diagrama de Clases de Diseño. Ambos son distintos conceptualmente, mientras el primero modela elementos del dominio el segundo presenta los elementos de la solución software. Sin embargo, ambos comparten la misma notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos (asociaciones).

Clases

Una clase se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática (plegada), con los detalles como atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase. En la figura 5 se ve cómo una misma clase puede representarse a distinto nivel de detalle según interese, y según la fase en la que se esté.

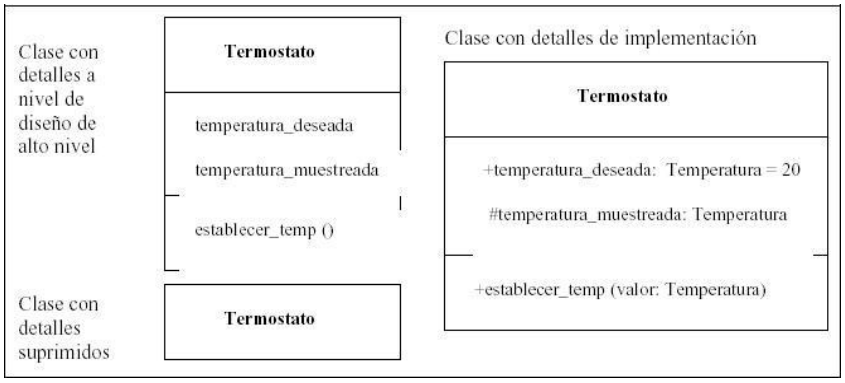


Ilustración 12: Notación para clases a distintos niveles de detalle

Objetos

Un objeto se representa de la misma forma que una clase. En el compartimento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis:

nombre_del_objeto:
nombre_de_la_clase

Puede representarse un objeto sin un nombre específico, entonces sólo aparece el nombre de la clase.

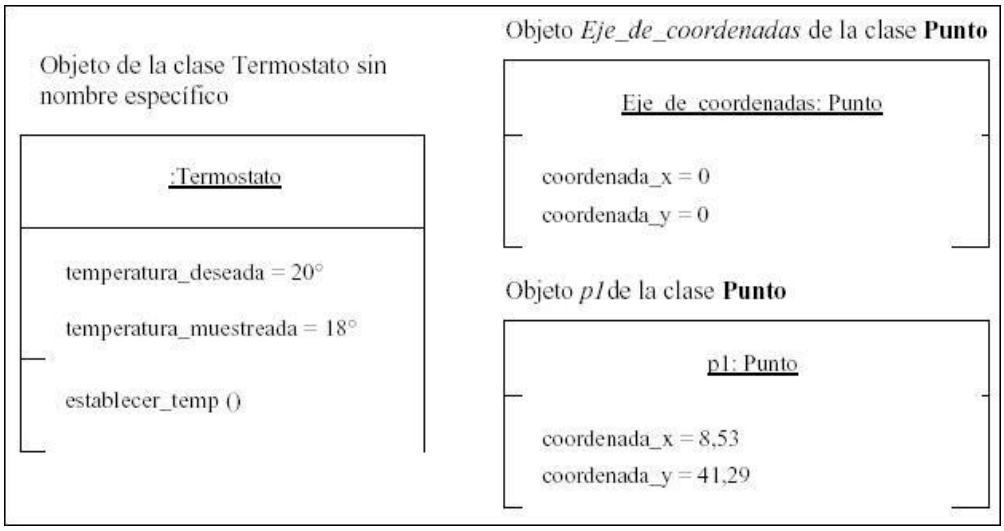


Ilustración 13: Ejemplos de objetos

Asociaciones

Las asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. A continuación se verán los más importantes de entre dichos elementos gráficos.

Nombre de la Asociación y Dirección

El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la asociación. En el ejemplo de la figura 7 se puede leer la

asociación como “Un objeto de la clase Perro es mascota de un objeto de la clase Persona”.

Ilustración 14: Ejemplo de asociación con nombre y dirección

Los nombres de las asociaciones normalmente se incluyen en los modelos para aumentar la legibilidad. Sin embargo, en ocasiones pueden hacer demasiado abundante la información que se presenta, con el consiguiente riesgo de saturación. En ese caso se puede suprimir el nombre de las asociaciones consideradas como suficientemente conocidas.

En las asociaciones de tipo agregación y de herencia no se suele poner el nombre.

Multiplicidad

La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

- Con un número fijo: *1*.
- Con un intervalo de valores: *2..5*.
- Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, *2..** significa 2 o más.
- Con una combinación de elementos como los anteriores separados por comas: *1, 3..5, 7,15..**.
- Con un asterisco: * . En este caso indica que puede tomar cualquier valor (cero o más).

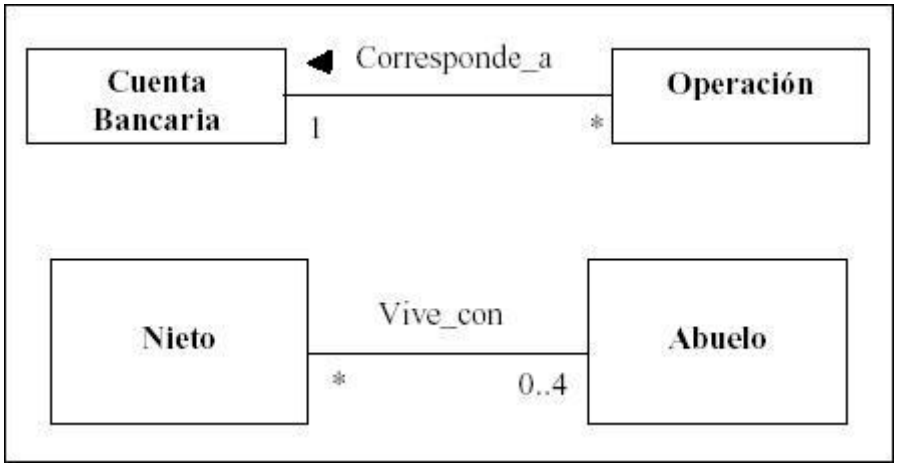


Ilustración 15: Ejemplos de multiplicidad de asociaciones

Roles

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. Se representa en el extremo de la asociación junto a la

clase que desempeña dicho rol.

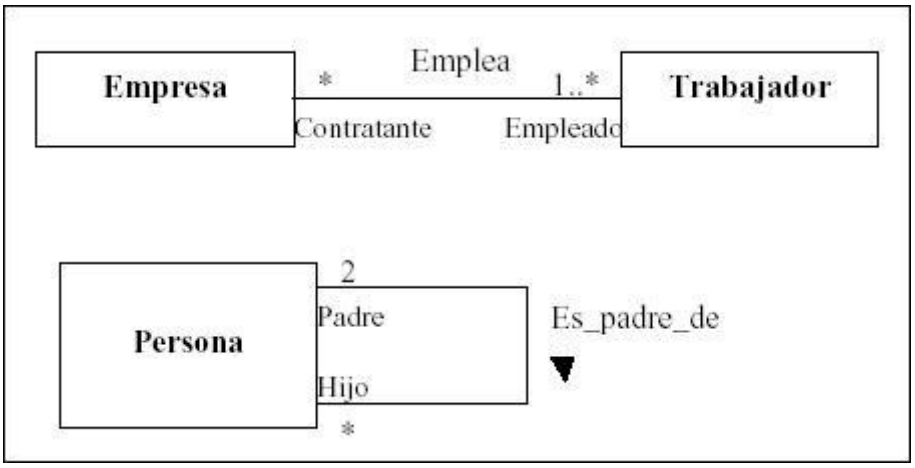


Ilustración 16: Ejemplo de roles en una asociación

Agregación

El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo".

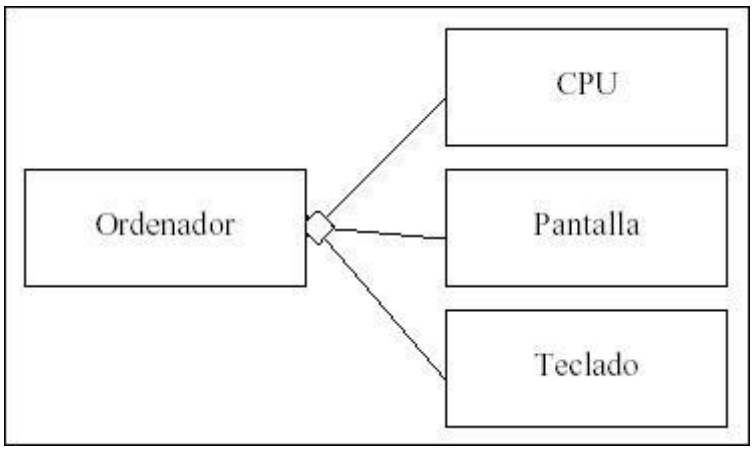


Ilustración 17: Ejemplo de agregación

Clases Asociación

Cuando una asociación tiene propiedades propias se representa como una clase unida a la línea de la asociación por medio de una línea a trazos. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea (como ocurre en el ejemplo de la figura 12). Por el contrario, cuando la clase asociación tiene alguna operación o asociación propia, entonces se pone el nombre en la clase asociación y se puede quitar de la línea.

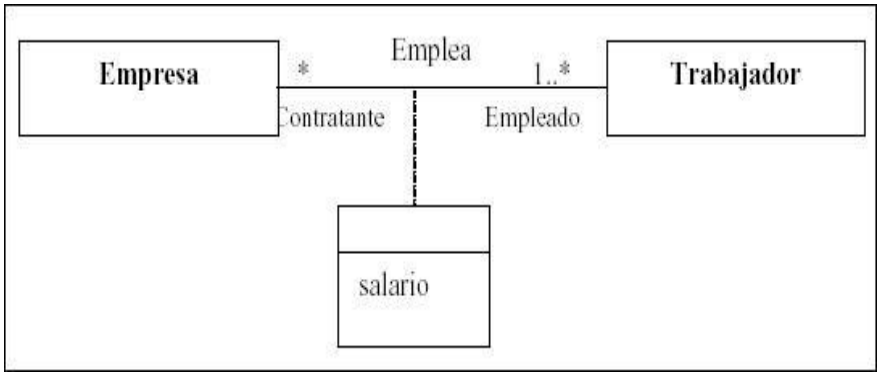


Ilustración 18: Ejemplo de clase asociación

Asociaciones N-Arias

En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central. Si se muestra multiplicidad en un rol, representa el número potencial de tuplas de instancias en la asociación cuando el resto de los N- 1 valores están fijos. En la figura 11 se ha impuesto la

restricción de que un jugador no puede jugar en dos equipos distintos a lo largo de una temporada, porque la multiplicidad de “Equipo” es 1 en la asociación ternaria.

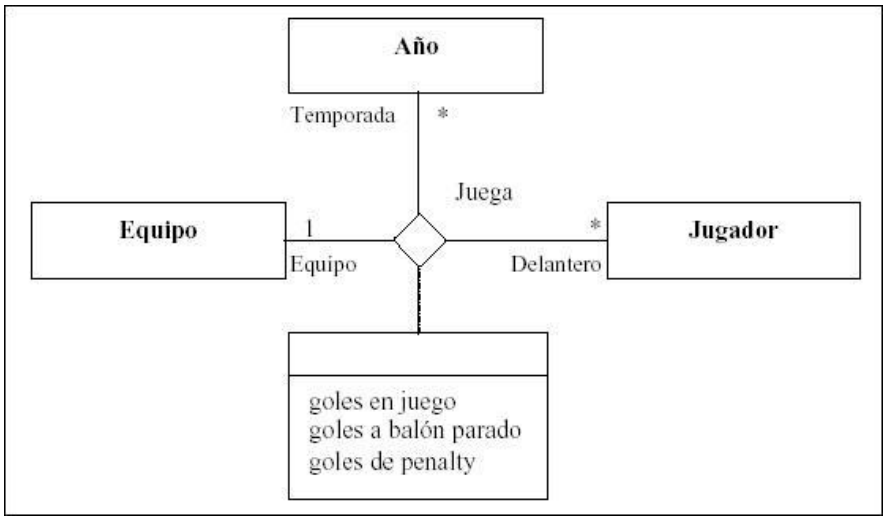


Ilustración 19: Ejemplo de asociación ternaria

Navegabilidad

En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

Herencia

La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase “padre”.

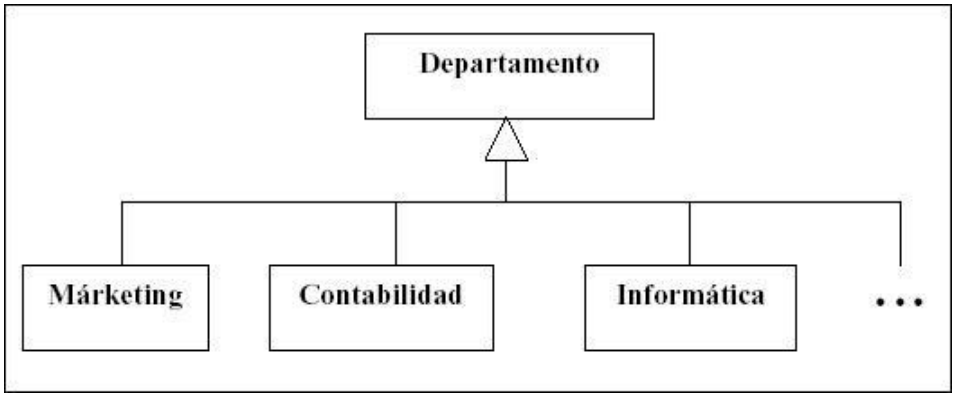


Ilustración 20: Ejemplo de herencia

Si se tiene una relación de herencia con varias clases subordinadas, pero en un diagrama concreto no se quieren poner todas, esto se representa mediante puntos suspensivos. En el ejemplo de la figura 13, sólo aparecen en el diagrama 3 tipos de departamentos, pero con los puntos suspensivos se indica que en el modelo completo (el formado por todos los diagramas) la clase "Departamento" tiene subclases adicionales, como podrían ser "Recursos Humanos" y "Producción".

Elementos Derivados

Un elemento derivado es aquel cuyo valor se puede calcular a partir de otros elementos presentes en el modelo, pero que se incluye en el modelo por motivos de claridad o como decisión de diseño. Se representa con una barra "/" precediendo al nombre del elemento derivado.

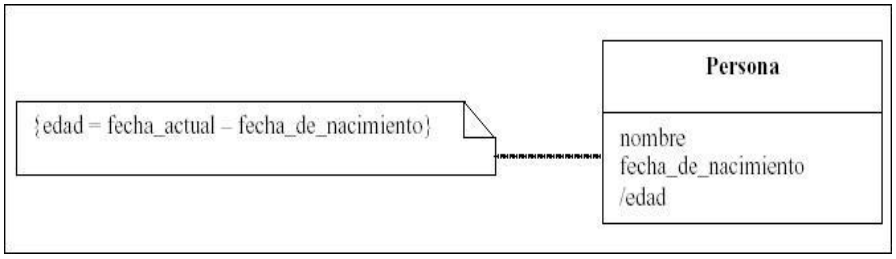


Ilustración 21: Ejemplo de atributo derivado

1.2.1.1.1.2. Diagrama de Casos de Uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

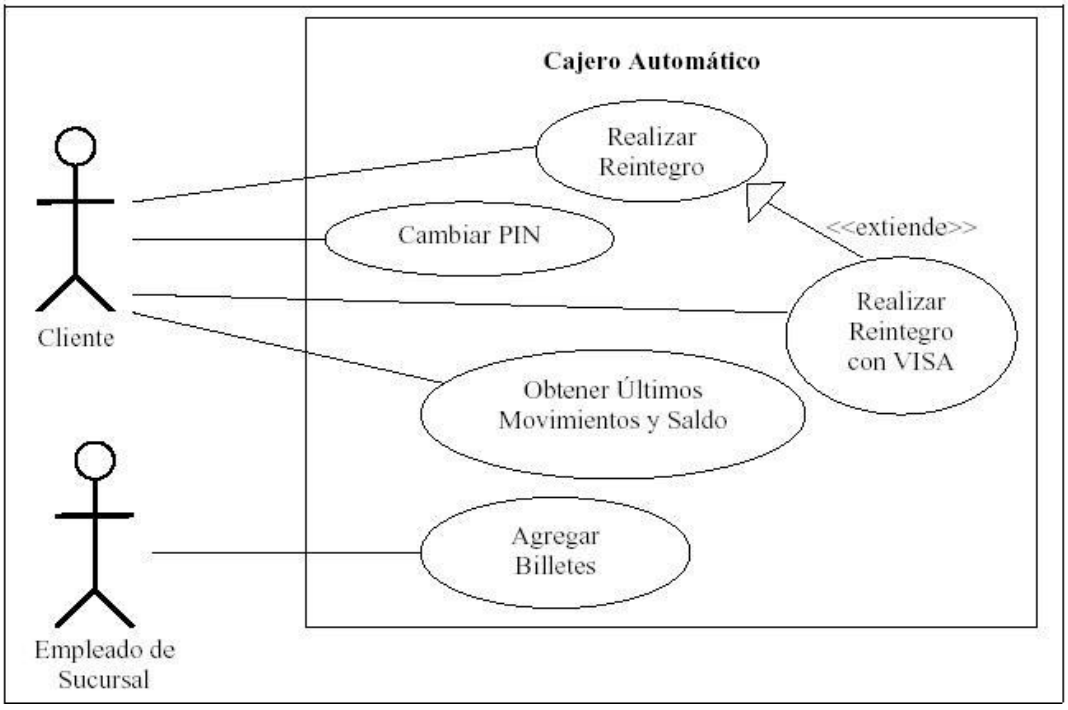


Ilustración 22: Diagrama de Casos de Uso

Elementos

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: Actores, casos de uso y relaciones entre casos de uso.

Actores

Un actor es una entidad externa al sistema que realiza algún tipo de interacción con el mismo. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores (otros sistemas, sensores, etc.).

Casos de Uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

Relaciones entre Casos de Uso

Entre dos casos de uso puede haber las siguientes relaciones:

- **Extiende**: Cuando un caso de uso especializa a otro extendiendo su funcionalidad.
- **Usa**: Cuando un caso de uso utiliza a otro.

Se representan como una línea que une a los dos casos de uso relacionados, con una flecha en forma de triángulo y con una etiqueta <<extiende>> o <<usa>> según sea el tipo de relación. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea. En la figura 15 se muestra un ejemplo de Diagrama de Casos de Uso para un cajero automático.

1.2.1.1.1.3. Diagramas de Interacción

En los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.

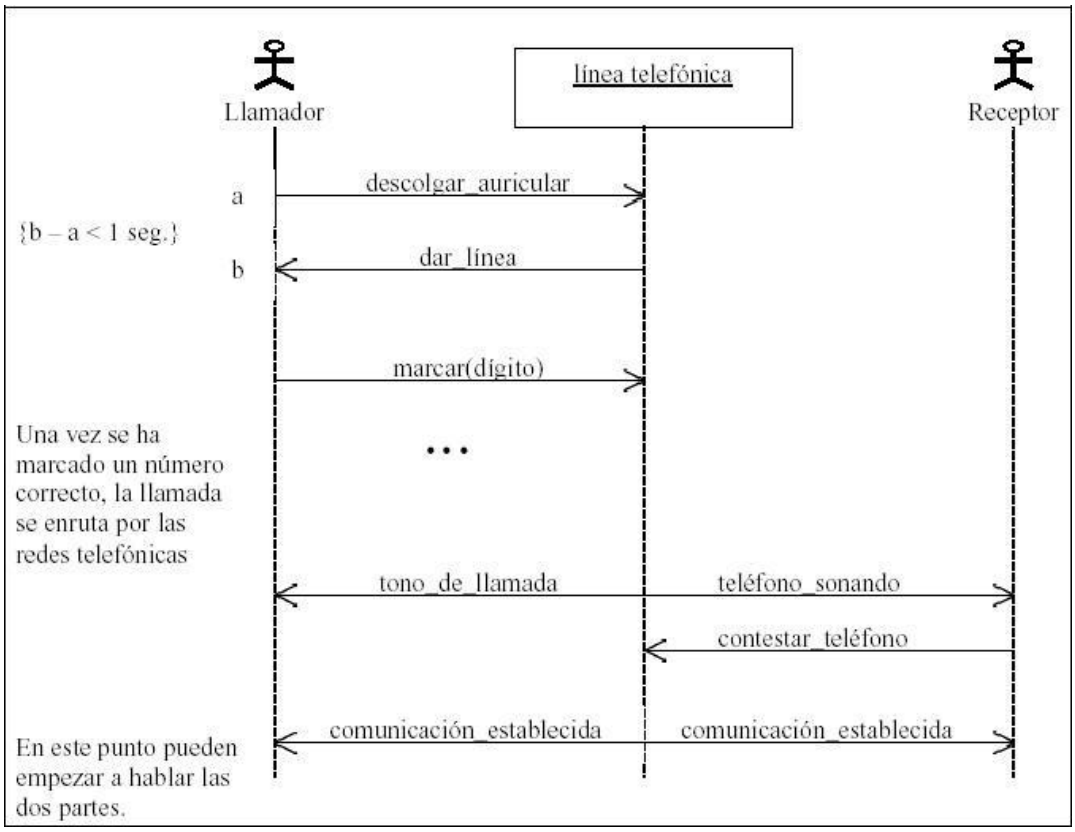


Ilustración 23: Diagrama de Secuencia

1.2.1.1.1.4. Diagrama de Secuencia

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo.

Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren. En la figura 16 se representa el Diagrama de Secuencia para la realización de una llamada telefónica.

1.2.1.1.1.5. Diagrama de Colaboración

Un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

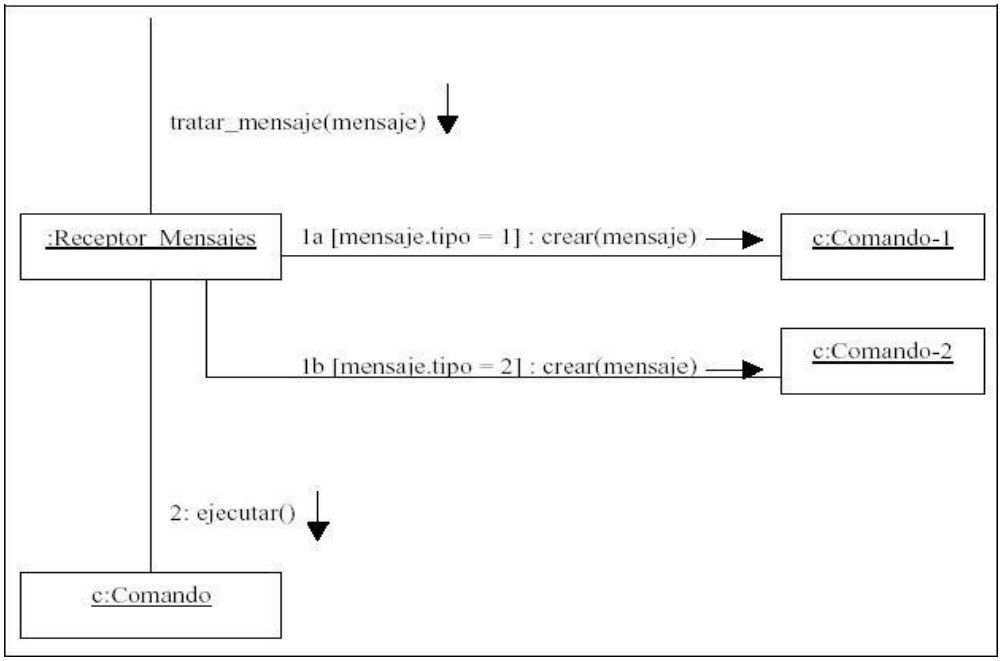


Ilustración 24: Diagrama de Colaboración

En cuanto a la representación, un Diagrama de Colaboración muestra a una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que "circulan", y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis.

Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia. Se pueden indicar alternativas con condiciones entre corchetes (por ejemplo *3 [condición_de_test] : nombre_de_método()*), tal y como aparece en el ejemplo de la figura 17. También se puede mostrar el anidamiento de mensajes con números de secuencia como *2.1*, que significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los *2. x* .

1.2.1.1.1.6. Diagrama de Estados

Un Diagrama de Estados muestra la secuencia de estados por los que pasa un caso de uso o un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos. Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimento aparece el nombre del estado. El segundo compartimento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas.

Una acción de entrada aparece en la forma *entrada/acción_asociada* donde *acción_asociada* es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición

la acción de entrada se ejecuta.

Una acción de salida aparece en la forma *salida/acción_ asociada*. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta.

Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma *nombre_de_evento/acción_asociada.*

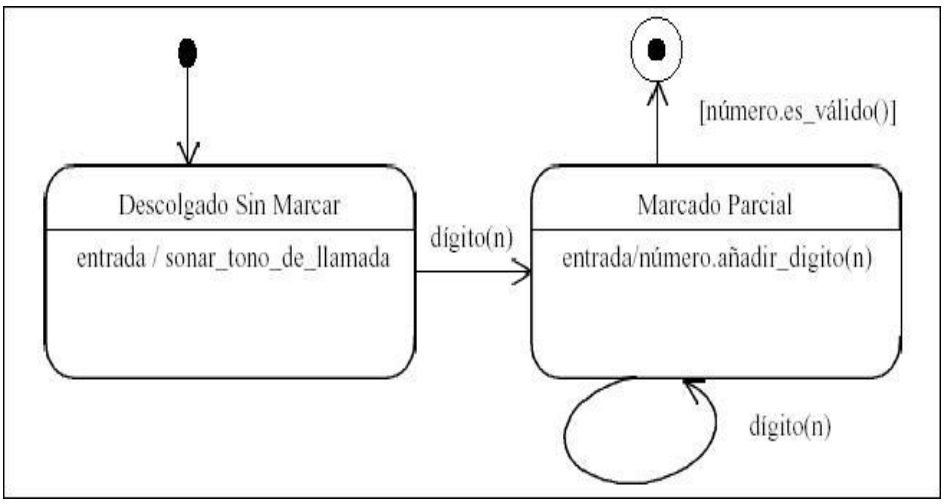


Ilustración 25: Diagrama de Estados

Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (del caso de uso o del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudoestados, pues un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final(es).

1.2.2. Desarrollo Orientado a Objetos

1.2.2.1. Proceso de Desarrollo

Cuando se va a construir un sistema software es necesario conocer un lenguaje de programación, pero con eso no basta. Si se quiere que el sistema sea robusto y mantenible es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto. Tal proceso de desarrollo se ocupa de plantear cómo se realizan las distintas actividades, y cómo se relacionan los productos de las mismas. Con el uso de un proceso de desarrollo adecuado la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente, especialmente cuando se trata de un equipo de desarrollo formado por varias personas.

Craig Larman [Larman99] propone un proceso de desarrollo orientado a objetos. Este proceso no fija una metodología estricta, sino que define una serie de actividades que pueden realizarse en cada fase, las cuales deben adaptarse según las condiciones del proyecto que se esté llevando a cabo. Se ha escogido seguir este proceso debido a que aplica los últimos avances en Ingeniería del Software, y a que adopta un enfoque eminentemente práctico, aportando soluciones a las principales dudas y/o problemas con los que se enfrenta el desarrollador. Su mayor aportación consiste en atar los cabos sueltos que anteriores métodos dejan.

La notación que se usa para los distintos modelos, tal y como se ha dicho anteriormente, es la proporcionada por UML, que se ha convertido en el estándar de facto en cuanto a notación orientada a objetos. El uso de UML permite integrar con mayor facilidad en el equipo de desarrollo a nuevos miembros y compartir con otros equipos la documentación, pues es de esperar que cualquier

desarrollador versado en orientación a objetos conozca y use UML (o se esté planteando su uso).

Se va a abarcar todo el ciclo de vida, empezando por los requisitos y acabando en el sistema funcionando, proporcionando así una visión completa y coherente de la producción de sistemas software. El enfoque que toma es el de un ciclo de vida evolutivo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido por casos de uso, es decir, por la funcionalidad que ofrece el sistema a los futuros usuarios del mismo. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

1.2.2.2. Visión General

El proceso a seguir para realizar desarrollo orientado a objetos es complejo, debido a la complejidad que nos vamos a encontrar al intentar desarrollar cualquier sistema software de tamaño medio-alto. El proceso está formado por una serie de actividades y subactividades, cuya realización se va repitiendo en el tiempo aplicadas a distintos elementos. En este apartado se va a presentar una visión general para poder tener una idea del proceso a alto nivel, y más adelante se verán los pasos que componen cada fase. Las tres fases al nivel más alto son las siguientes:

Planificación y Especificación de Requisitos: Planificación, definición de requisitos, construcción de prototipos, etc.

Construcción: La construcción del sistema. Las fases dentro de esta etapa son las siguientes:

Diseño de Alto Nivel: Se aborda el problema viendo al sistema a construir como una caja negra, centrándonos en la visión que desde el exterior tienen los actores, esto es, en los casos de uso. Se analiza el problema construyendo el modelo concepcual.

Diseño de Bajo Nivel: El sistema definido en la fase anterior se especifica en detalle, describiendo todas las operaciones que el sistema va a tener que realizar internamente para satisfacer lo especificado en el diseño de alto nivel.

Implementación: Se lleva lo especificado en el diseño a un lenguaje de programación.

Pruebas: Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado en la etapa de Planificación y Especificación de Requisitos.

Instalación: La puesta en marcha del sistema en el entorno previsto de uso.

De ellas, la fase de Construir es la que va a consumir la mayor parte del esfuerzo y del tiempo en un proyecto de desarrollo. Para llevarla a cabo se va adoptar un enfoque evolutivo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del diseño de alto y bajo nivel hasta la implementación y pruebas, tal y como se muestra en la figura 19. El sistema va creciendo incrementalmente en cada ciclo.

Con esta aproximación se consigue disminuir el grado de complejidad que se trata en cada ciclo, y se tiene pronto en el proceso una parte del sistema funcionando que se puede

contrastar con el usuario/cliente.

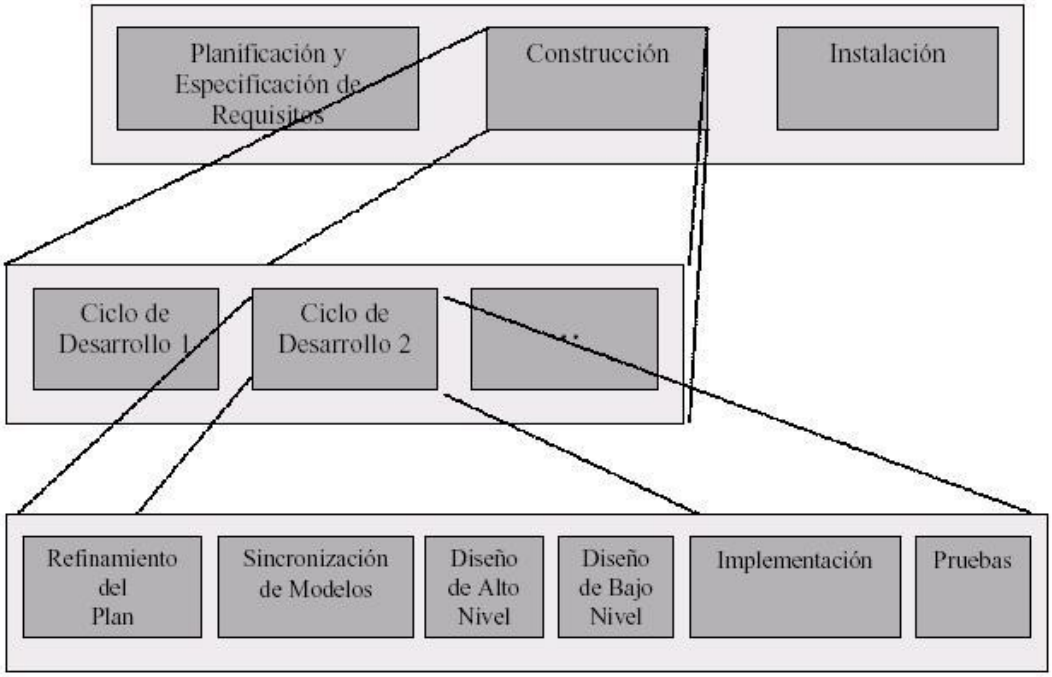


Ilustración 26: Desarrollo Evolutivo en la Construcción

1.3. Tecnología

1.3.1. Android

En pocas palabras, Android es un sistema operativo dirigido a equipos móviles, tales como teléfonos y otros dispositivos informáticos limitados, tales como netbooks y tabletas computadoras.

El concepto y la plataforma fue la creación de Android Inc., una compañía de lanzamiento pequeña de Palo Alto, California, que fue adquirida por Google en 2005. Su objetivo declarado era crear un sistema pequeño, estable, flexible y actualizarse fácilmente operativo para teléfonos móviles que sería muy atractivo para los fabricantes de dispositivos y operadores de telefonía (Jordan & Greyling, 2011). Android es la primera plataforma completa, abierta y gratuita para móviles (Conder & Lauren, 2011):

- Completa: Los diseñadores tomaron un enfoque integral cuando se desarrolló la plataforma Android. Se comenzó con un sistema operativo seguro y se construyó un framework de software robusto en la parte superior que permite oportunidades de desarrollo de aplicaciones ricas.
- Abierta: La plataforma Android es proporcionada a través de licencias de código abierto. Los desarrolladores tienen acceso sin precedentes a las características del teléfono al desarrollar aplicaciones.
- Libre: Las aplicaciones de Android son libres de desarrollar. No hay licencias o regalías por pagar. No hay cuotas de afiliación. Sin costo para las pruebas. Las aplicaciones Android pueden ser distribuidas y comercializadas en gran variedad de formas.

1.3.1.1. Kernel Linux 2.6

Android fue creado con el kernel 2.6 de Linux, el cual es open source (Felker, 2011). El equipo de Android eligió este kernel porque proveía las funcionalidades principales para el desarrollo del sistema operativo Android. Las principales funcionalidades que incluye el kernel 2.6 de Linux son:

- Modelo de Seguridad: El kernel de Linux maneja la seguridad entre la aplicación y el sistema.
- Gestión de Memoria: El kernel gestiona la memoria por el programador, dejándolo a uno libre de desarrollar su aplicación.
- Gestión de procesos. El kernel de Linux gestiona también los procesos, asignando recursos según los requieran los procesos.
- Pila de Red: El kernel de Linux gestiona la comunicación en la red.

- Modelo de Manejador: La meta de Linux es asegurar que todo funcione. Los proveedores de hardware pueden construir sus drivers para Linux.

1.3.1.2. Arquitectura

Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, middleware y aplicaciones básicas para el usuario (Conder & Lauren, 2011). Su diseño cuenta, entre otras, con las siguientes características:

- Busca el desarrollo rápido de aplicaciones, que sean reutilizables y verdaderamente portables entre diferentes dispositivos.
- Los componentes básicos de las aplicaciones se pueden sustituir fácilmente por otros.
- Cuenta con su propia máquina virtual, Dalvik, que interpreta y ejecuta código escrito en Java.
- Permite la representación de gráficos 2D y 3D.
- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM.
- Controla los diferentes elementos hardware: Bluetooth, Wi-Fi, cámara fotográfica o de vídeo, GPS, acelerómetro, infrarrojos, etc., siempre y cuando el dispositivo móvil lo contemple.
- Cuenta con un entorno de desarrollo muy cuidado mediante un SDK disponible de forma gratuita.
- Ofrece un plug-in para uno de los entornos de desarrollo más populares, Eclipse, y un emulador integrado para ejecutar las aplicaciones.

Cada una de las capas de la arquitectura de Android utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores.

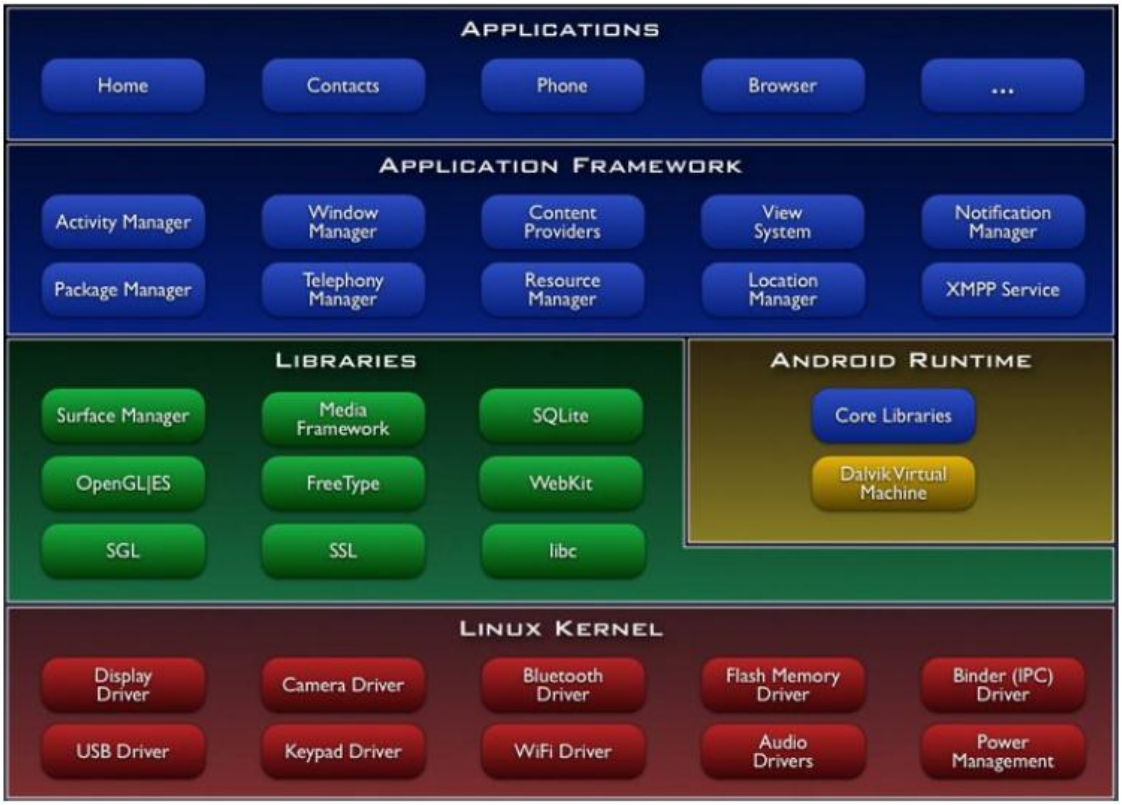


Ilustración 27: Arquitectura de Android

La capa más inmediata es la corresponde al núcleo de Android. Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluya un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

La elección de Linux 2.6 se ha debido principalmente a dos razones: la primera, su naturaleza de código abierto y libre se ajusta al tipo de distribución que se buscaba para Android (cualquier otra opción comercial disponible hoy día hubiera comprometido la licencia de Apache); la segunda es que este kernel de Linux incluye de por sí numerosos drivers, además de contemplar la gestión de memoria, gestión de procesos, módulos de seguridad, comunicación en red y otras muchas responsabilidades propias de un sistemas operativo (Burnette, 2010).

La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases de Java, y la máquina virtual Dalvik.

Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El framework de aplicaciones representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework, representado por este nivel.

El último nivel del diseño arquitectónico de Android son las aplicaciones. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas

aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

1.3.1.3. Framework de Android

El framework de Android fue desarrollado con varias funcionalidades (Felker, 2011). Estas funcionalidades fueron obtenidas de numerosos proyectos open source. La salida de estos proyectos resultó en lo siguiente:

- Run time de Android: El run time de Android está compuesto por librerías básicas de Java y la máquina virtual Dalvik.
- Open GL (librerías gráficas): Este API es utilizado para producir gráficos de computadora en 2D y 3D.
- WebKit: Este motor de navegador web open source provee las funcionalidades para mostrar contenido web y simplificar la carga de páginas.
- SQLite: Este motor de base de datos relacional open source está diseñado para estar embebido en los dispositivos.
- Media frameworkds: Estas librerías permiten reproducir y grabar audio y video.
- Secure Sockets Layer (SSL): Estas librerías son responsables de la seguridad en Internet.

1.3.1.4. Tipos de componentes Android

Las aplicaciones en Android tienen cuatro componentes básicos (Ableson, Sen, & King, 2011):

Activity

Una aplicación puede tener una interfaz de usuario, pero no tiene que tener uno. Si tiene una interfaz de usuario, tendrá al menos una actividad.

La manera más fácil pensar en una actividad Android es que se relacionan con una pantalla visible, debido a que más a menudo que no hay una relación uno a uno entre una actividad y una interfaz de usuario de pantalla. Esta relación es similar a la de un controlador en el paradigma MVC.

La mayoría de las aplicaciones permiten la ejecución de varias acciones a través de la existencia de una o más pantallas. Por ejemplo, piénsese en una aplicación de mensajes de texto. En ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra tercera puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas deberías estar representada a través de un componente Activity, de forma que navegar de una ventana a otra implica lanzar una actividad o dormir otra. Android permite controlar por completo el ciclo de vida de los componentes Activity.

Service

Si una aplicación tendrá un largo ciclo de vida, a menudo es mejor ponerlo en un servicio. Para ejemplo, una utilidad de sincronización de datos de fondo debe ser aplicado en un De servicio. Una mejor práctica es poner en marcha los servicios en forma periódica o cuando sea necesario-, provocó por un sistema de alarma, y luego tener el servicio terminará cuando su tarea se ha completado. Al igual que la actividad, un servicio es una clase en tiempo de ejecución de Android que se debe se extienden, como se muestra en el siguiente listado. En este ejemplo se extiende a un servicio, y periódicamente publica un mensaje informativo en el registro de Android.

Otro ejemplo típico de este componente es un reproductor de música. La interfaz del reproductor muestra al usuario las distintas canciones disponibles, así como los típicos botones de reproducción, pausa, volumen, etc. En el momento en el que el usuario reproduce una canción, ésta se escucha mientras se siguen visionando todas las acciones anteriores, e incluso puede ejecutar una aplicación distinta sin que la música deje de sonar. La interfaz de usuario del reproductor sería un componente Activity, pero la música en reproducción sería un componente Service, porque se ejecuta en background. Este elemento está implementado por la clase de mismo nombre Service.

BroadcastReceiver

Si una aplicación quiere recibir y responder a un evento global, como un zumbido teléfono o un mensaje de texto, que debe registrarse como BroadcastReceiver.

Como los servicios, BroadcastReceivers no tiene una interfaz de usuario. Aún más importante, el código se ejecuta en el método OnReceive de un BroadcastReceiver debe hacer ninguna suposición sobre las operaciones de persistencia o de larga duración.

Content Provider

Si una aplicación gestiona los datos y necesita exponer los datos a otras aplicaciones ejecuta en el entorno de Android, usted debe considerar un ContentProvider. Si un componente de aplicación (actividad, servicio o BroadcastReceiver) necesita tener acceso a datos de otra aplicación, el componente de acceso a la otra aplicación ContentProvider. El ContentProvider implementa un conjunto estándar de métodos para permitir que una aplicación para acceder a un almacén de

datos. El acceso puede ser para leer o escribir operaciones, o para ambos. A ContentProvider puede proporcionar datos a una actividad o Servicio de la aplicación contienen, así como a una actividad o servicio contenidos en otras aplicaciones.

Una clase que implemente el componente Content Provider contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación. Existe una colección de clases para distintos tipos de gestión de datos en el paquete android.provider. Además, cualquier formato adicional que se quiera implementar deberá pertenecer a este paquete y seguir sus estándares de funcionamiento.

Intent

Un Intent es el elemento básico de comunicación entre los distintos componentes Android. Es una clase que permite especificar una Activity a ejecutar, llamando a uno de los métodos de la clase Activity con ese Intent de parámetro. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones.Un Intent consiste básicamente en la voluntad de realizar alguna acción, generalmente asociada a unos datos. Lanzando un Intent, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar qué aplicación entre las instaladas es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una URL en algún navegador web, o escribir un correo electrónico desde algún cliente de correo.

Un intent está formado por una acción, datos (que se representan mediante URIs), datos extra en pares clave/valor y un nombre de clase explícito, llamado nombre del componente.

1.3.2. Android Studio

Android Studio es un entorno de desarrollo integrado (IDE) para la plataforma Android. Fue anunciado por Ellie Powers el 16 de mayo de 2013. Android Studio está disponible para desarrolladores para probarlo gratuitamente. Basado en IntelliJ IDEA de JetBrains, está diseñado específicamente para desarrollar para Android. Está disponible para descargar para Windows, Mac OS X y Linux.

Entre sus características principales se incluye:

- Renderización en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Gradle.
- Refactorización especifica de Android y arreglos rápidos.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.

CAPITULO II: DESCRIPCIÓN DE LA ARQUITECTURA FORMULADA

Después de hacer una evaluación de la situación actual y realizar la investigación bibliográfica de los temas referidos a la plataforma Android y a las arquitecturas de software se procede a presentar la descripción de la arquitectura formulada.

2.1. Contexto del Proyecto

Un factor crítico a la hora de desarrollar una aplicación móvil en la plataforma Android es el diseño de la estructura y componentes con la que contará la aplicación, lo cual guiará en adelante el desarrollo del software y determinara los niveles de calidad del producto conseguido. La presente investigación busca brindar una arquitectura aplicable a diversos proyectos de aplicaciones en Android, enfocándose en el componente móvil, sin incluir las decisiones de diseño a tomarse para otros componentes, como por ejemplo un backend web.

2.2. Documentación de la Arquitectura

Uno de los desarrollos más importantes dentro de la construcción del software ha sido el desarrollo de la arquitectura de software, que permite representar la estructura de un sistema a un nivel mayor que el dado por la programación o incluso el diseño. El propósito de este capítulo es describir a detalle la arquitectura de software que se ha formulado.

La arquitectura de software está compuesta por un conjunto de decisiones significativas acerca de la organización y estructuración del sistema, los componentes a utilizarse y la manera en la que estos se comunican; decisiones que dirigen el diseño y la evolución de este a través del tiempo. Una arquitectura puede definirse a varios niveles, en el caso de la presente investigación se desarrolla una arquitectura a nivel de aplicación.

Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo y mantenimiento de los sistemas de software es necesario que se cuente con una manera precisa de representarla. De esta manera la arquitectura puede ser entendida y aplicada adecuadamente dentro de un proceso de desarrollo, además de poder ser extendida por nuevas investigaciones.

Una arquitectura de software es algo complejo que no puede ser descrito en una sola dimensión. Un enfoque para la descripción de arquitecturas de software es el uso de un modelo pesado, único y sobrecargado, lo que la hace difícil de entender y de dar mantenimiento, volviéndose rápidamente irrelevante para la tarea de construir un sistema. Una mejor alternativa a ese enfoque es partir la descripción arquitectónica en un número de vistas separadas, cada una de las cuales brinda un aspecto de la arquitectura, lo cual es más efectivo. Las diferentes vistas se encargan de asuntos diferentes de ingeniería, la separación de asuntos ayuda a mostrar mejor las decisiones tomadas. Se elige un conjunto de vistas a ser utilizadas en la representación de la arquitectura según las estructuras que se quieren mostrar de la arquitectura formulada. En cada vista se muestra las decisiones tomadas y las alternativas consideradas.

La descomposición de un sistema en estructuras es esencial para un entendimiento del sistema con muchos niveles. Esto es precisamente lo que se pretende con las vistas, descomponer al sistema en estructuras.

Podemos decir que documentar una arquitectura de software consiste en documentar detalladamente las vistas relevantes a esta. Considerando lo anterior para documentar la arquitectura propuesta se presentarán diferentes vistas que permitan enfocar mejor las estructuras y componentes elegidos desde perspectivas distintas, así como la explicación del comportamiento de la arquitectura y la fundamentación correspondiente.

2.3. Documentación de las Vistas

Existen múltiples vistas que pueden ser utilizadas para representar una arquitectura de software. Diversos autores presentan diferentes vistas para esta tarea, cada una

de las cuales sirve para mostrar una parte o un enfoque de un todo mayor. La decisión de las vistas que se deben utilizar en cada caso depende de lo que se desee mostrar.

No en todos los casos se requiere utilizar todas las vistas existentes. Se elige un conjunto de vistas a ser utilizadas en la representación de la arquitectura según las estructuras que se quieren mostrar y los interesados a quienes va dirigido.

La principal pregunta en este punto es qué vistas son relevantes y deben ser utilizadas. Esto depende de los objetivos de la documentación. En el caso de la presente investigación se trata de una arquitectura enfocada en el diseño de aplicaciones móviles, por lo cual la arquitectura será utilizada por personas encargadas de tal tarea, con conocimientos técnicos sobre desarrollo.

Debido a que la arquitectura está pensada para ser utilizada a nivel del diseño de aplicaciones móviles, en la descripción de las vistas se llega al detalle suficiente para facilitar esa tarea, describiendo componentes propios de la plataforma Android en las vistas que correspondan a un nivel técnico.

Al tratarse de una arquitectura que podrá ser utilizada en distintos proyectos algunos elementos serán presentados a un nivel de abstracción alto debido a que será tarea del diseñador que aplique la arquitectura seleccionar los componentes adecuados para sus tareas específicas. Los componentes generales y base de la arquitectura contarán con una descripción más detallada. De igual forma, se omiten vistas enfocadas en la implementación y despliegue de la aplicación ya que estas decisiones serán tomadas por el diseñador o la persona correspondiente que esté utilizando la arquitectura.

Además de existir distintas vistas para describir la arquitectura existen distintas notaciones para describir las vistas. En la presente investigación se ha elegido la notación UML. UML fue desarrollado originalmente para permitir realizar un diseño detallado, lo cual encaja muy bien con lo que se desea desarrollar, además de ser ampliamente aceptado y utilizado como estándar en el desarrollo de software. UML permite que se represente de manera semi-formal la estructura

general del sistema, con la ventaja de que este mismo lenguaje puede ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios.

Se ha considerado presentar las siguientes vistas:

- **Vista de módulos:** En esta vista se presentan de manera estática los principales componentes de la arquitectura.
- **Vista de ejecución:** En esta vista se presenta cómo se comportan los componentes en tiempo de ejecución.

Para cada vista presentada se utiliza la siguiente organización:

- **Representación Gráfica:** Esta es la presentación inicial de la vista, la que consiste en un diagrama UML en el cual se muestran los principales elementos y la relación existente entre ellos. Es el punto inicial para la descripción de la vista.
- **Catálogo de elementos:** En este catálogo de describe detalladamente todos los elementos que forman parte de la vista, explicando sus propiedades de acuerdo al tipo de vista que corresponda.
- **Comportamiento:** Se presenta el comportamiento de los elementos de la vista, considerando que algunos elementos tienen interacciones complejas con el entorno, mediante escenarios y diagramas adecuados. El objetivo de esto es facilitar el entendimiento y utilización de la arquitectura propuesta.
- **Fundamentación:** Aquí se explica las decisiones arquitectónicas tomadas, concernientes a la vista actual, brindando un argumento convincente. Esto permite la revisión de la arquitectura, mejora el entendimiento y la posible extensión de la arquitectura en un futuro trabajo.

Finalmente se documenta las pautas a considerar para aplicar la arquitectura presentada en el diseño de un proyecto.

2.4. Vistas

Para la documentación de la arquitectura propuesta se han considerado las siguientes vistas arquitectónicas.

2.4.1. Vista de Módulos

Las vistas modulares presentan las principales unidades de implementación, descomponiendo el software en unidades manejables, indicando de qué se encarga cada unidad, los servicios que provee a otras unidades y cómo se ensamblan para lograr algo más grande. Permiten identificar cómo cambios en una parte del sistema pueden afectar otras partes del sistema. Permiten desarrollar un plano para la construcción del software.

Para esta vista se utiliza un tipo de vista modular llamada de uso, donde se muestra la relación de dependencia existente entre los elementos, indicando los módulos que deben existir para que el sistema funcione correctamente. Este estilo permite un desarrollo incremental ya que se van agregando elementos o especializando los elementos existentes según las necesidades del proyecto. Mediante esta vista mostramos la estructura básica que conforma la arquitectura formulada de manera estática.

2.4.1.1. Representación Gráfica

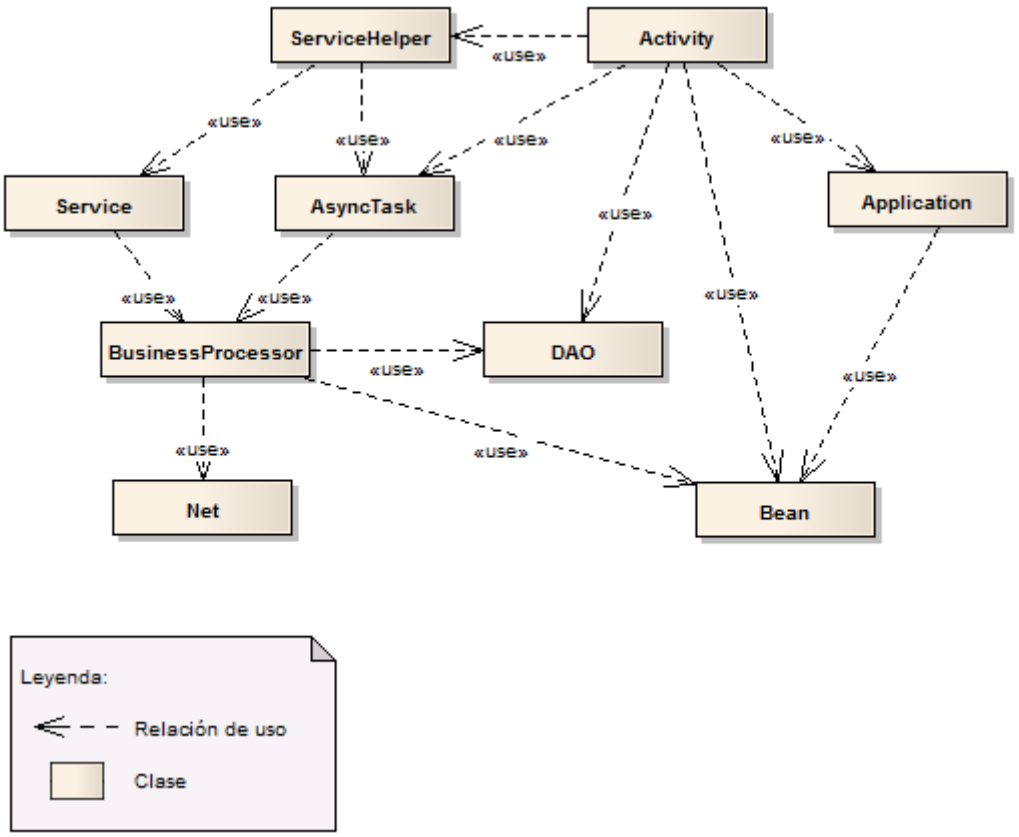


Ilustración 28: Representación Gráfica de la Vista de Módulos

2.4.1.2. Catálogo de Elementos

A continuación se describen los elementos presentes en esta vista. Para cada elemento se presenta su descripción y funcionalidad dentro de la arquitectura, la relación de uso que tiene con otros elementos y datos adicionales referentes a su implementación donde se tocan algunos puntos más técnicos.

2.4.1.2.1. Activity

Descripción

Los Activity representan las vistas dentro de la aplicación móvil. Es a través de éstos que se interactúa con el usuario, mostrándole información y capturando sus datos de entrada.

Un enfoque muy común al programar en Android es llenar los Activity de código referente a la vista y además incluir toda la lógica de negocio, temas de persistencia, comunicación con servidor, de tal forma que un Activity puede ser suficiente para contar con una aplicación completa. El problema con esto es que se están aumentando las responsabilidades de un solo componente, lo cual va contra el principio de alta cohesión, además que no se aprovechan otros componentes brindados por Android, los cuales pueden mejorar la performance y robustez de la aplicación.

Lo que se propone en esta arquitectura es reducir las responsabilidades de las Activity, de forma que se enfoque en la interacción con el usuario y delegue las tareas de lógica de negocio y procesamiento de los datos ingresados a otros componentes. Con esto se busca mejorar la calidad de la aplicación, ya que en muchas ocasiones los componentes a los que se delega son componentes especializados de Android para determinadas tareas y también se facilita el desarrollo ya que se puede realizar una mejor separación de tareas entre programadores, así como se facilita el mantenimiento y la aplicación se hace más clara de entender.

Relación con otros elementos

- **ServiceHelper:** El Activity hace uso del ServiceHelper para delegarle las tareas a realizarse con los datos

obtenidos como entrada por parte del usuario. Asímismo se reciben de manera asíncrona los resultados de estas invocaciones.

- **AsynkTask:** El Activity además puede hacer uso del AsyncTask para delegar tareas, como hace con el ServiceHelper, se recomienda hacer esto para tareas más rápidas.
- **Application:** El Activity hace uso del elemento Application para acceder a las variables globales de la aplicación, por ejemplo los datos del usuario actual.
- **DAO:** Se usan los DAO para recuperar los datos que debe mostrar en la interfaz, por ejempl la lista de productos que está grabada permanentemente en el móvil.
- **Bean:** Se hace uso de los Beans que encapsulan las entidades de negocio de la aplicación.

Implementación

Los Activity son implementados como se hace normalmente, extendiendo de la clase Activity base, implementando los métodos de manejo de ciclo de vida y obteniendo los views definidos en el layout. Los Activity deberán ser registrados en el Manifext.xml.

Además de simples Activity es común el uso de Fragments (elementos que representan una porción de la interfaz de usuario), estos elementos van anexados a los Activities así que de ser utilizados simplemente se delegan las tareas designadas al Activity a sus Fragments correspondientes.

En ocasiones son utilizados frameworks con clases propias que son especializaciones de la clase Activity base, como el caso del uso del popular frameworks ActionBarSherlock (para facilitar la implementación del patrón de diseño ActionBar). El

uso de estos frameworks no interfiere con el correcto funcionamiento de la arquitectura.

Para obtener el resultado de las invocaciones al ServiceHelper se hace uso de los Broadcast, por lo que hay que registrar un BroadcastReceiver. Además el Activity deberá consultar por el resultado de una invocación en caso haya sido pausada, lo que causaría que no escuche el Broadcast.

2.4.1.2.2. ServiceHelper

Descripción

El ServiceHelper funciona como un facade para todos los servicios brindados a los Activity, canalizando todas las invocaciones, viene a ser un API o la interfaz a utilizarse. Sus métodos son asíncronos de manera que no bloquean el flujo de los Activity al invocarlos. La arquitectura propuesta recomienda que toda aplicación cuente con un ServiceHelper. La interfaz que presenta este elemento para llegar a los Service es sencilla, simplificando la forma en la que los Activitys la usan y encargándose de de hacer las invocaciones más complejas.

Este elemento no es el encargado de llevar a cabo estos servicios realmente, sino que a su vez delega esta tarea a otro elemento que es el Service, llevando un registro de las invocaciones hechas para evitar enviar múltiples veces la misma solicitud. A su vez se encarga de enviar el mensaje de vuelta a quien lo invocó. Como es el elemento que controla las invocaciones hechas, tiene el conocimiento de su estado actual, es decir se puede consultar si aún están pendientes o no,

además de cuál fue su resultado. Para ello se recomienda que a cada petición sea asignado un identificador generado por este elemento.

Relación con otros elementos

- **Service:** Se hace uso de los Service para delegarle la ejecución de los servicios ofrecidos al Activity. Además debe estar a la escucha de la respuesta.

Implementación

A nivel de implementación este elemento es un singleton y no requiere ser una especialización de ninguna clase ni ser registrado en ningún archivo de configuración. Sólo debe declarar de forma pública los métodos que ofrece.

Debe encargarse de realizar la invocación a los Service a los que delega las tareas, para lo cual debe realizar el mapeo de los parámetros recibidos a un Intent, necesario para la invocación al Service. También debe implementar el mecanismo de callback para obtener la respuesta del Service pues su ejecución es asíncrona y el Service no tiene referencia del ServiceHelper.

2.4.1.2.3. Service

Descripción

Los Service son los elementos que aprovechan el potencial de los componentes Service de Android, permitiendo que los procesos sean ejecutados en background y con una prioridad superior a la que se logra por otros medios, con esto

aseguramos que el sistema operativo no elimine el proceso por falta de recursos, lo que aumenta la robustez de la aplicación.

Este elemento se encarga que la tarea a realizarse se ejecute en un hilo independiente, fuera del hilo principal de la aplicación, pero la tarea en sí es delegada a otro elemento; de forma que el Service sólo se encarga de crear el hilo donde será ejecutada dicha tarea, recibir los parámetros de la invocación y enviarlos al BusinessProcessor.

Un proyecto puede tener varios Service, según lo considere el diseñador, para no recargar un solo Service con funcionalidades o por motivos de performance y forma de trabajo como se verá más adelante.

Relación con otros elementos

- **BusinessProcessor:** El Service crea el hilo donde se ejecutará la tarea y hace el uso del BusinessProcessor para que se encargue de llevarla a cabo, enviándole los parámetros que a su vez recibió.

Implementación

Este elemento debe ser una clase que herede del Service proporcionado por la plataforma Android, clase especializada en ejecutar procesos en background, sin precisar de una interfaz de usuario.

Básicamente se tienen dos opciones para implementar este elemento. La primera es crear una clase que herede de Service, sin embargo esto no es suficiente para tener una tarea ejecutándose en background, ya que un Service se ejecuta dentro del hilo principal de la aplicación, por lo cual se debe

crear explícitamente un Thread (el cual es conocido como un worker thread). La segunda opción es crear una clase que herede de IntentService, que es una especialización de Service y que ya se encarga de crear un hilo fuera del hilo principal, pero tiene la particularidad que todas las invocaciones a un IntentService son encoladas, de manera que todo el tiempo sólo hay como máximo una invocación siendo atendida. Esto puede ser adecuado para una aplicación, en caso no se requiera paralelismo y no se desee abarcar muchos recursos del móvil, además de facilitar algunos problemas de concurrencia. Sin embargo en otras ocasiones sí se desean tareas en paralelo, para ello se puede tener varios IntentService o seguir la primera opción descrita, utilizando simplemente la clase Service.

En ambos casos la clase debe ser declarada en el archivo Manifext.xml, Android se encargará de su instanciación y su ciclo de vida y para invocarla se debe hacer uso de Intents de la siguiente forma:

```
Intent intent = new Intent(context, MiService.class);

context.startService(intent);
```

Los Service deben recuperar los datos de la invocación que vienen como extras del Intent que recibe y enviar estos datos al BusinessProcessor como una invocación normal a un método. Uno de los parámetros que recibe debe ser el objeto que sirva como mecanismo de callback para notificar el resultado o progreso de la tarea.

2.4.1.2.4. AsyncTask

Descripción

Los AsyncTask son una alternativa a los Service para ejecutar procesos en background aunque no comparten su nivel de prioridad. Estos componentes suelen estar muy ligados a los Activity ya que cuentan con métodos que facilitan la tarea de actualización de la interfaz de usuario, la cual se complica por el hecho que un AsyncTask se ejecute en un hilo independiente del hilo principal. Como en el caso de los Service, su principal tarea es crear el hilo independiente (lo cual hace implícitamente el AsyncTask) para evitar que tareas largas bloqueen la aplicación, impidiendo al usuario interactuar con la interfaz, pero la tarea en sí es delegada al BusinessProcessor. El uso de AsyncTask es más sencillo que el de los Service, por eso puede decidirse usar esta opción pero hay que considerar que como se mencionó no tiene su nivel de prioridad.

Relación con otros elementos

- **BusinessProcessor:** El AsyncTask crea el hilo donde se ejecutará la tarea y hace el uso del BusinessProcessor para que se encargue de llevarla a cabo, enviándole los parámetros necesarios.

Implementación

Este elemento debe ser una clase que herede del AsyncTask proporcionado por la plataforma Android, implementando los métodos que exige, clase utilizada para tareas asíncronas, como su mismo nombre lo indica. No precisa de ser declarado en el archivo Manifest.xml. No se necesita tampoco crear un

hilo dentro de esta clase ya que este es creado por defecto de forma implícita.

Se dice que su implementación es más sencilla que la de un Service porque no requiere de llenarse un Intent con los parámetros a enviarle, sino que simplemente se invoca a su método de ejecución. Además tampoco se requiere de un mecanismo de callback puesto que ya cuenta con métodos específicos para la comunicación de vuelta con el hilo principal de la aplicación.

Una consideración a tener en cuenta es que si el AsyncTask es usado desde un Activity, éste se encuentra muy amarrado a dicho Activity, lo que puede causar que el AsyncTask sea destruido en caso se destruya el Activity. Es por esto que se recomienda usarlo sólo para tareas cortas.

2.4.1.2.5. BusinessProcessor

Descripción

Los BusinessProcessor son los elementos que contienen la lógica de negocio de la aplicación, alejándose de temas de componentes propios de la plataforma Android para concentrarse en las tareas propias de los requerimientos de la aplicación. Estos elementos pueden ser traídos fácilmente de la capa de negocio de otras plataformas como la web sin tener que realizar muchas modificaciones para adaptarlas a la plataforma móvil.

Un proyecto puede tener varios de estos elementos, separándolos según procesos de negocio, funcionalidades, etc. Mantener las tareas agrupadas en varios BusinessProcessor permite que cada uno de estos elementos mantenga un alto

nivel de cohesión, facilita la asignación de trabajos a los desarrolladores y el mantenimiento de la aplicación.

Dentro de estos elementos es común encontrar lógicas complejas, manejo de entidades de negocio, persistencia de datos y comunicaciones con servidores a través de la red, entre otras cosas. Por ejemplo, en el caso de una aplicación de toma de pedidos se podría contar con un BusinessProcessor llamar PedidoProcessor, que se encargue de agregar ítems al carrito, validar cantidades, enviar pedido al servidor, etc.

Relación con otros elementos

- **DAO:** Hace uso de estas entidades para leer o modificar datos en un medio permanente.
- **Bean:** Utiliza los beans para manipular la información de las entidades de negocio propias de la aplicación.
- **Net:** Utiliza los elementos encargadas de realizar intercambio de información a través de la red.

Implementación

Los BusinessProcessor no necesitan heredar ninguna clase en particular ni ser declarados en el Manifest.xml, sólo tienen que contar con los métodos necesarios para las funcionalidades de negocio que implementen. Cada método que sea invocado desde el Service debe tener como parámetro de entrada el objeto callback que será utilizado para notificar la finalización o el progreso de la tarea a realizarse. Si el BusinessProcessor fue invocado desde un Service se estará ejecutando dentro de un hilo aparte, lo que quiere decir que no hay problema con que cuenten con tareas de larga duración.

2.4.1.2.6. Application

Descripción

El elemento Application se encarga de mantener el estado global actual de la aplicación, en este se puede agregar variables como por ejemplo el usuario actual, el último producto elegido, el cliente actual, etc. según los requerimientos específicos de la aplicación a desarrollarse. Trabajar de esta manera nos permite contar con algo similar a una sesión, como se trabaja en las aplicaciones web. Mantener aquí variables como las mencionadas facilita compartir información entre los componentes de la aplicación, evitar información repetida y simplifica las invocaciones entre componentes.

Además de mantener variables globales este suele ser un buen lugar para activar o desactivar ciertas funcionalidades de la aplicación como un tracking gps, limpieza de memoria cuando el usuario cierra sesión, etc.

Relación con otros elementos

- **Bean:** Se hace uso de los Beans que encapsulan las entidades de negocio de la aplicación y serán almacenados como variables globales.

Implementación

El elemento Application es una clase que hereda de la clase Application brindada por el sdk de Android a todas las aplicaciones desarrolladas. Además de esto deberá realizarse la configuración correspondiente en el archivo Manifest.xml

El sistema operativo se encargará de la instanciación de esta clase cuando la aplicación sea iniciada y se mantendrá viva

durante todo el ciclo de vida de la aplicación, esto la convierte en un lugar seguro para almacenar el estado actual de la aplicación. Para la obtención de una instancia de esta clase, que cabe remarcar que es un singleton, se necesita de un objeto Context y se hace de la siguiente forma:

myApplication *= (MyApplication)* ***context****.getApplication();*

2.4.1.2.7. DAO

Descripción

El elemento DAO es el encargado de la persistencia de los datos utilizados en la aplicación en alguno de los medios brindados por Android. Como su nombre lo indica este elemento sigue el patrón de diseño DAO de acceso a datos, lo que significa que esta clase es la encargada de manejar la conexión al medio de almacenamiento utilizado y la modificación de los datos.

Las principales opciones son utilizar la base de datos SQLite que viene por defecto instalada en el sistema operativo, las preferences que son un mecanismo rápido para grabar configuraciones en Android o el uso de archivos. La elección de qué mecanismo utilizar depende de aspectos como la complejidad de datos a almacenar, la cantidad, la estructura que tenga, etc. pudiéndose usar varios de ellos en un mismo proyecto. La ventaja de contar con este elemento es que elementos como el BusinessProcessor no tienen que enterarse de cómo se implementa el almacenamiento de datos.

Se recomienda tener varios DAO en el proyecto, uno por cada entidad de negocio, aunque también pueden ser agrupadas las funcionalidades de persistencia de varias entidades en un solo

DAO si no son muchas y no se aumenta demasiado la complejidad del DAO.

La grabación o modificación de datos en los DAO suele ir de la mano con un proceso de replicación de los datos en el servidor. Esta replicación no debería ser transparente para la experiencia del usuario.

Implementación

La implementación de estos elementos varía según la opción elegida de las presentadas en líneas anteriores. Normalmente se incluyen métodos para cada tarea básica de insertar, actualizar, eliminar y recuperar los datos. También es recomendable que los valores retornados desde los medios de almacenamiento sean mapeados a los Beans correspondientes según cada entidad.

2.4.1.2.8. Bean

Descripción

Los Beans son los encargados de representar las entidades de negocio según el proyecto, ellas encapsulan los datos y facilitan su manejo e intercambio entre los diferentes componentes. Estos Beans suelen ser replicas de los Beans que se tiene en otras plataformas como en el backend web, en estos casos no siempre es necesario que se manejen todos los atributos en el móvil sino un subconjunto de ellos, puesto que las funcionalidades de la aplicación web suele ser más limitada, así se evita aumentar el volumen de datos manejados innecesariamente.

Implementación

A nivel de implementación los Beans son clases sencillas en java, simples POJOs que no requieren implementar ninguna interfaz en especial. Mapean mediante atributos toda la información que se requiere de las entidades de negocio. Adicionalmente se recomienda manejar un identificador móvil a estas entidades, además del identificador que se suele traer de la web. También tener un flag que indique el estado de sincronización de cada Bean con el servidor para facilitar el proceso de replicación.

2.4.1.2.9. Net

Descripción

Este elemento es el encargado de realizar las comunicaciones de la aplicación a través de la red. Normalmente esto significa enviar y recibir datos hacia y desde un servidor en la web. Esta es una tarea que requiere ciertas configuraciones según el mecanismo a utilizar por lo cual esta clase encapsula un trabajo que probablemente tendría que ser replicado en varios puntos. Además debe encargarse de manejar los múltiples escenarios que pueden presentarse al querer hacer peticiones a través de la red, permitiendo que la aplicación siga su funcionamiento normal aunque dichas peticiones no sean exitosas, por ejemplo si se encuentra en una zona de no cobertura.

Implementación

La implementación de estas clases suele ser mediante el uso de librerías nativas de Android para conexiones http o utilizando librerías de terceros como es el caso de la librería de Spring para hacer invocaciones a servicios web de tipo REST.

2.4.1.3. Comportamiento

Los siguientes son 2 de los escenarios principales que suceden típicamente dentro de una aplicación móvil y que servirá para mostrar cómo funciona la arquitectura propuesta, con esto se busca facilitar su entendimiento y uso en proyectos a desarrollarse.

2.4.1.3.1. Grabación en base de datos móvil

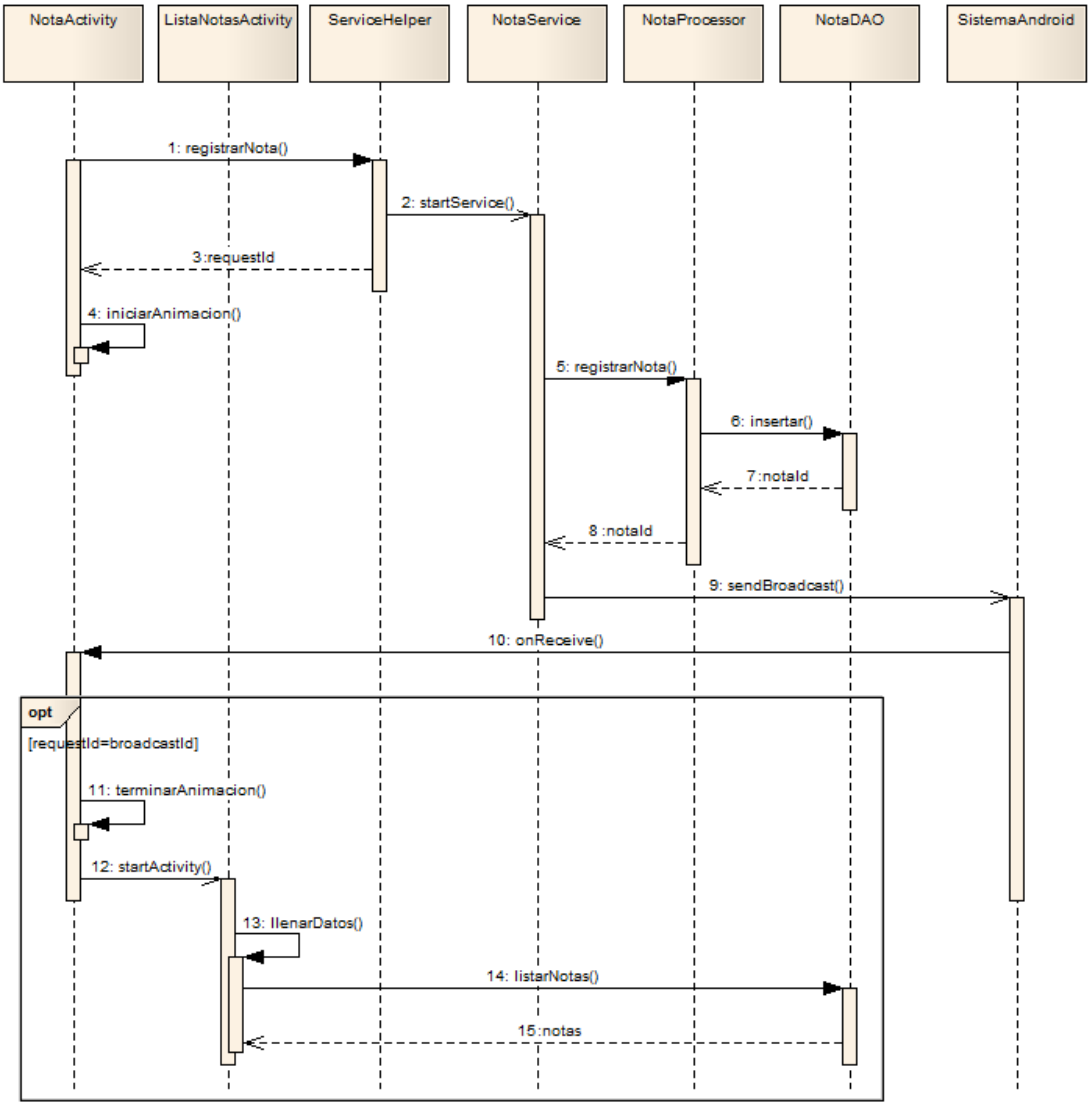


Ilustración 29: Diagrama de secuencia – Grabación en base de datos móvil

El primer escenario mostrado es dentro del contexto de una aplicación para tomar notas, el registro de una nota dentro de la memoria local del teléfono. Se detallan los pasos del diagrama:

1: El Activity es el que inicia el proceso de registro. En este caso la clase es llamada NotaActivity. Después de capturar los

datos ingresados mediante los View debe hacer uso del ServiceHelper que le sirve de facade, a través de una interfaz sencilla que es el método registrarNota(), el cual le oculta toda la complejidad de la tarea. Dentro del NotaActivity puede incluirse una validación de datos.

2: El ServiceHelper recibe la petición de realizar la tarea de registro y se la delega al NotaService. Deberá armar el Intent con los valores recibidos y utilizar el método startService(). Es una llamada asíncrona por lo tanto el ServiceHelper no debe esperar a obtener una respuesta.

3: El ServiceHelper se encarga de generar un id único para cada request, el cual es el valor de retorno para el NotaActivity. Ambos deben almacenar este valor. El ServiceHelper para llevar un control sobre los request que va atendiendo y el Activity para poder consultar más adelante el resultado de su request.

4: Al saber que se está realizando la tarea de registro el NotaActivity puede mostrar una animación en pantalla para dar un feedback al usuario, esto puede ser por ejemplo mostrar un dialog de progreso.

5: El NotaService se encarga de crear un hilo auxiliar para ejecutar la tarea de forma paralela. Obtiene los parámetros del Intent que recibe al ser invocado y delega la tarea al NotaProcessor utilizando su método registrarNota().

6: El NotaProcessor tiene la lógica del registro de la nota, lo que principalmente es hacer uso de NotaDAO, invocando al método registrar().

7: El NotaDAO es el responsable de manejar la persistencia de las notas en memoria no volátil, mayormente se utiliza la base de datos sqlite.

8: Como resultado del registro se obtiene el id asignado en la base de datos a la nueva nota. Este valor sirve también para saber si se registró correctamente.

9: Lo siguiente es notificar a los interesados que se ha terminado de realizar la tarea de registro. Para eso se empaqueta en un Intent los datos que se quieran comunicar y se lanza el broadcast utilizando el método sendBroadcast().

10: El sistema operativo Android se encargará de notificar a los listeners registrados para el broadcast invocando a su método onReceive().

11: El NotaActivity deberá validar que el requestId que recuperó al invocar al ServiceHelper sea el mismo que el que viene con el Broadcast, esto para asegurarse que la tarea terminada sea la misma que el Activity solicitó. De ser así puede retirar la animación de progreso y mostrar algún mensaje.

12: Como consecuencia al registro de la nota, se puede cambiar el Activity actual por el ListaNotasActivity, para ello se usa le método startActivity().

13: El ListaNotasActivity contiene, como su nombre lo indica, una lista con todas las notas registradas. Para llenar esta lista hace uso de su propio método llenarDatos().

14: Para obtener los datos el Activity hace uso del NotasDAO, invocando a su método listarNotas().

15: El valor devuelto es la lista de notas registradas, incluida la que se acaba de registrar. Con estos datos de podrá llenar la lista del Activity.

2.4.1.3.2. Envío de datos a servidor

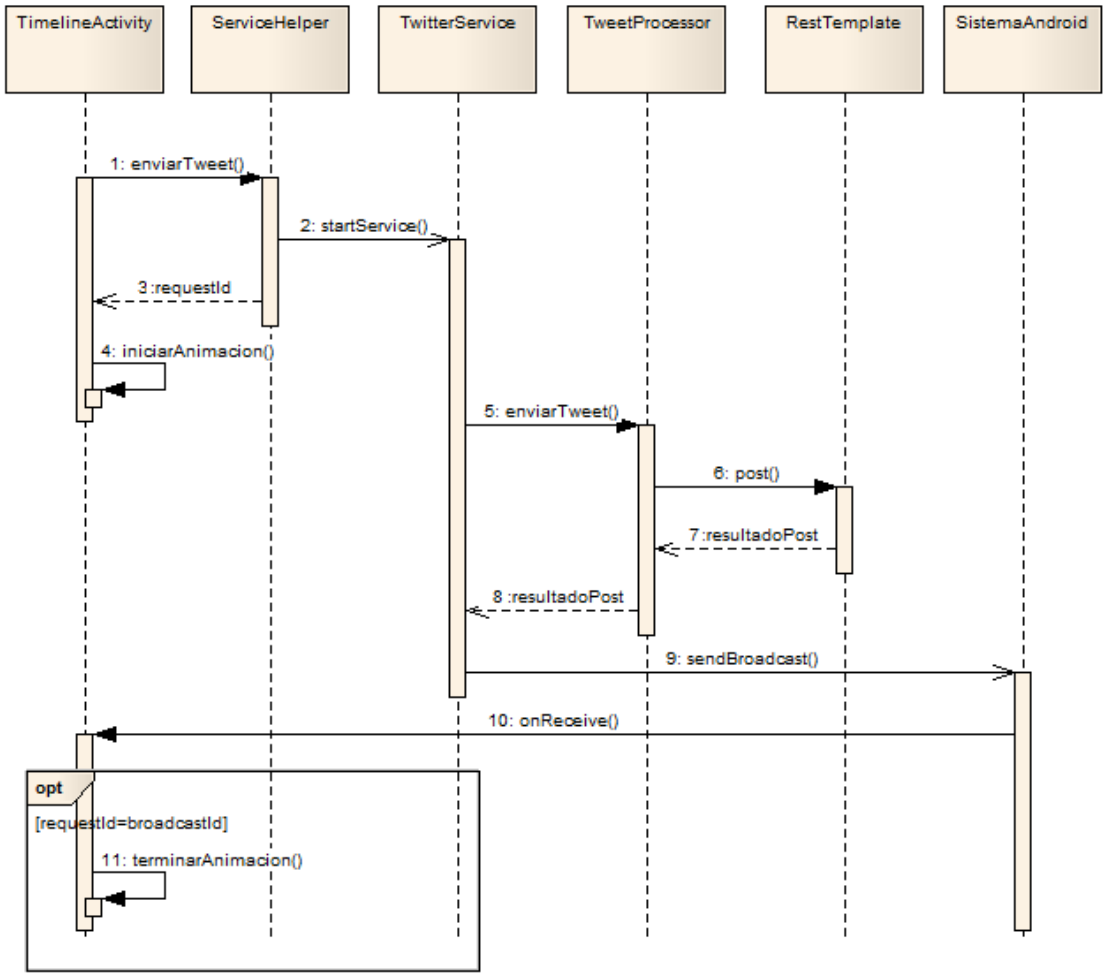


Ilustración 30: Diagrama de secuencia – Envío de datos a servidor

El segundo escenario mostrado, que utiliza muchos de los mismos elementos que el primero, es dentro del contexto de un cliente para Twitter, se encarga de enviar un Tweet por la red utilizando un servicio web REST. Se detallan los pasos del diagrama:

1: El TimelineActivity es donde se muestra el timeline del usuario y además cuenta con un dialog para escribir un nuevo Tweet. Cuando se ha ingresado los datos del Tweet se hace uso del método enviarTweet() del ServiceHelper.

2: El ServiceHelper delegará la tarea del registro al TwitterService, enviando un Intent y haciendo uso del método asíncrono startService().

3: El ServiceHelper se encarga de generar un requestId, que es devuelto al Activity y sirve para llevar control del request.

4: El TimelineActivity puede mostrar una animación mientras los datos son enviados por la red. Por ejemplo es común mostrar un icono girando en la parte superior de la derecha de la pantalla, lo que indica que hay un proceso en acción. Para esto se invoca al método interno iniciarAnimacion().

5: El TwitterService se encarga de crear el hilo auxiliar y delegar la tarea al TweetProcessor, invocando a su método enviarTweet.

6: El TweetProcessor es el responsable del envío del tweet al servidor. Para esto hace se encarga de armar los datos a enviar por REST, principalmente los datos del tweet, y hace uso de la clase RestTemplate, que forma parte de la librería de Spring para consumir servicios web REST (podría utilizarse cualquier otra librería). En este caso se hace uso del método post().

7: El valor recuperado es la respuesta del servidor ante la petición web enviada.

8: El TweetProcessor retorna al TwitterService el resultado del envío del Tweet.

9: Para notificar a los listeners de la finalización del envío del tweet se crea un Intent con los datos obtenidos y se envía el Broadcast haciendo uso del método sendBroadcast().

10: El sistema operativo Android se encarga de notificar a los listeners enviando el Broadcast e invocando el método onReceive de dichos listeners.

11: El TimelineActivity valida que la tarea completada sea la que solicitó para la publicación del post. De ser así puede actualizar su interfaz quitando el icono de tarea en proceso.

2.4.1.4. Fundamentación

Se presentan las principales decisiones arquitectónicas tomadas:

2.4.1.4.1. Estructura basada en el modelo de intercambio de datos - Google IO

La base de la estructura de la arquitectura propuesta sigue la estructura presentada por Google en su conferencia IO 2010 como parte de su modelo de intercambio de datos para consumir web services de tipo REST. Se pudo ver que el potencial de la estructura que utilizan va más allá de ese consumo de web services. El núcleo de nuestra arquitectura consiste en una adaptación de los elementos presentados en dicho modelo, extendiéndolo para que más allá de un intercambio de datos sirva para todas las funcionalidades típicas en una aplicación móvil, ya que conceptos como la ejecución de tareas en background, uso de Services o persistencia de datos son aplicables a una gran cantidad de proyectos. La extensión consiste en agregarle elementos

tomados de otros modelos como el descrito en la aplicación YAMBA de uso de una clase Application para la gestión global del estado de la aplicación y algunas adaptaciones como en el caso de los BusinessProcessor, que abarcan más que los Processor de Google, cuya tarea era sólo actualizar el estado de sincronización de una entidad, ahora se convierten en los responsables de almacenar la lógica de negocio de la aplicación.

2.4.1.4.2. Nomenclatura de los elementos

Los nombres elegidos para los elementos también están basados en lo presentado en Google IO 2010, con algunas variaciones como se mencionó para el caso del BusinessProcessor. Los elementos presentados serán convertidos en clases a nivel de código, la mayoría de los nombres son genéricos y se deberá agregar alguna palabra más que indique su funcionalidad, por ejemplo para el elemento Service, un proyecto puede tener la clase SincronizacionService o PedidoService, de igual forma para los demás elementos. Para el caso de los singleton no hay problema con usar el nombre presentado como para el ServiceHelper.

2.4.1.4.3. Uso de Service y AsynkTask

Existe mucha confusión entre el uso de estos dos componentes y frecuentemente los desarrolladores se preguntan cuál es la mejor opción porque aparentemente ambos permiten ejecutar tareas en segundo plano y evitan que ésta bloquee la interfaz

de usuario o continúe ejecutándose aunque la aplicación sea pausada.

La arquitectura es flexible en que presenta ambas opciones. Lo que se recomienda es utilizar AsyncTask para tareas cortas (como grabar datos de manera local), es más fácil de implementar y puede interactuar fácilmente con la interfaz del Activity. Por otro lado los Service pueden ser utilizados para tareas más largas a ejecutarse en segundo plano (como enviar daos a un servidor web), es un poco más complejo en su implementación pero brinda una mayor robustez a la aplicación.

2.4.1.4.4. Lógica de negocio fuera de Activity

Una práctica común en el desarrollo de aplicaciones Android es poner la lógica de negocio en los Activity, esto tiene que ver con la conceptualización de que un Activity no es una interfaz de usuario, sino que representa una funcionalidad provista por la aplicación. El problema de hacerlo de esta manera es que se carga muchas responsabilidades a un solo elemento, dañándose la alta cohesión y bajo acoplamiento, además de hacerse más difícil la asignación de tareas y el mantenimiento de sistemas. Basado en esto es que se decidió quitar la responsabilidad de la lógica de negocio y asignarla a otro elemento dedicado, dejándose sobre todo lógicas referentes a la validación de datos, además de todo lo referente a la experiencia visual del usuario. Sin embargo se puede incluir partes de lógica de negocio en el Activity, por flexibilidad y velocidad de desarrollo, pero siempre considerando no incluir tareas que consuman mucho tiempo en ejecutarse. Otro criterio para no incluir la lógica de negocio

aquí es porque se está amarrando algo que debe ser reutilizable con un elemento muy propio de la plataforma de implementación.

2.4.1.4.5. Uso de Application para variables globales

Es común tener en las aplicaciones móviles la necesidad de contar con un lugar donde almacenar las variables que representen el estado actual de la aplicación, como se hace con las variables de sesión en las aplicaciones web. Existen pro y contras de usar este enfoque, algunos autores dicen que se aumenta el acoplamiento y dificulta la reutilización de los Activity, lo cual hasta cierto punto es cierto; sin embargo mantener estas variables globales simplifica el desarrollo y evita el tedioso trabajo de enviar siempre dichas variables de elemento en elemento, pensando en esto la arquitectura recomienda contar con variables globales, el enfoque propuesto en otras fuentes. El siguiente punto es dónde almacenar dichas variables, crear una clase singleton parecería suficiente, incluir los atributos necesarios en dicha clase y permitir a todos los demás elementos acceder a ellos y modificarlos. Google recomienda utilizar una extensión de la clase Application, toda aplicación Android cuenta con una. La gran ventaja de utilizad esta clase es que se encuentra explícitamente atada al ciclo de vida de la aplicación, lo que significa que se mantiene con vida desde el inicio de la aplicación hasta su fin, de esta forma se asegura que las variables globales no se perderán.

2.4.2. Vista de Ejecución

Las vistas de ejecución, que algunos autores llaman vistas tipo Component-and-Connector, muestran cómo se estructuran un conjunto de elementos en tiempo de ejecución. En este caso los elementos que conforman la vista son procesos e hilos, por mencionar los que se utilizan en esta investigación, además de los conectores complejos que dan información sobre cómo se conectan estos elementos. Este tipo de vista se enfoca en los mecanismos de interacción, llamadas locales o remotas e invocaciones síncronas y asíncronas. Esta vista demás nos permitirá examinar aspectos de performance y confiabilidad de los componentes elegidos dentro de la plataforma Android.

Para esta vista se utiliza un tipo de vista de ejecución llamada de publish-subscribe, la cual se enfoca en los eventos de comunicación entre los elementos. Los mensajes enviados desde el publicador deben llegar a todos los subscritos que están a la escucha de un evento. En este caso particular los componentes son los hilos en ejecución de la aplicación. El flujo de eventos considerados es el de mensajes de resultado de la operación realizada, el retorno; las invocaciones de ida no presentan complicación y fueron explicadas en la vista anterior. Por ejemplo, los hilos auxiliares deben notificar al hilo principal que han terminado una tarea para que este realice una actualización en la interfaz de usuario.

2.4.2.1. Representación Gráfica

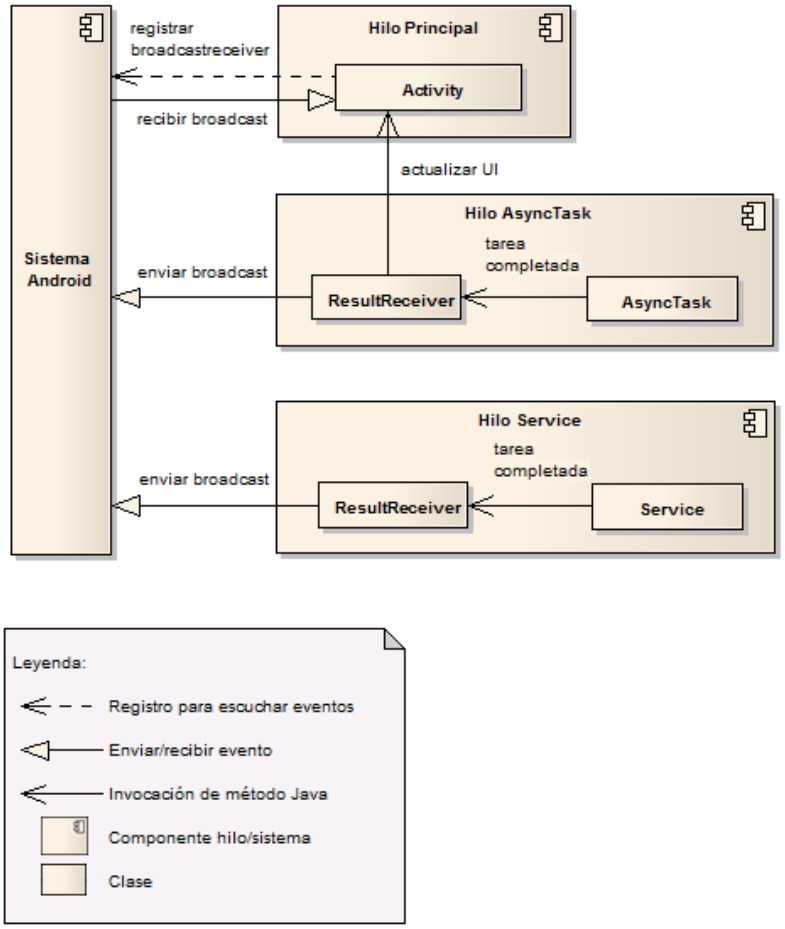


Ilustración 31: Representación Gráfica de la Vista de Ejecución

2.4.2.2. Catálogo de Elementos

A continuación se describen los elementos presentes en esta vista. Para cada elemento se presenta su descripción, datos de implementación y la forma en que se relaciona con los eventos de publicación y subscripción.

2.4.2.2.1. Sistema Android

Descripción

El mismo sistema de Android funciona como el conector en esta vista, es a través de su mecanismo de Broadcast, que actúa como el bus de eventos generados. Es el encargado de anunciar y escuchar a los componentes que desean publicar o subscribirse a eventos. Mediante los Broadcast se permite la comunicación entre diversos elementos que la plataforma Android que tengan acceso a un objeto Context. Los elementos que deseen ser notificados de algún evento deberán primero registrarse como listeners, cuando se genere el evento deseado el sistema Android se encargará de notificar a todos los listeners que tenga registrados; este registro puede hacerse de forma dinámica. Los publicadores sólo deberán informar al sistema de los eventos que quieran notificar y éste se encargará de notificarlo. Básicamente los Activity (subscriptor) se registrarán al sistema operativo y los Service (publicador) pedirán la notificación de eventos.

Implementación

No es necesario realizar ninguna implementación para este elemento ya que el propio sistema operativo Android implementa el mecanismo de Broadcast.

2.4.2.2.2. Hilo Principal

Descripción

El hilo principal, también conocido como hilo de UI, es donde se ejecutan por defecto todas las instrucciones de la aplicación y es creado al iniciarse la misma. Como se indicó en la vista

anterior es mejor reservarlo para tareas de interfaz de usuario y ejecuciones de código cortas para evitar que la aplicación se bloquee y no permita al usuario interactuar con ella, o que el sistema operativo lo entienda como un problema y lance un error de tipo ANR (Application Not Responding).

El principal componente que se ejecuta dentro de este hilo es el Activity, el cual debe delegar las tareas complejas a los Services. Al ejecutarse de manera asíncrona dentro de hilos auxiliares se requiere un mecanismo de comunicación con dichos hilos auxiliares para conocer el resultado de la ejecución de las tareas. Se opta por utilizar el mecanismo de Broadcast de Android. Para esto los Activity deben registrarse a los eventos que quieran escuchar, los cuales están asociados a las tareas que solicitan a los Service. Al hacerlo de esta forma se logra que los Activity y los Service no estén fuertemente acoplados, ya que no tienen referencias el uno al otro. El mecanismo de publicador subscriptor permite que al terminarse una tarea sea notificado un listener diferente al que inició la tarea, sin que esto impacte en el Service; la aplicación podría haber cambiado de Activity mientras se ejecuta la tarea o la instancia del Activity haya sido terminada por el sistema operativo y actualmente se esté utilizando una instancia diferente, pero también registrada como listener, y la aplicación seguirá funcionando con normalidad. El Activity recibirá una notificación cada vez que acontezca el evento al que está registrado. En ese momento podrá actualizar la interfaz, por ejemplo actualizar la lista de productos que se ha actualizado consultando en background al servidor.

Implementación

Para implementar la subscripción de los Activity a eventos hay que utilizar la interfaz de BroadcastReceiver brindada por

Android. El Activity debe registrarse utilizando un objeto Context (el mismo Activity hereda de Context). De estarse utilizando Fragments se puede utilizar como Context el Activity al que pertenecen.

El registro consta de 2 elementos, un filtro y una acción. El filtro indica qué broadcasts debe escuchar el Activity, es decir, qué eventos son de su interés. Para ello se utiliza la clase IntentFilter. La acción es el código a ejecutarse cuando se recibe la notificación del evento. Se hace uso de la interfaz BroadcastReceiver, la cual requiere de un método con las acciones a realizarse, normalmente actualizaciones de interfaz de usuario.

El Activity pasa por varios estados durante su ciclo de vida. Sólo debe mantenerse registrada a la escucha de eventos mientras se encuentre activa, de lo contrario Android detectará un error al tratar de notificar a un Activity no activa. Es por ello que se debe registrar cada vez que se ejecute el método onResume y anular el registro en el método onPause. Un punto a tomar en cuenta es que si el evento se concluyó mientras el Activity estaba pausado, al resumirse no se enterará que ocurrió la notificación y podría quedar esperando por un evento que ya pasó. Para evitar esto, al resumirse debería consultar al ServiceHelper si un request determinado ya ha concluido y cuál fue el resultado, para ello se generó un identificador por request como se explicó en la vista anterior.

El Activity recibirá un Intent al ser notificado de un Broadcast, dentro de los cuales se encontrarán los valores de resultado, enviados por el publicador. Se recomienda no llenar el Intent con los valores a utilizarse para poblar la interfaz sino que sólo indicar, por ejemplo, que se terminó correctamente una tarea y

que el Activity se encargue de obtener los valores recuperados utilizando un DAO.

Eventos

- **Recibe:** Notificaciones de terminación de tareas delegadas a los Service para poder actualizar la interfaz de usuario.

2.4.2.2.3. Hilo Service

Descripción

Como se mencionó los Service se encargan de crear hilos auxiliares para la ejecución de tareas largas y complejas. Dentro de una aplicación se pueden crear múltiples hilos auxiliares según sean requeridos, estos hilos continúan ejecutándose aunque la aplicación no sea visible y tienen un alto nivel de prioridad de manera que es poco probable que el sistema operativo los elimine cuando necesite liberar recursos.

Estos hilos auxiliares requieren un método para comunicar al hilo principal la finalización de las tareas asignadas. No puede realizar una invocación directa pues no tiene una referencia al Activity. En lugar de ello utiliza el mecanismo de Broadcast como un publicador. El Service se encarga de informar que ha concluido una tarea (ejecutada por el BusinessProcessor) y hay que realizar una notificación. La notificación a mandarse puede definirse en otro lugar, por ejemplo en el elemento ServiceHelper, para mantener el Service enfocado en la lógica de negocio. El envío del Broadcast puede estar definido en un nivel superior de invocación y utilizar un mecanismo de callback para que ese código sea ejecutado desde el hilo auxiliar.

Al utilizar el mecanismo de Broadcast el Service no se preocupa por quienes están a la escucha del evento ni qué acción realizarán al recibirlo. Esta invocación no es bloqueante así que puede continuar ejecutándose sin esperar a que sea recibida. El Service no tiene que notificar sólo la finalización de la tarea, también puede anunciar avances o progreso realizado.

Implementación

El hilo Service debe enviar el Broadcast al sistema operativo Android para que este notifique a los elementos subscritos. Como se mencionó la lógica de quién debe ser notificado debe ser ubicada en un elemento como el ServiceHelper y para invocarlo se utiliza un mecanismo de callback. De esta manera se puede separar el código del envío de la notificación con la lógica de negocio propia de la tarea a realizarse.

Para realizar este callback se usa la clase ResultReceiver, que es una inferfaz genérica para recibir un resultado de callback. Se implementa esta clase y se envía como parámetro al Service, el cual lo utilizará llenando un objeto Bundle con los datos que tenga sobre la ejecución de la tarea. Estos datos serán parte del Intent que debe armar el ResultRecever y que recibirá el elemento subscrito al ser notificado del Broadcast; por ejemplo, si se tratara de una tarea de sincronización se podría enviar un flag que indique si la sincronización fue exitosa o no. El ResultReceiver deberá utilizar un objeto Context y su método sendBroadcast para enviar la notificación.

Entre los datos que se envía con el Broadcast debe enviarse el id generado al iniciarse la tarea para que el Activity que está a la escucha del Broadcast pueda saber que el resultado obtenido

es para la tarea que ha solicitado. Además el ServiceHelper debería almacenar por contingencia los resultados obtenidos para cada tarea, así como su estado. Esto es para que el Activity pueda consultar el resultado de una tarea en caso haya perdido el Broadcast enviado por no encontrarse en estado activo.

Eventos

- **Envía:** Notificaciones de del progreso o terminación de tareas ejecutándose en el hilo auxiliar.

2.4.2.2.4. Hilo AsyncTask

Descripción

Como ya se ha mencionado, el segundo método provisto por la presente arquitectura para la ejecución de tareas en segundo plano es el uso de AsyncTask. También se ha dicho que los AsyncTask pueden ser más fáciles de implementar y se recomienda su uso para tareas cortas ya que no tienen el nivel de prioridad de un Service. Parte de esta facilidad se debe a que los AsyncTask cuentan con mecanismos especiales para comunicarse con el hilo principal de la aplicación de forma más directa, pero también puede seguir el mismo mecanismo de comunicación que usan los Service, a través de Broadcast, siguiendo el patrón de publicador subscriptor sin preocuparse por quienes están a la escucha de los eventos. Este último método toma unos pasos más de implementar pero brinda una mayor robustez.

Implementación

Hay dos maneras de implementar la comunicación del AsyncTask con el hilo principal de la aplicación. La primera es haciendo uso del Broadcast, esta implementación es similar a la indicada para un Service. También se puede utilizar la clase ResultReceiver para realizar una invocación de broadcast, sobre todo si el AsyncTask está siendo creado en el ServiceHelper. La otra manera es a través de los métodos onProgressUpdate y onPostExecute, para actualizar la interfaz de usuario cuando hay un avance en la tarea o esta ha sido finalizada respectivamente. Estos métodos tienen acceso directo a los Views que pertenecen al hilo principal. Cabe recalcar que en el caso de estos métodos su ejecución no es asíncrona así que el AsyncTask tendrá que esperar mientras se ejecuta

Muchas veces se incluyen como clases internas dentro de los Activity para poder actualizar directamente sus Views, a través de métodos con los que cuenta para esto. La contraparte de incluir el AsynkTask en el Activity es que se amarran mucho ambos elementos. Usar esta forma sólo si la tarea es corta y se puede asegurar que el Activity no será destruido, sino es así hay que tomar precauciones para los casos de excepción.

Eventos

- **Envía:** Notificaciones de del progreso o terminación de tareas ejecutándose en el hilo auxiliar.

2.4.2.3. Comportamiento

A continuación se presenta un escenario de comunicación para esclarecer la interacción entre los elementos presentado.

2.4.2.3.1. Registro y recepción de notificaciones Broadcast

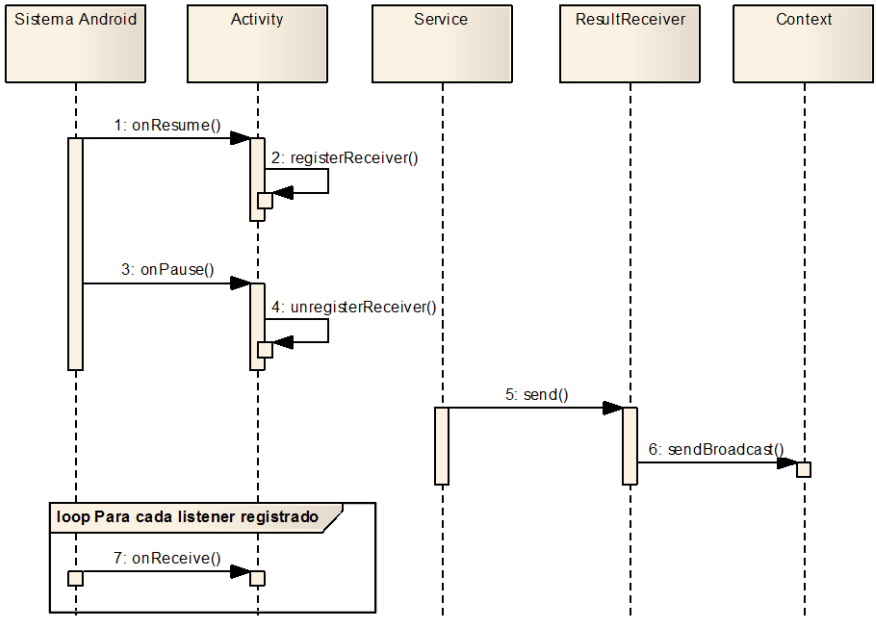


Ilustración 32: Diagrama de Secuencia – Registro y recepción de Broadcast

El diagrama presentado muestra el comportamiento de los elementos para desempeñar su tarea de subscriptor y publicador. Se detallan los pasos del diagrama:

1: Lo primero que debe hacerse para utilizar el patrón de subscriptor publicador es registrar al subscriptor. Este registro debe realizarse cuando el Activity se encuentre activa, por lo tanto debe ubicarse dentro del método onResume() que es invocado por el sistema Android cada vez que el Activity retoma el control de la pantalla.

2: Mediante el método registerReceiver() el Activity queda registrado para recibir los Broadcast de determinado tipo, según el filtro que se mande como parámetro; además manda un BroadcastReceiver con el código a ejecutarse al recibir la notificación, utilizando una implementación de la interfaz BroadcastReceiver, que puede ser el mismo Activity.

3: De igual manera, debe anularse el registro a los Broadcast cuando el Activity deje de estar en primer plano. Cuando esto suceda el sistema Android invocará al método onPause() del Activity.

4: Se utiliza el método unregisterReciever(), que es la contraparte del registerReceiver(), para quitar el registro de escucha de Broadcast.

5: Cuando el Service (que se ejecuta en un hilo auxiliar) termine de ejecutar una tarea solicitada requiere notificar al hilo principal. Para ello invoca al método send() del ResultReceiver, el cual es el mecanismo de callback para que el Service no tenga la lógica asociada al envío del Broadcast.

6: El ResultReceiver se encargará de armar el Intent con los datos a enviar en el Broadcast y hará uso de un Context para invocar al método sendBroadcast(), que se encarga de enviar el Broadcast.

7: Finalmente cuando el sistema Android detecta que hay un Broadcast pendiente se encarga de enviarlo a todos los listener registrados. La notificación consiste en invocar al método onReceive() perteneciente a la interfaz BroadcastReceiver, que fue creada al hacer el registro de receptor de Broadcast.

2.4.2.4. Fundamentación

Se presentan las principales decisiones arquitectónicas tomadas:

2.4.2.4.1. Uso de Broadcast como mecanismo de comunicación entre hilos

Al utilizarse hilos auxiliares en la arquitectura es necesario un mecanismo para la comunicación entre ellos. Se eligió el uso de Broadcast por ser un mecanismo de notificación de eventos que forma parte de la plataforma Android y por encontrarse implementada y gestionada por el mismo sistema operativo. Es relativamente fácil realizar el registro y la escucha de eventos a través de este canal. Este es el mecanismo sugerido por Google en el IO de 2010 en su modelo de intercambio de datos, el cual también requiere comunicar hilos auxiliares que realizan las invocaciones REST con los Activity para notificar la finalización de la comunicación con el servidor. Otra ventaja de utilizar este método es que se mantiene la separación entre elementos de interfaz y los elementos de lógica de negocio, así como la flexibilidad entre subscriptores y publicadores.

2.4.2.4.2. Múltiples hilos en la aplicación

La arquitectura propuesta recomienda la utilización de hilos separados para las tareas de larga ejecución con la finalidad de evitar bloquear la interfaz, mejorar la robustez de la aplicación para que esta se siga ejecutando aunque se encuentre en segundo plano y hacer la aplicación lo más transparente y permisiva para el usuario de manera que pueda continuar utilizándola sin saber que parte de la tarea aún se esté ejecutando, por ejemplo puede estar enviándose datos al servidor pero para la percepción del usuario esto es transparente. Es por esto la importancia del mecanismo de comunicación entre el hilo principal y los hilos auxiliares, gran parte de la arquitectura se basa en el uso de múltiples hilos. El número exacto de hilos que se ejecutarán dependerá mucho de

la naturaleza de la aplicación y a la necesidad de paralelismo que tenga. En general se recomienda no tener demasiados hilos ejecutándose en paralelo debido a la cantidad de recursos del teléfono que podrían consumir y a problemas de comunicaciones entre procesos más complejas que se tiene que atender. El uso de los IntentService limita el número de hilos necesarios ya que todas las tareas son encoladas en cada IntentService, una práctica que se recomienda en esta arquitectura.

2.5. Aplicación de la arquitectura en un proyecto

La arquitectura propuesta está destinada a ser usada durante la etapa de diseño de aplicaciones móviles en Android, por lo que consideramos los siguientes pasos para su aplicación en el desarrollo de un proyecto. Cabe señalar que el objetivo de la presente investigación no la creación de una metodología de desarrollo, y lo que se presenta son pasos recomendados a ser utilizados de manera independientemente de la metodología con la que se esté trabajando (ICONIX, RUP, XP, etc.).

1) **Tener análisis completo:** Al tratarse de una arquitectura de diseño, se requiere de tener completa la fase de análisis del proyecto o al menos de la iteración en la que se está trabajando.

2) **Definir los Activities y Fragment:** Estos componentes serán la interfaz visual con la que el usuario interactuará, capturando los eventos de ingreso y mostrando resultados de salida. Estos pueden ser identificados desde los casos de uso, prototipos y otros artefactos que se tenga. Depende del diseñador diferenciar los casos en que use Activities o Fragments. Se recomienda considerar uno por cada funcionalidad principal del proyecto, o por cada caso de uso, favoreciendo la reutilización.

3) **Identificar los procesos a ejecutarse:** Teniendo los Activities y Fragments lo siguiente será identificar los procesos a realizarse (grabar un dato en el

teléfono, enviar un dato al servidor, realizar algún procesamiento), los cuales pueden ser categorizados en síncronos, asíncronos, secuenciales o paralelos. Los procesos se podrán obtener en base a los requerimientos especificados en los casos de uso. Estos procesos realizan algún procesamiento o alguna conexión con el servidor.

4) **Definir los Service y AsyncTask:** En base a los procesos identificados y categorizados se puede crear los Service y AsyncTask necesarios, en base a los beneficios mencionados anteriormente que cada uno posee. Para los procesos paralelos se recomienda utilizar el AsyncTask y para los procesos asíncronos y secuenciales se recomienda utilizar los Services (En ambos casos se estará trabajando desde hilos secundarios). Estos componentes serán los encargados de sacar la ejecución del hilo principal y evitar problemas de rendimiento.

5) **Definir los demás elementos:** Posteriormente se debe definir los demás elementos pertenecientes a la arquitectura como Beans (según los objetos del modelo identificado, los cuales pueden ser un espejo de los beans en el servidor, dado el caso), DAOs (mayormente se utilizan DAOs que se conectan a una base de datos mysql, también se puede usar un Content Provider, Shared Preferences, o cualquier tipo de almacenamiento) y Business Processors (estos contendrán la lógica de la aplicación, pueden ser componentes reutilizados de otras aplicaciones, móviles o no).

6) **Crear diagramas de secuencia:** Se recomienda los diagramas de secuencia porque brindan mejor visibilidad de la interacción entre los componentes de diseño y son fácilmente entendibles al momento de hacer la programación. No se considera diagramas de colaboración pues se enfocan más en los elementos que en su interacción.

7) **Crear diagramas de paquetes y de clases:** Estos diagramas presentarán la vista estática del diseño realizado. El diagrama de paquetes tendría que ser muy similar al diagrama de paquetes presentado al inicio de este capítulo, aunque se puede agregar elementos según se considere adecuado. Este

diagrama guía el paso del diseño a la construcción del software. Se recomienda presentar el diagrama de clases agrupando los componentes de cada paquete a menos que se cuente con muy pocas clases, de esta forma se mejora la visibilidad de los componentes.

8) **Crear otros diagramas:** Dependiendo del caso se puede agregar otros diagramas al diseño, como diagramas de colaboración y despliegue.

9) **Elegir herramientas de prueba:** Se puede elegir herramientas con las que posteriormente será probada la implementación de lo diseñado en el proyecto. Recomendamos el uso de Android Testing Framework, el DDMS, SonarQube y Selendroid, las cuales fueron definidas previamente.

CAPITULO III: CASO

Lo que se plantea en este capítulo es usar la arquitectura adaptada propuesta en un caso de aplicación real. En este caso se aplicará dentro del contexto de un sistema web móvil para la gestión de servicios, seguimiento y control de unidades de una empresa de taxis.

Este proyecto consta de dos módulos, el módulo web que fue desarrollado utilizando Java y Mysql, con los frameworks Primefaces, Google Maps, Spring e Hibernate; y el módulo móvil que fue desarrollado en Android, que es donde se aplica la arquitectura formulada. Para la comunicación entre ambos módulos se utilizan web services REST.

Este sistema se utilizó debido a que el módulo móvil cuenta con varias funcionalidades donde se puede apreciar el uso de la arquitectura. Estas funcionalidades son genéricas para la mayoría de aplicaciones móviles así que mediante su exposición se pretende que puedan ser usadas como base para nuevos proyectos, además que mejoren el entendimiento de la arquitectura, contrastando lo propuesto con su aplicación en un caso real.

El objetivo al presentar este caso no es documentar todo el proceso de desarrollo del software, sino enfocarse en el diseño de la aplicación móvil, enfocándose en la estructura utilizada y el uso de los elementos descritos en el capítulo anterior, por lo cual algunas etapas o diagramas han sido obviados en la medida que no aporten directamente a la exposición de la arquitectura formulada. Además de lo concerniente al diseño se agregan los puntos considerados necesarios para el mejor entendimiento del caso.

3.1. Descripción del Caso

Una de los negocios que puede verse beneficiado por la tecnología móvil es el de servicio de taxis. Estando las unidades pertenecientes a la empresa esparcidas en una ciudad y en constante movimiento la única forma en que tengan acceso a una aplicación es teniéndola en su teléfono móvil; de esta forma pueden recibir y enviar información, como la ubicación geográfica exacta donde se encuentran o los servicios que tiene que atender, junto con datos como la dirección del cliente.

Además, en la actualidad contamos con un servicio brindado por la empresa Google llamado Google Maps con el cual podemos visualizar mapas de manera fácil e intuitiva en la web, el cual nos permite mostrar en un mapa dinámico la ubicación de un dispositivo procesando la información enviada por el GPS del equipo celular, a partir de esto y gracias a este servicio podemos realizar y generar información extra al usuario, información de mucha utilidad como el tracking, la posición actual, cercar áreas geográficas, etc.

El servicio brindado por las empresas de transporte de pasajeros propiamente dicho no ha evolucionado de manera notable durante años, el servicio viene siendo ofrecido al público de la misma manera, así mismo podríamos agrupar los procesos llevados a cabo de la siguiente manera:

- **Servicio Libre:** La unidad vehicular se encuentra en la ciudad y ofrece sus servicios al público, el pasajero que requiera del servicio sube al taxi, indica su destino, se procede a hacer el recorrido, una vez llegado al destino se procede a cobrar por el recorrido.
- **Servicio Asignado:** El pasajero llama a la central para solicitar un servicio de transporte, el encargado en la central llena un cuaderno de control de servicios asignados, hace un llamado a todos los taxis para verificar quien se encuentra disponible y/o cerca del lugar donde se brindara el servicio y asigna el servicio, el taxi procede a ir al lugar indicado, recoge el pasajero, lo lleva al destino y se cobra por el servicio brindado.
- **Control de Unidades:** La central llama cada cierto tiempo a las unidades para saber cuál es su posición y estado actual. Además cada unidad debe reportarse al momento de iniciar o terminar su jornada.

Como podemos observar en los procesos descritos anteriormente, se presentan los siguientes problemas e inconvenientes:

- La central de taxis no cuenta con la información necesaria y en tiempo real de las unidades.

- No se tiene información sobre la ubicación geográfica exacta de las unidades en el momento que se necesita tanto para el control de las unidades como para la asignación de servicios.
- El control de las unidades se hace manualmente, cada unidad debe reportar su posición y estado.
- No se registra la información de una manera persistente necesaria.
- No se lleva una cuenta de los servicios realizados por las unidades en el servicio libre.
- No se lleva una cuenta de los orígenes y/o destinos más comunes de los servicios brindados.

En conclusión, no genera información útil para la empresa. El presente caso se basa en utilizar los dispositivos móviles y sus características, que integrados a un sistema web permita generar información útil para la empresa.

3.2. Análisis de Requerimientos

3.2.1. Diagrama de casos de uso

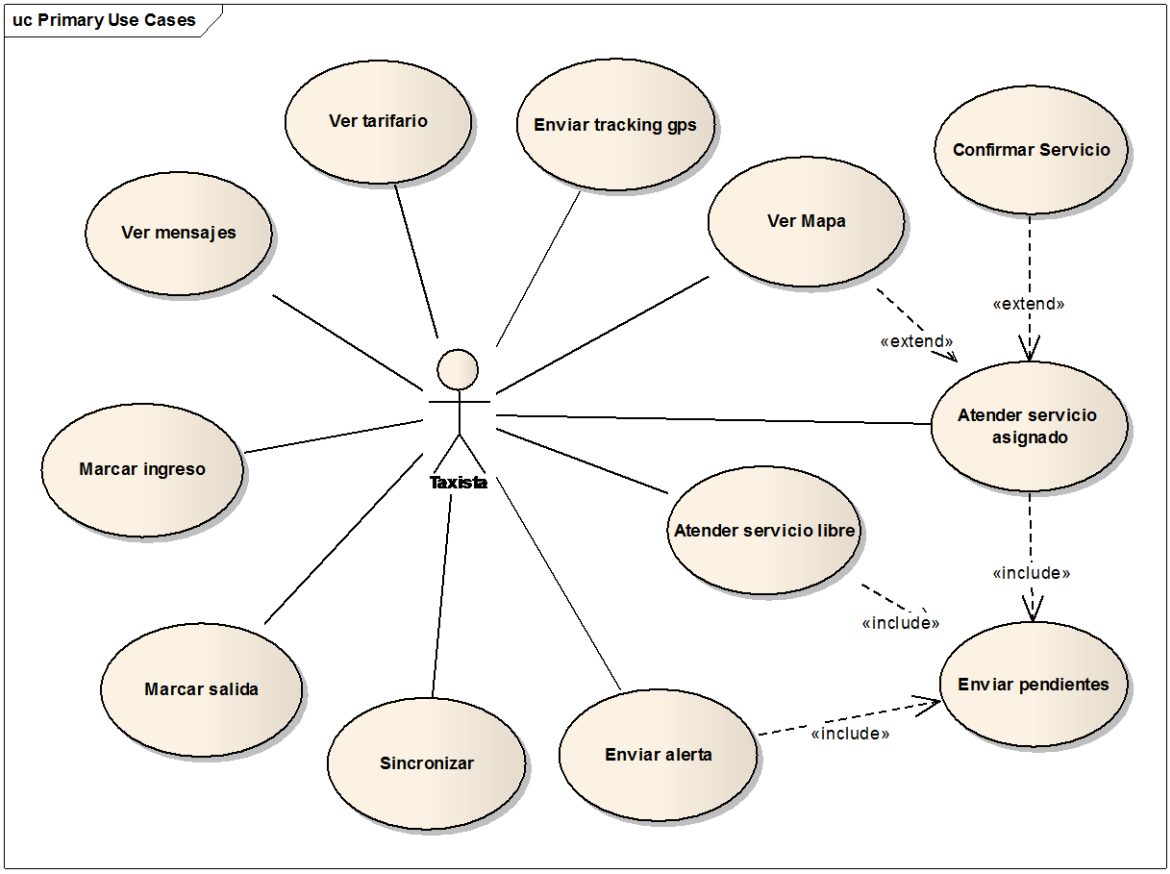


Ilustración 33: Diagrama de casos de uso del caso

3.2.2. Especificación de actores

Actor: Taxista	
Descripción	El taxista es el único usuario de la aplicación móvil, quien se encuentra en la ciudad realizando servicios de taxi y se comunica con la base usando su teléfono
Características	• El taxista se encuentra en una móvil recorriendo la ciudad. • Jornadas de trabajo de 12 horas aprox. • La aplicación no debe distraerlo del manejo de su vehículo y debe ser fácil de usar. • Conocimiento básico en uso de teléfonos inteligentes.

3.2.3. Especificación de casos de uso

Nombre de Caso de Uso: Login	
Descripción	Ingreso del taxista al sistema
Actor Principal	Taxista
Precondiciones	
Postcondiciones	Se ingresa al sistema.
Flujo Básico	1. El taxista inicia la aplicación en su móvil. 2. El taxista ingresa su código y su contraseña. 3. El sistema valida los datos del taxista. 4. El taxista ingresa a la aplicación.
Flujos Alternativos	3a El taxista ingresa un código o una contraseña inválidos. 1. El sistema muestra un mensaje de error y pide que intente de nuevo. 3b El equipo está fuera de cobertura. 1. El sistema muestra un mensaje de fuera de cobertura y pide que se intente luego.

Nombre de Caso de Uso: Marcar ingreso	
Descripción	Caso que describe el registro de la marcación de ingreso del taxista a la aplicación.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login.
Postcondiciones	Marcación de ingreso registrada.
Flujo Básico	1. El taxista ingresa al sistema después del login 2. El sistema muestra la lista de móviles disponibles a las que está asociado el taxista. 3. El taxista elige una móvil. 4. El sistema registra la marcación de ingreso. 5. El sistema muestra un mensaje de bienvenida. 6. El sistema muestra el menú principal de la aplicación.
Flujos Alternativos	2a No hay ninguna móvil disponible. 1. El sistema muestra un mensaje indicando al taxista que todas las móviles asociadas a él se encuentran ocupadas.

Nombre de Caso de Uso: Marcar salida	
Descripción	Caso que describe el registro de la marcación de salida del taxista a la aplicación.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login.

	• Se ha marcado ingreso.
Postcondiciones	Marcación de salida registrada.
Flujo Básico	1. El taxista elige la opción de marcar salida en el menú principal. 2. El sistema solicita la confirmación de la marcación de salida. 3. El taxista confirma la marcación. 4. El sistema registra la marcación de salida. 5. El sistema muestra un mensaje de salida. 6. El sistema cierra la aplicación.
Flujos Alternativos	3a El taxista no confirma la marcación. 1. El sistema vuelve a mostrar el menú principal.

Nombre de Caso de Uso: Sincronizar	
Descripción	Caso que describe la sincronización de datos entre el servidor y la aplicación móvil.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login. • Se ha marcado ingreso.
Postcondiciones	Datos sincronizados.
Flujo Básico	1. El taxista elige la opción sincronizar dentro del menú principal. 2. El sistema muestra la pantalla de sincronización, donde se muestran los elementos que se pueden sincronizar (alertas, zonas, estados y tarifas). 3. El taxista marca los elementos que quiere sincronizar (puede marcar todos para realizar una sincronización total o algunos para una sincronización parcial). 4. El sistema realiza la sincronización de los elementos seleccionados y los registra en el móvil (eliminando los datos anteriores). 5. El sistema muestra un mensaje de éxito en la sincronización.
Flujos Alternativos	3a El taxista no selecciona ningún elemento. 1. El sistema muestra un mensaje solicitando que se seleccione al menos un elemento para realizar la sincronización. 4a Hay un error al sincronizar los datos. 1. El sistema muestra un mensaje de error indicando que la sincronización no fue exitosa. 4b El equipo está fuera de cobertura. 1. El sistema muestra un mensaje indicando que no hay cobertura. 2. Los datos del equipo quedan como antes de elegir hacer la sincronización.

Nombre de Caso de Uso: Enviar tracking GPS	
Descripción	Caso que describe el envío de la posición gps del taxi de manera continua al servidor.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login. • Se ha marcado ingreso.
Postcondiciones	
Flujo Básico	1. El taxista ingresa a la aplicación. 2. El sistema valida que haya realizado su marcación de ingreso. 3. El sistema valida que el teléfono tenga activada la funcionalidad de GPS. 4. El sistema empieza a escuchar posiciones GPS del teléfono. 5. El sistema captura la posición GPS. 6. El sistema envía la posición GPS. 7. El sistema espera un margen de tiempo. *Se repiten los pasos 4-6 mientras la aplicación se encuentre activa.* 8. El sistema deja de escuchar posiciones GPS.
Flujos Alternativos	2a El teléfono no tiene activada la funcionalidad de GPS. 1. El sistema muestra un mensaje solicitando al taxista que active la funcionalidad de GPS del teléfono. 2b El equipo está fuera de cobertura 1. El sistema almacena en su memoria las posiciones no enviadas para enviar todas juntas cuando vuelva a tener cobertura.

Nombre de Caso de Uso: Atender servicio asignado	
Descripción	Caso que describe la atención de un servicio asignado al taxista, pasando a través de los diferentes estados de atención hasta terminarla.
Actor Principal	Taxista
Precondiciones	• El taxista elige la opción • Se ha marcado ingreso. • Se ha sincronizado datos.
Postcondiciones	Servicio terminado.
Flujo Básico	1. El taxista elige la opción de atender servicio en el menú principal. 2. El sistema muestra la lista de servicios asignados. 3. El taxista elige el servicio que desea atender. 4. El sistema muestra los datos del servicio elegido (código, nombre del cliente, fecha, estado actual,

	dirección, referencia), además de la opción de actualizar estado y ver mapa. 5. El taxista elige la opción actualizar estado. 6. El sistema muestra la lista de estados disponibles. 7. El usuario elige el estado deseado. 8. El sistema registra el estado del servicio. 9. Invoca al caso de uso "Enviar pendientes". *El taxista repite los pasos 5-8 hasta que se elija un estado de finalización.* 10. El sistema actualiza el servicio como terminado. 11. Invoca al caso de uso "Enviar pendientes". 12. El sistema muestra un mensaje de finalización del servicio y vuelve a la pantalla principal.
Flujos Alternativos	2a No hay ningún servicio asignado. 1. El sistema muestra un mensaje indicando que no hay servicios asignados sincronizados. 3a El usuario pulsa el botón de sincronizar. 1. El sistema realiza la sincronización de los servicios asignados. 2. El sistema muestra un mensaje de éxito. 4a El servicio asignado no ha sido confirmado por el taxista. 3. Invoca al caso de uso "Confirmar servicio". 5a El taxista elige la opción de ver en mapa. 1. Invoca al caso de uso "Ver mapa".

Nombre de Caso de Uso: Confirmar servicio	
Descripción	Caso que describe la confirmación por parte del taxista de un servicio que le ha sido asignado, ya sea positiva o negativamente.
Actor Principal	Taxista
Precondiciones	
Postcondiciones	Servicio confirmado
Flujo Básico	1. El sistema muestra los datos del servicio asignado, con las opciones de aceptar o rechazar. 2. El taxista elige la opción de aceptar. 3. El sistema registra los datos de confirmación. 4. Invoca al caso de uso "Enviar pendientes". 5. El sistema muestra los datos del servicio con las opciones de atención.
Flujos Alternativos	2a El taxista elige la opción de rechazar 1. El sistema registra los datos de confirmación 2. Invoca al caso de uso "Enviar pendientes" 3. El sistema muestra el menú principal del a aplicación.

Nombre de Caso de Uso: Atender servicio libre	
Descripción	Caso que describe la atención de un servicio libre (tomado en la calle) por el taxista, pasando a través de los diferentes estados de atención hasta terminarla.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login. • Se ha marcado ingreso. • Se ha sincronizado datos.
Postcondiciones	Servicio terminado.
Flujo Básico	1. El taxista elige la opción de atender servicio en el menú principal. 2. El sistema muestra la lista de servicios actuales. 3. El taxista elige el botón de nuevo servicio. 4. El sistema registra un nuevo servicio 5. Invoca al caso de uso "Enviar pendientes" 6. El sistema muestra los datos del servicio elegido (fecha, estado actual), además de la opción de actualizar estado y ver mapa. 7. El taxista elige la opción actualizar estado. 8. El sistema muestra la lista de estados disponibles. 9. El usuario elige el estado deseado. 10. El sistema registra el estado del servicio 11. Invoca al caso de uso "Enviar pendientes". *El taxista repite los pasos 5-8 hasta que se elija un estado de finalización.* 12. El sistema actualiza el servicio como terminado. 13. Invoca al caso de uso "Enviar pendientes". 14. El sistema muestra un mensaje de finalización del servicio y vuelve a la pantalla principal.
Flujos Alternativos	3a El usuario elige un servicio ya iniciado 1. El sistema carga los datos del servicio. 7a El taxista elige la opción de ver en mapa. 1. Invoca al caso de uso "Ver mapa".

Nombre de Caso de Uso: Enviar alerta	
Descripción	Caso que describe el envío de una alerta desde el móvil al servidor dentro de una lista predefinida.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login • Se ha marcado ingreso • Se ha sincronizado datos
Postcondiciones	Alerta enviada
Flujo Básico	1. El taxista elige la opción de enviar alerta en el

	menú principal 2. El sistema muestra una lista de las alertas disponibles, las cuales fueron sincronizadas 3. El taxista elige la alerta a enviar 4. El sistema registra la alerta 5. Invoca al caso de uso “Enviar pendientes” 6. El sistema muestra un mensaje de éxito de la acción y regresa al menú principal
Flujos Alternativos	3a El taxista no encuentra la alerta deseada 1. El taxista regresa al menú principal de la aplicación

Nombre de Caso de Uso: Enviar pendientes	
Descripción	Caso que describe el envío de datos pendientes del móvil al servidor.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login • Se ha marcado ingreso
Postcondiciones	Pendientes enviados
Flujo Básico	1. El taxista presiona el icono de envío de pendientes en la parte superior de la pantalla 2. El sistema valida que haya pendientes 3. El sistema envía los datos pendientes de envío 4. El sistema actualiza el estado de los registros enviados del móvil, según el éxito del envío. 5. El sistema muestra un mensaje de éxito del envío de pendientes.
Flujos Alternativos	1a El sistema detecta que hay datos pendientes de enviar. 1. Continúa en el paso 3 del flujo principal. 2a No hay datos pendientes de enviar 1. El sistema muestra un mensaje indicando que no hay datos pendientes de enviar 3a Hay un error al enviar los pendientes. 1. El sistema deja el estado de los datos como pendiente de envío. 2. El sistema muestra un mensaje de error indicando que el envío de pendientes no fue exitoso. 3b El equipo está fuera de cobertura 1. El sistema muestra un mensaje de fuera de cobertura y pide que se intente enviar los pendientes luego.

Nombre de Caso de Uso: Ver mapa	
Descripción	Caso que describe la vista del mapa en el teléfono
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login • Se ha marcado ingreso
Postcondiciones	
Flujo Básico	1. El taxista elige la opción de ver mapa en el menú principal. 2. El sistema muestra el mapa de la ciudad, centrado en la posición actual del taxista. 3. El sistema muestra mediante un pin la posición actual. 4. El sistema actualiza la posición del taxi en el mapa según va capturando posiciones GPS
Flujos Alternativos	2a Hay un servicio siendo atendido actualmente. 1. Se muestra en el mapa un pin con el servicio actual 2b El taxista pulsa sobre un pin 1. El sistema muestra información adicional sobre el servicio o la posición pulsada.

Nombre de Caso de Uso: Ver mensajes	
Descripción	Caso que describe la vista de los mensajes recibidos por el taxista.
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login • Se ha marcado ingreso
Postcondiciones	
Flujo Básico	1. El taxista elige la opción de ver mensajes en el menú principal. 2. El sistema muestra la lista de mensajes recibidos. 3. El taxista elije un mensaje de la lista. 4. El sistema muestra el detalle del mensaje elegido (título, fecha y contenido). 5. El sistema actualiza el mensaje como leído.
Flujos Alternativos	3a El usuario pulsa el botón de sincronizar. 1. El sistema realiza la sincronización de los mensajes. 2. El sistema muestra un mensaje de éxito.

Nombre de Caso de Uso: Ver tarifario	
Descripción	Caso que describe la consulta del tarifario entre un origen y un destino
Actor Principal	Taxista
Precondiciones	• Se ha realizado el login • Se ha marcado ingreso
Postcondiciones	
Flujo Básico	1. El taxista elige la opción de ver tarifario en el menú principal. 2. El sistema muestra la pantalla de tarifario con la lista de orígenes y destinos disponibles. 3. El taxista elige el origen y el destino. 4. El sistema busca los detalles de tarifa para el origen y destino seleccionados. 5. El sistema muestra la lista de tarifas disponibles.
Flujos Alternativos	5a El sistema no encuentra ninguna tarifa para el origen y destino seleccionados 1. El sistema muestra un mensaje indicando que no hay ninguna tarifa disponible para el origen y destino seleccionados.

3.2.4. Modelo de dominio

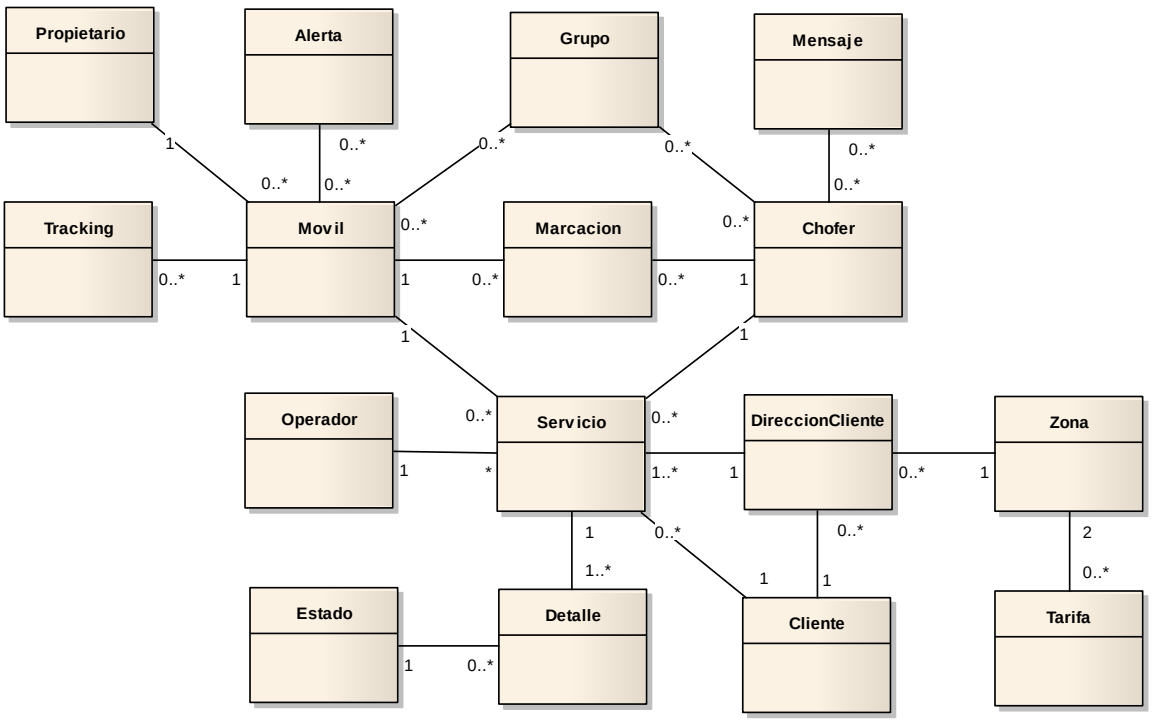


Ilustración 34: Modelo de dominio del caso

3.3. Diseño de la aplicación

Esta es la parte en el caso en la que se aplica la arquitectura propuesta según lo especificado en el capítulo anterior.

3.3.1. Activities y Fragments

Teniendo completa la fase de análisis se procede a identificar los Activities y Fragments:

Activities	
Nombre	**Descripción**
LoginActivity	Interfaz para el logueo a la aplicación
ListaMovilesActivity	Interfaz para elegir la móvil que está usando el taxista
BaseFragmentActivity	Interfaz básica reutilizable que incluye la barra lateral

Fragments	
Nombre	**Descripción**
ListaAlertasFragment	Interfaz con la lista de alertas para elegir
ListaEstadosFragment	Interfaz con la lista de estados aplicables para elegir
ListaMensajesFragment	Interfaz con los mensajes recibidos
ListaServiciosFragment	Interfaz con los servicios recibidos
MenuPrincipalFragment	Interfaz principal con las opciones de menú
ServicioFragment	Interfaz con el detalle del servicio actual
SincronizarFragment	Interfaz para elegir los elementos a sincronizar
TarifaFragment	Interfaz para cálculo de la tarifa
MapaFragment	Interfaz con el mapa y la posición actual
MensajeFragment	Interfaz de detalle del mensaje elegido

3.3.2. Identificar los procesos a ejecutarse

Se identifica y categoriza los procesos que debe realizar la aplicación y que involucran algún procesamiento o conexión con el servidor, según lo especificado en los casos de uso:

Procesos	
Nombre	**Descripción**
login	Ingreso del usuario a la aplicación validando sus credenciales
marcar ingreso	Selección de la móvil y envío de los datos de ingreso
marcar salida	Envío de los datos de salida
enviar alerta	Grabación local y envío de alerta seleccionada
sincronizar	Obtención de los datos desde el servidor
registrar detalle de servicio	Grabación local y envío del detalle de servicio al servidor
confirmar servicio	Envío de confirmación al servidor
crear servicio libre	Grabación local de servicio libre y envío al servidor
enviar tracking	Envío en background de la posición gps de la unidad móvil
enviar pendientes	Envío de pendientes al servidor en background de manera asíncrona

3.3.3. Services y AsynkTask

Tener los procesos identificados nos permite elegir los services y asyntask a utilizar. En este caso se eligió tener dos services:

Elementos	
Nombre	**Descripción**
MChoferBean	Representa la entidad de negocio Chofer
MDetalleServicioBean	Representa la entidad de negocio Servicio
MDetalleTarifaBean	Representa la entidad de negocio Tarifa
MMensajeBean	Representa la entidad de negocio Mensaje

MAlertaBean	Representa la entidad de negocio Alerta
MEstadoBean	Representa la entidad de negocio Estado de Servicio
MMovilBean	Representa la entidad de negocio Móvil
MAlertaEnviadaBean	Representa la entidad de negocio Alerta Enviada
MTarifaBean	Representa la entidad de negocio Tarifa
MZonaBean	Representa la entidad de negocio Zona
MPuntoGPSBean	Representa la entidad de negocio Punto GPS
MServicioBean	Representa la entidad de negocio Servicio
SincronizacionProcessor	Contiene la lógica para sincronizar los datos
PendientesProcessor	Contiene la lógica para envío de pendientes
ServicioProcessor	Contiene la lógica para la gestión de servicios
LoginProcessor	Contiene la lógica para login
AlertaProcessor	Contiene la lógica para el envío de alertas
ZonaDAO	Objeto de acceso a datos para zonas
TarifaDAO	Objeto de acceso a datos para tarifas
AlertaDAO	Objeto de acceso a datos para alertas
EstadoDAO	Objeto de acceso a datos para estados
DetalleServicioDAO	Objeto de acceso a datos para detalles de servicio
DetalleTarifaDAO	Objeto de acceso a datos para detalles de tarifa
AlertaEnviadaDAO	Objeto de acceso a datos para alertas enviadas
ServicioDAO	Objeto de acceso a datos para servicios
MensajeDAO	Objeto de acceso a datos para mensajes
TaxiServiceHelper	Helper que sirve de facade de la aplicación
TaxiApplication	Mantiene el estado global de la aplicación

3.3.4. Beans, Processors, DAOs, Helper y Application

Se definen las demás clases que se utilizarán para cumplir con los requerimientos, según los elementos especificados en la arquitectura:

Services	
Nombre	**Descripción**
TaxiService	Service encargado de los procesos de login, servicio, alertas y mensajes
TaxiBackgroundService	Service encargado de envío de pendientes al servidor
GPSTrackingService	Servicio encargado de envío de tracking gps

3.3.5. Diagramas de secuencia

En los diagramas de secuencia puede apreciarse que los elementos utilizados y la comunicación entre ellos siguen la base de lo presentado en la descripción de la arquitectura propuesta.

Login

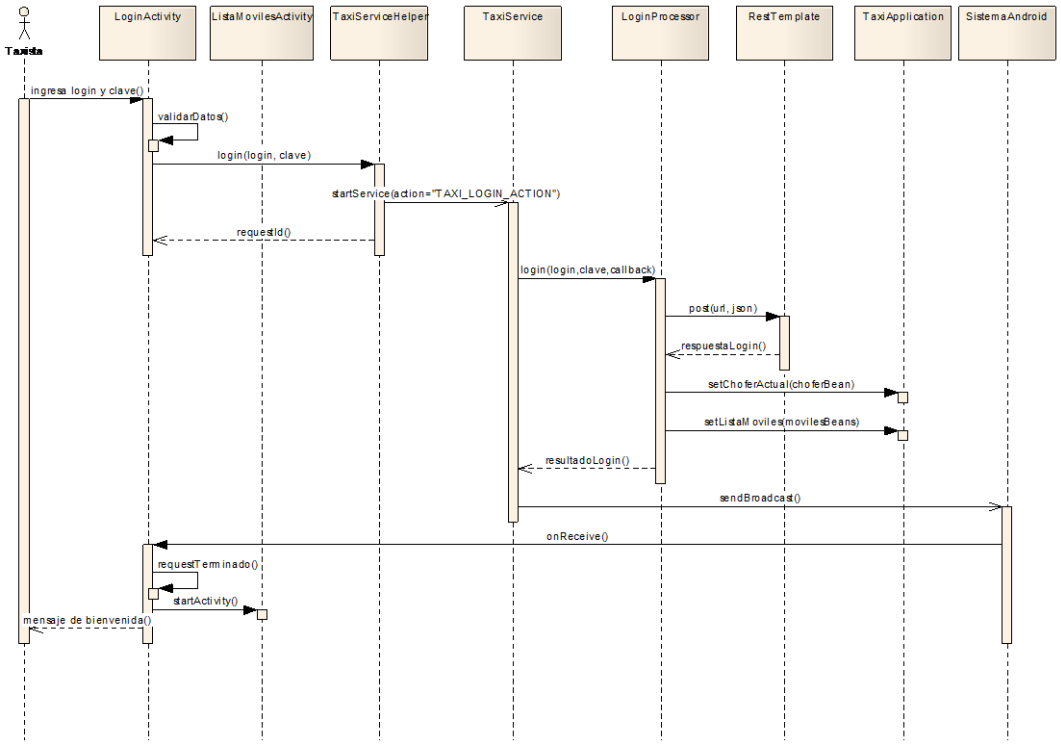


Ilustración 35: Diagrama de secuencia – Login

Marcar ingreso

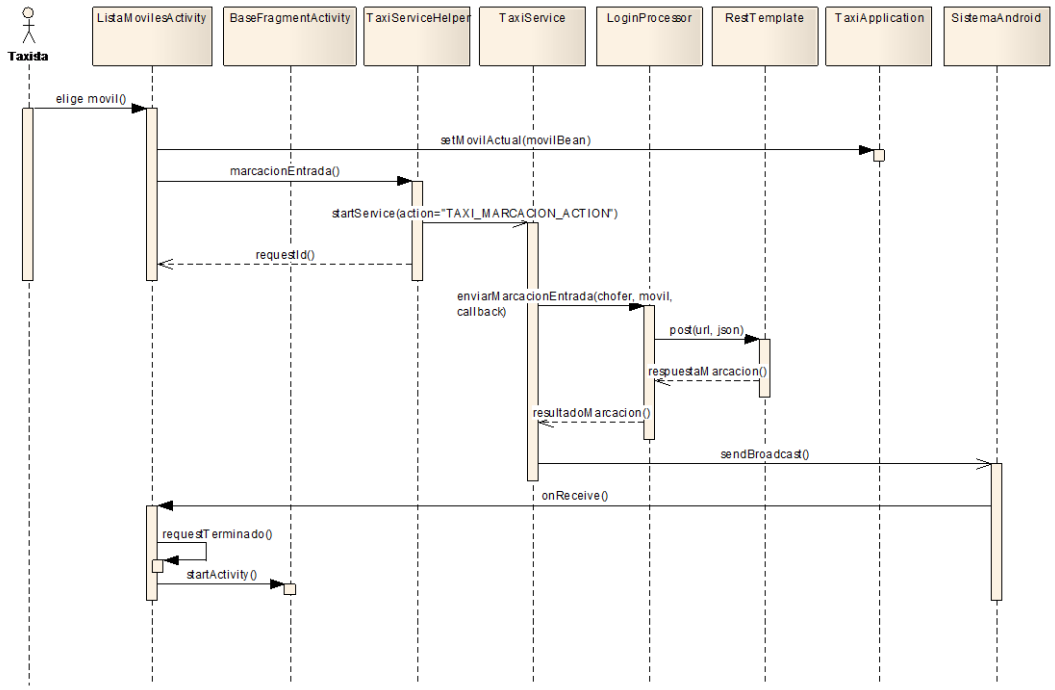


Ilustración 36: Diagrama de secuencia – Marcar ingreso

Marcar salida

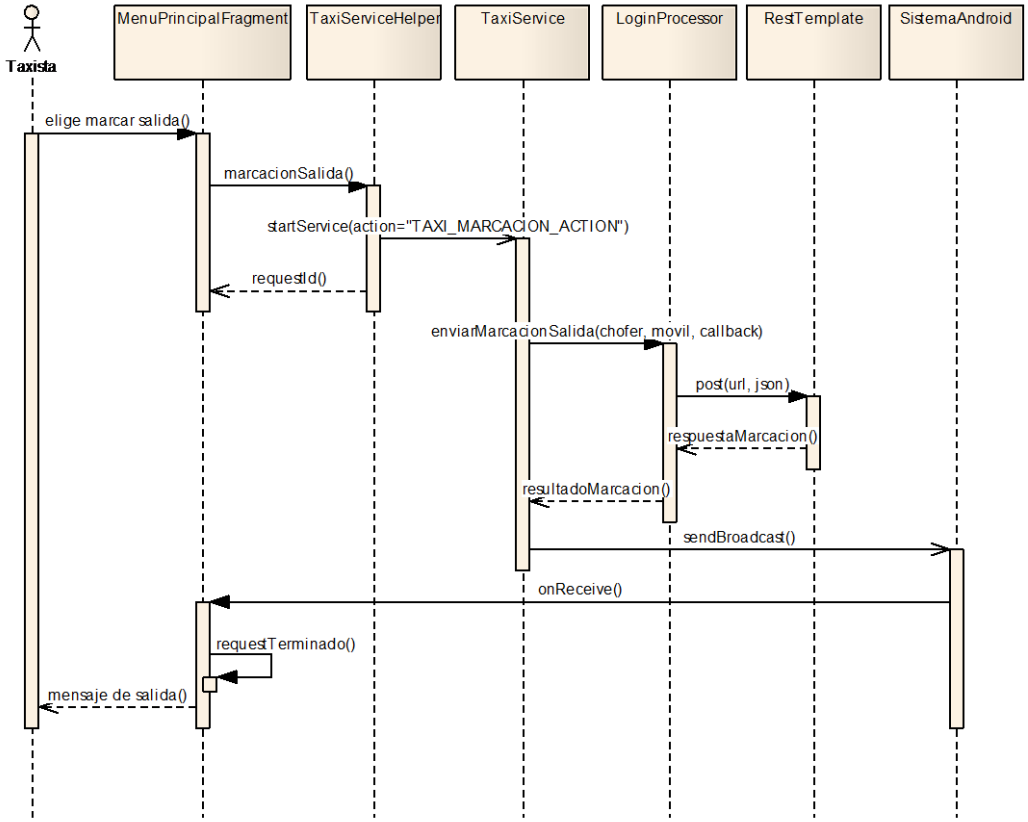


Ilustración 37: Diagrama de secuencia – Marcar salida

Sincronizar

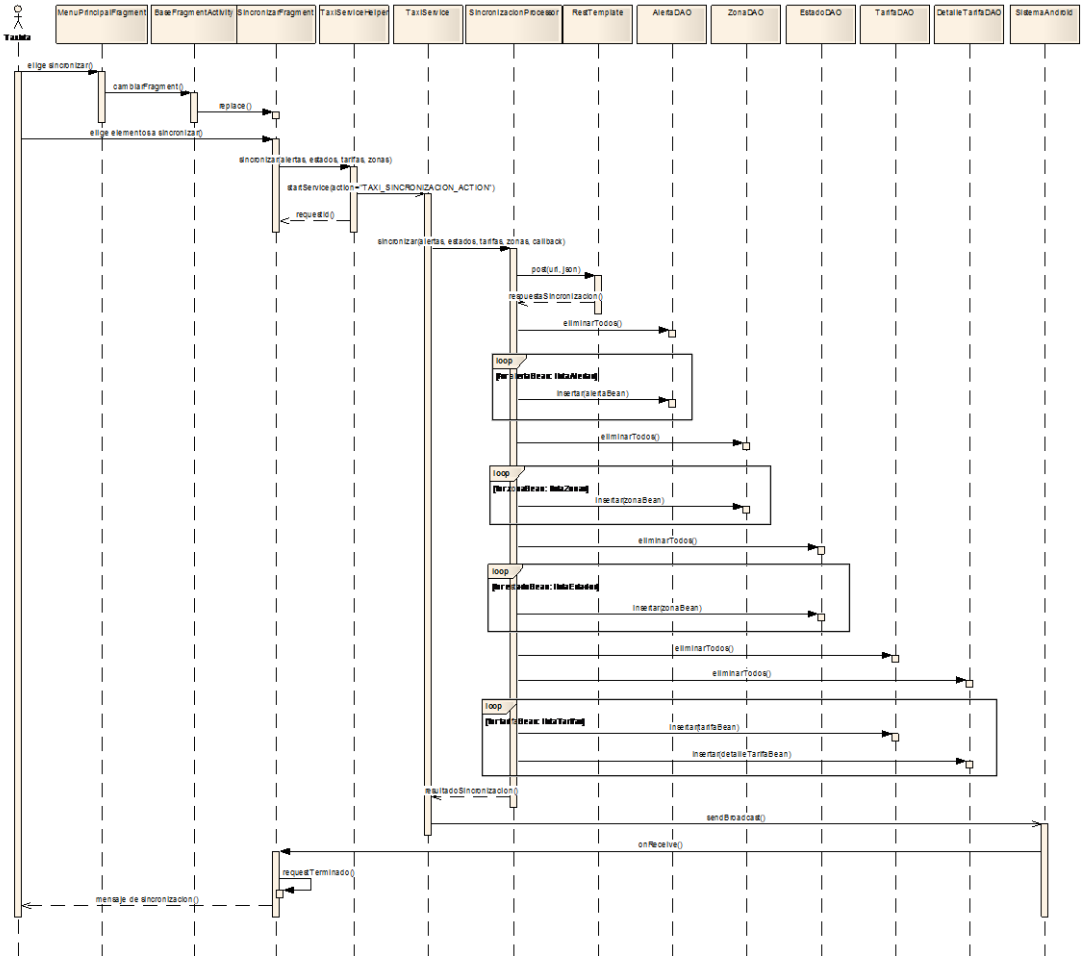


Ilustración 38: Diagrama de secuencia - Sincronizar

Ver mensajes

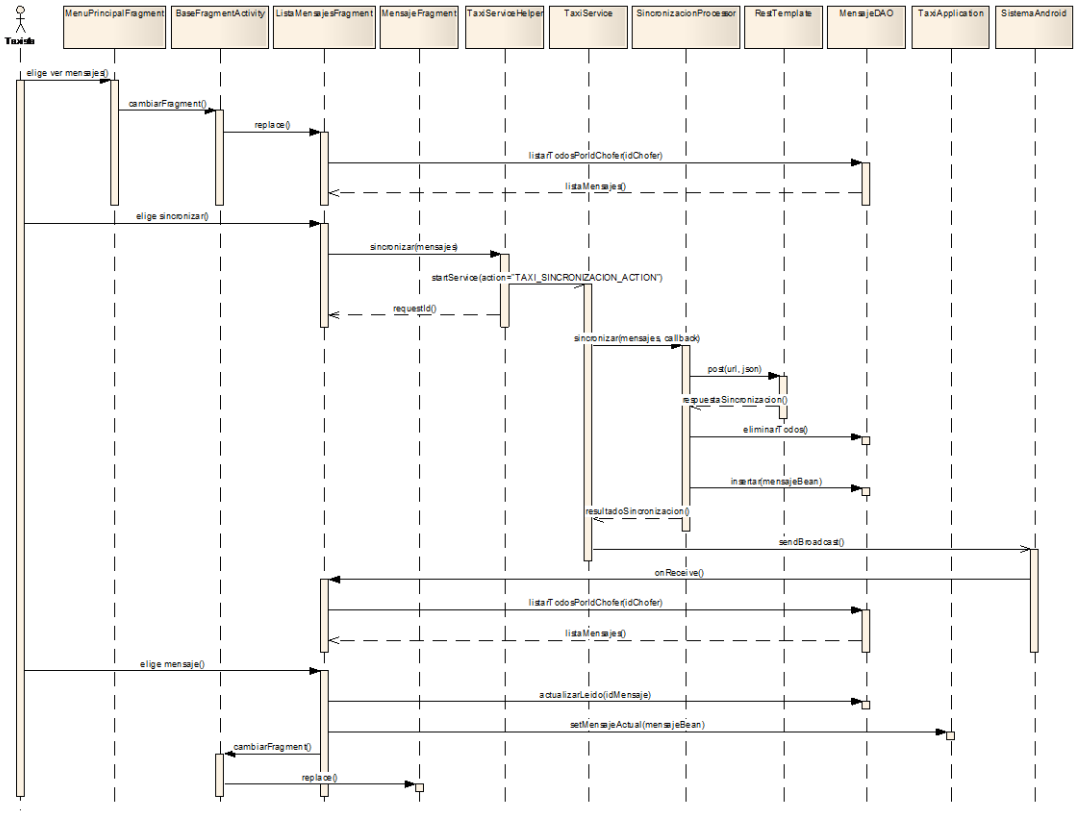


Ilustración 39: Diagrama de secuencia – Ver Mensajes

Ver tarifario

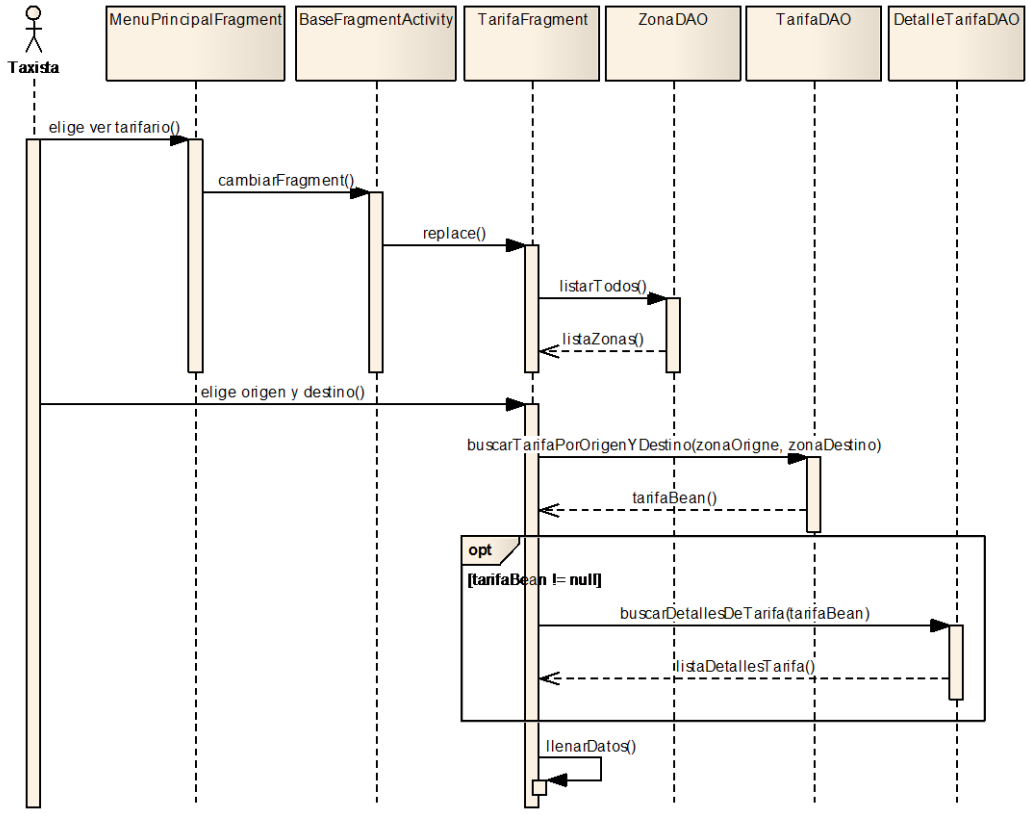


Ilustración 40: Diagrama de secuencia – Ver Tarifario

Ver mapa

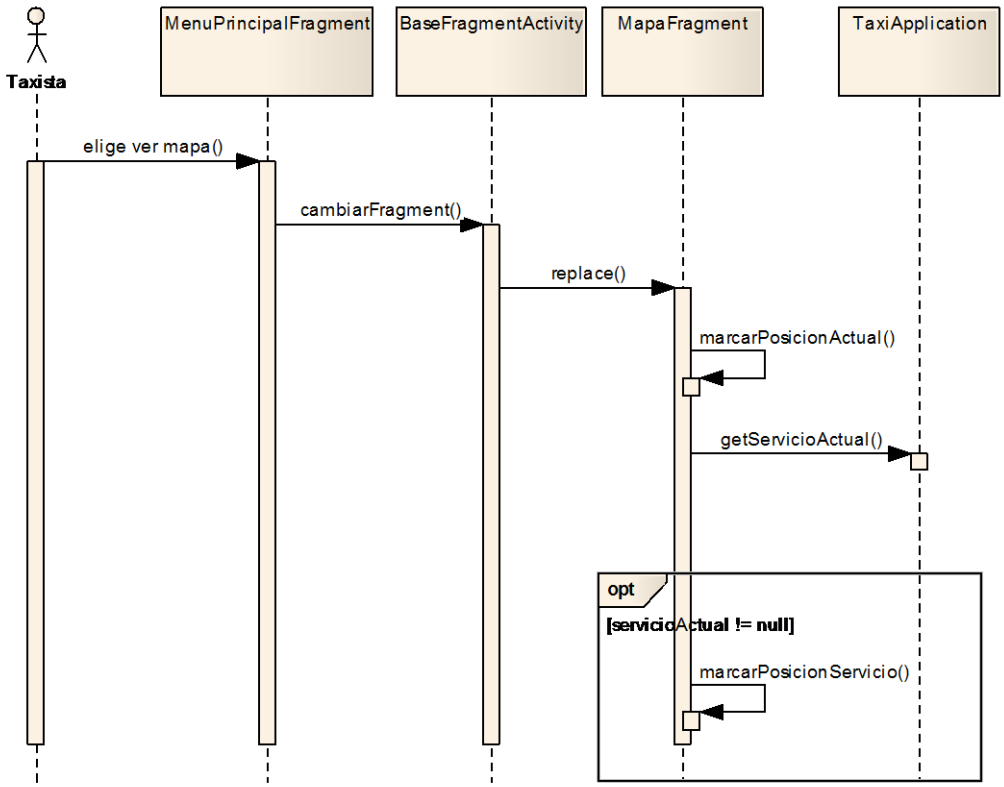


Ilustración 41: Diagrama de secuencia – Ver Mapa

Enviar tracking

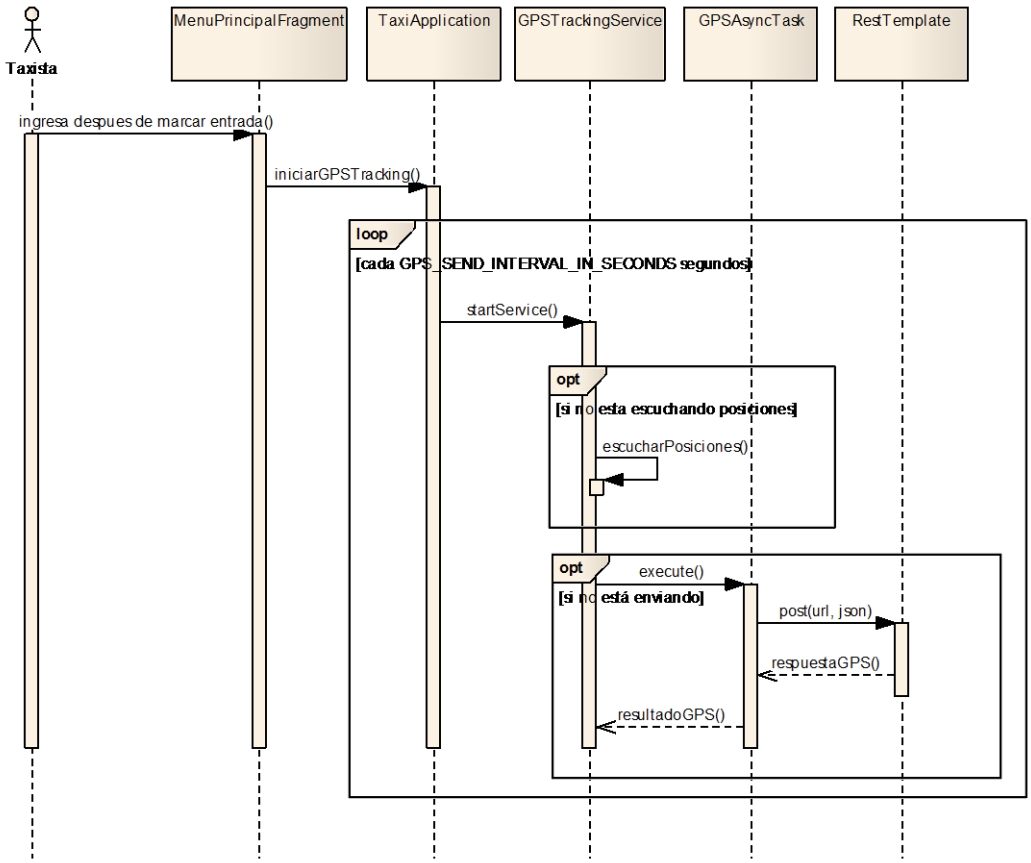
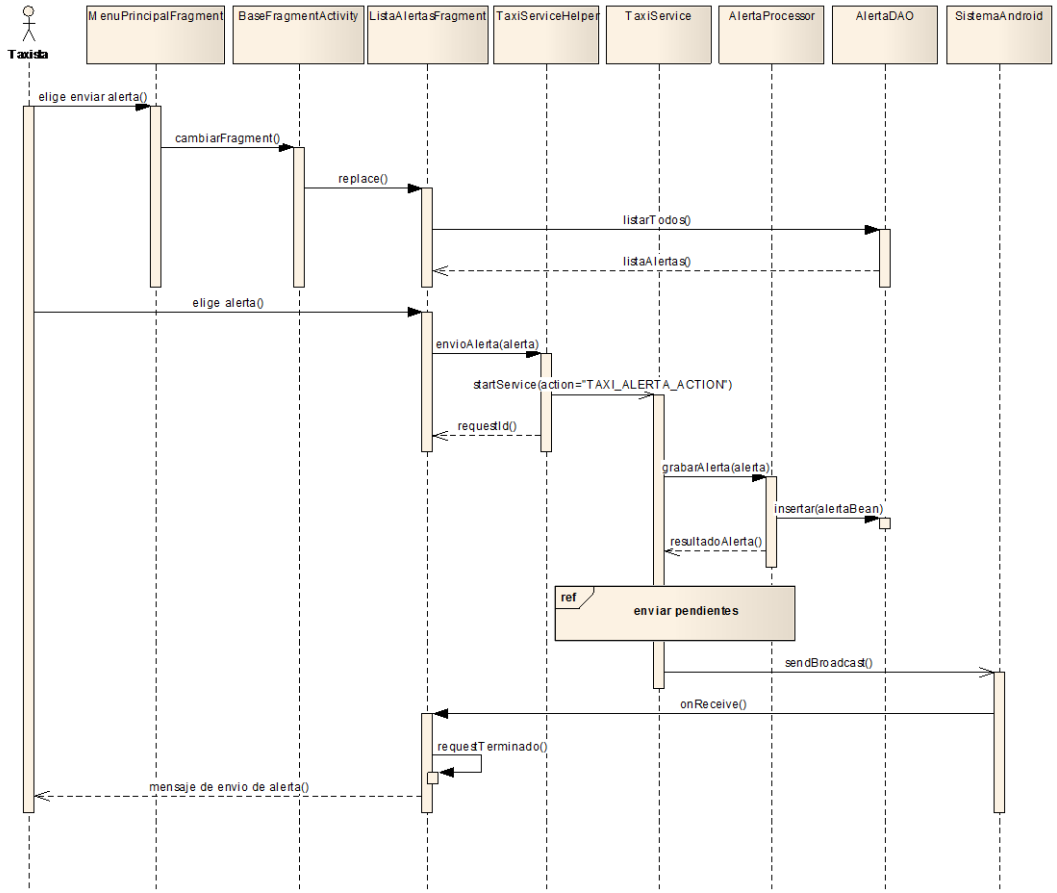
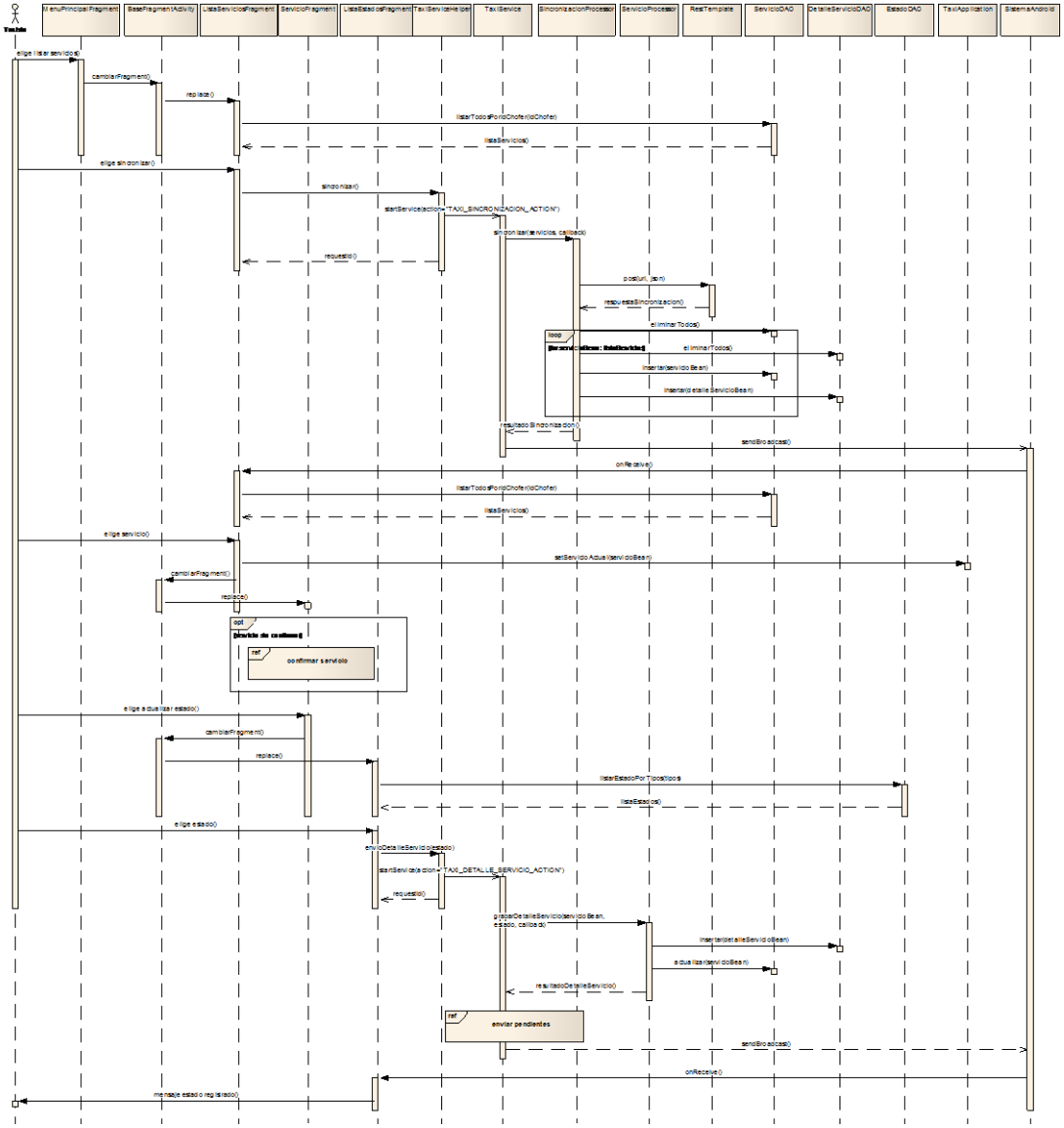


Ilustración 42: Diagrama de secuencia – Enviar tracking

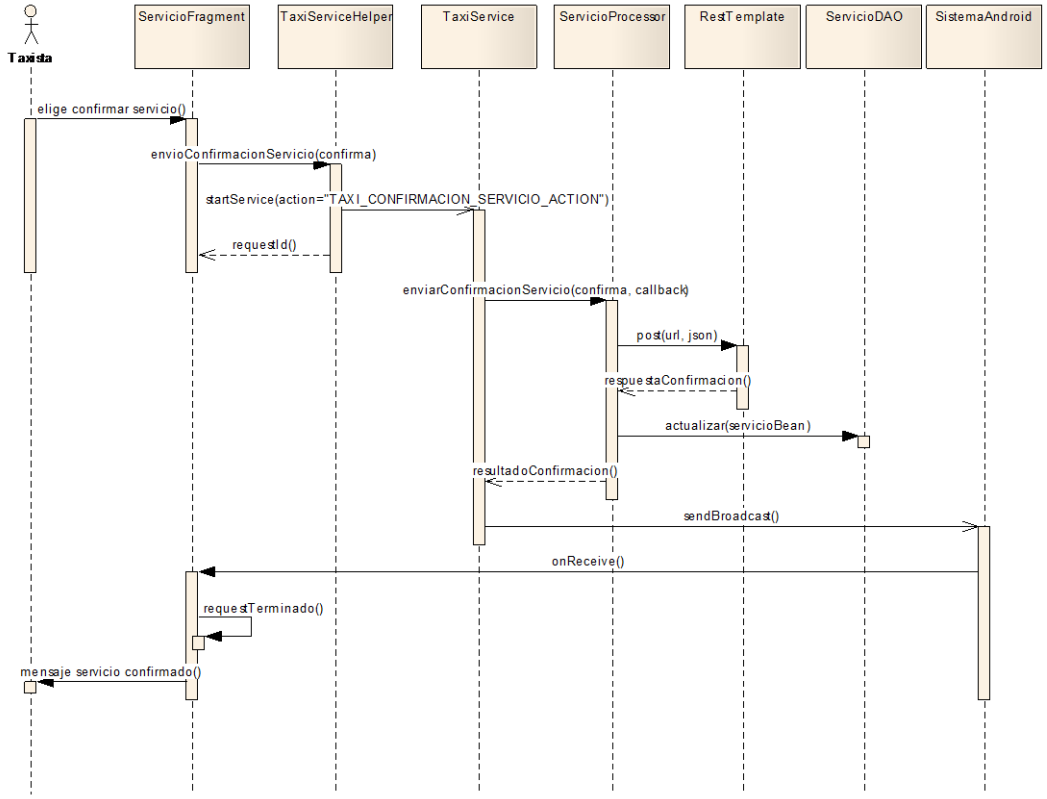
Enviar alerta



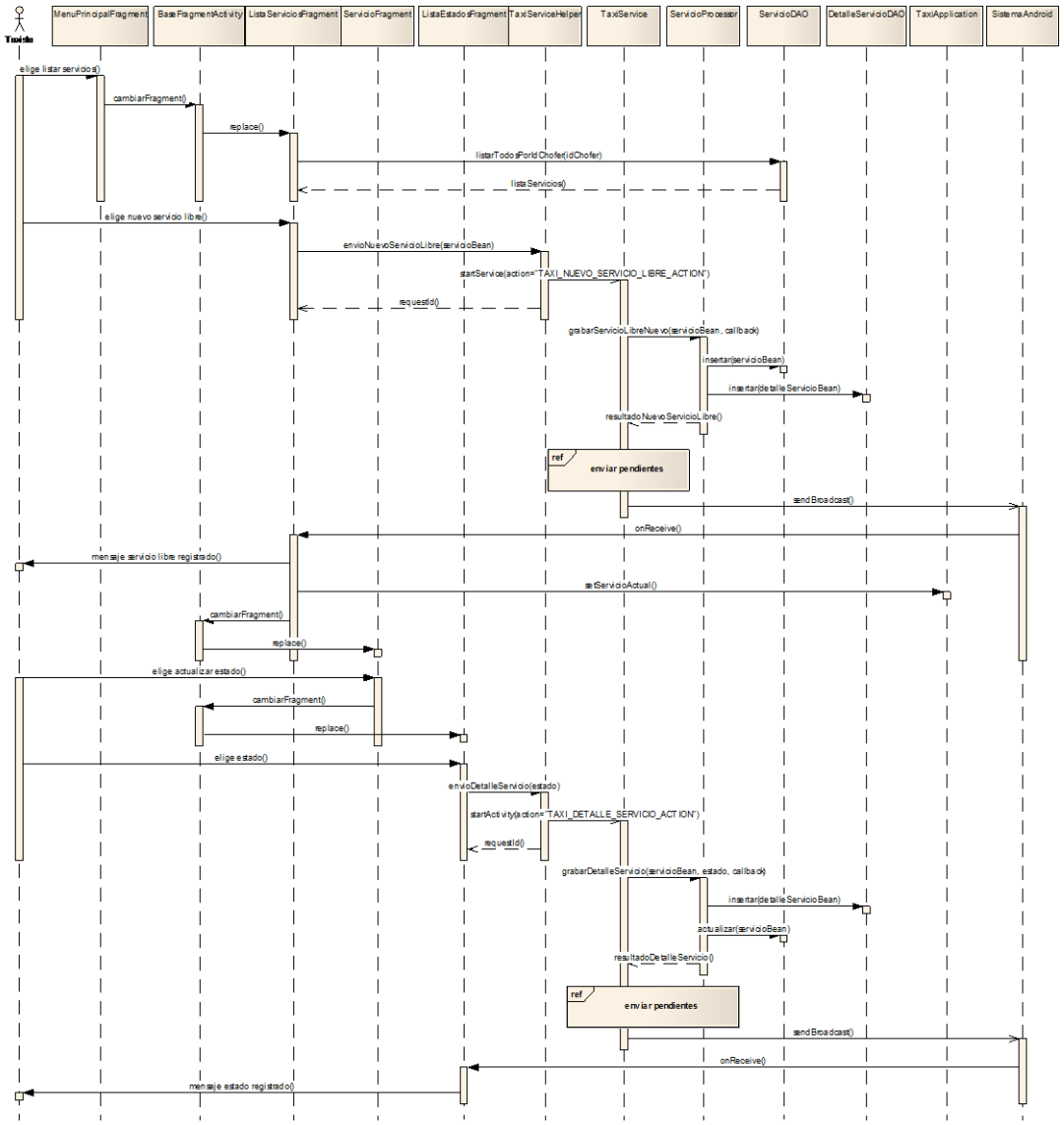
Atender servicio asignado



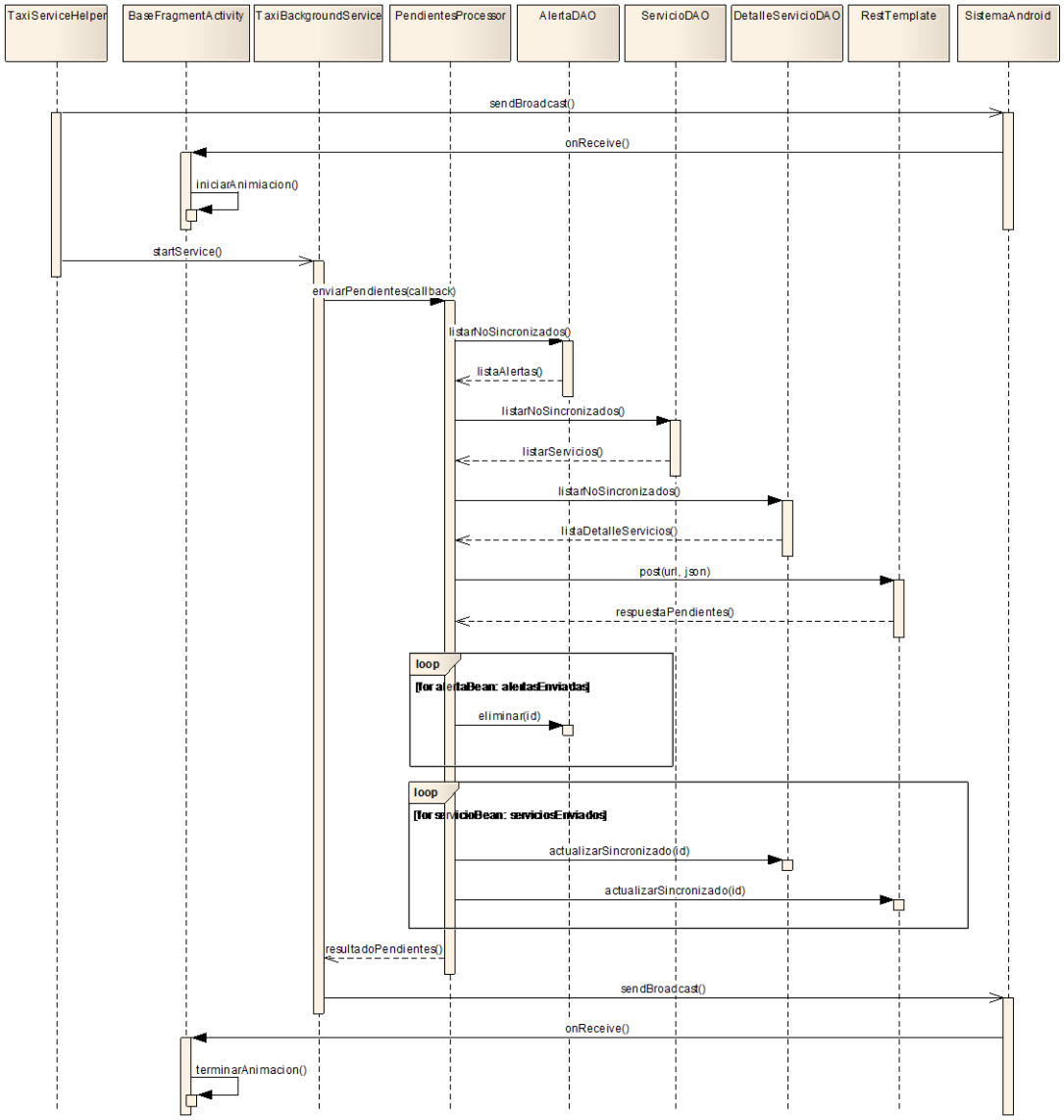
Confirmar servicio



Atender servicio libre



Enviar pendientes



3.3.6. Diagrama de paquetes

En el diagrama de paquetes puede observarse que los paquetes diseñados tienen una relación directa con los elementos propuestos en la arquitectura.

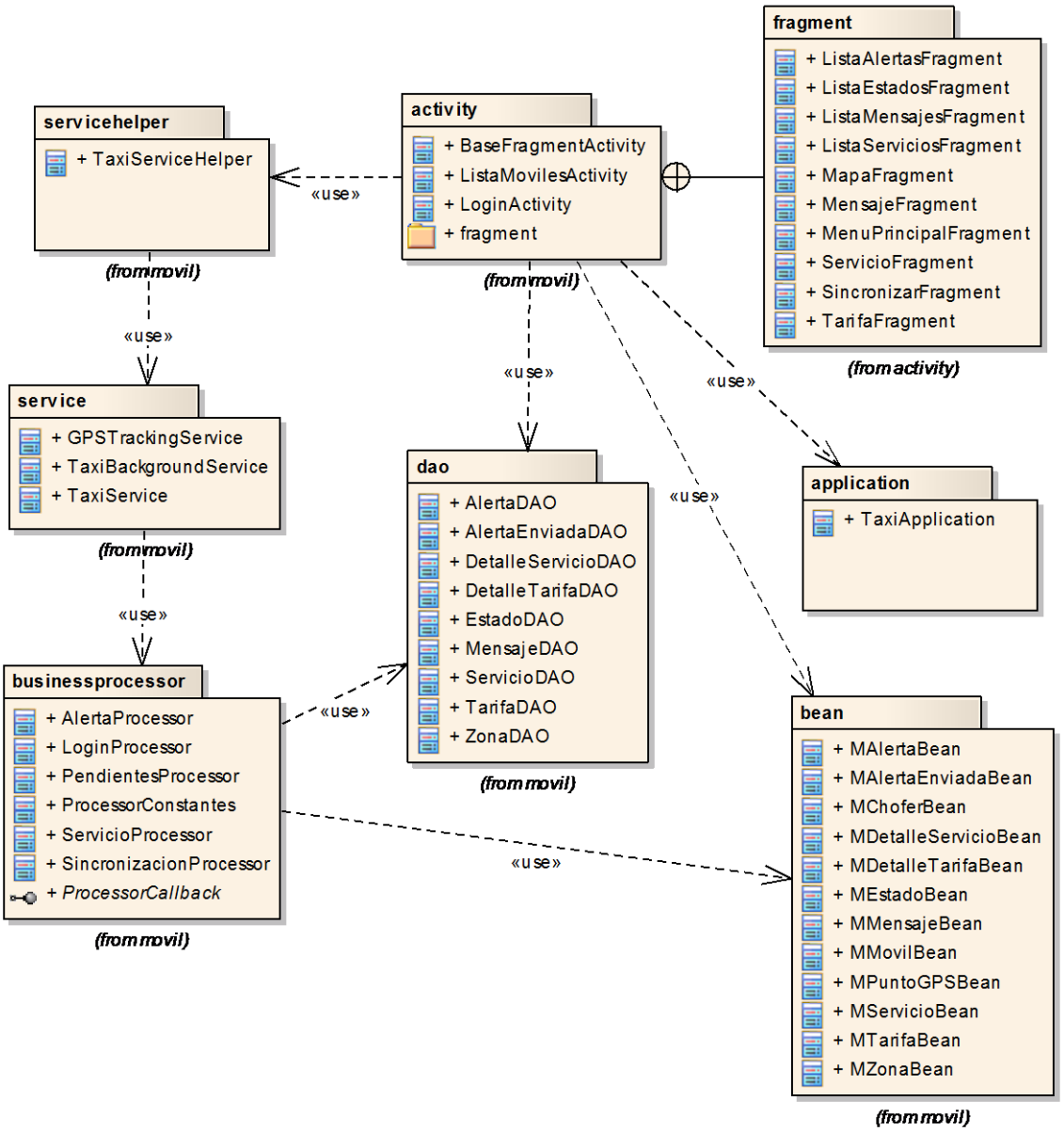


Ilustración 43: Diagrama de paquetes del caso

Paquetes	
Nombre	**Descripción**
activity	Clases asociadas a la interfaz de usuario.
fragment	Clases asociadas a la interfaz de usuario, Fragments que componen otros Activity.
servicehelper	Clases que exponen un facade de funcionalidades a los Activity.
service	Clases encargadas de crear hilos de ejecución en segundo plano.
businessprocessor	Clases que contienen la lógica de negocio.
dao	Clases de acceso a datos.
application	Clases con el estado global de la aplicación.
bean	Clases que encapsulan las entidades de negocio.

3.3.7. Diagrama de clases

El diagrama de clases es presentado separado por paquetes

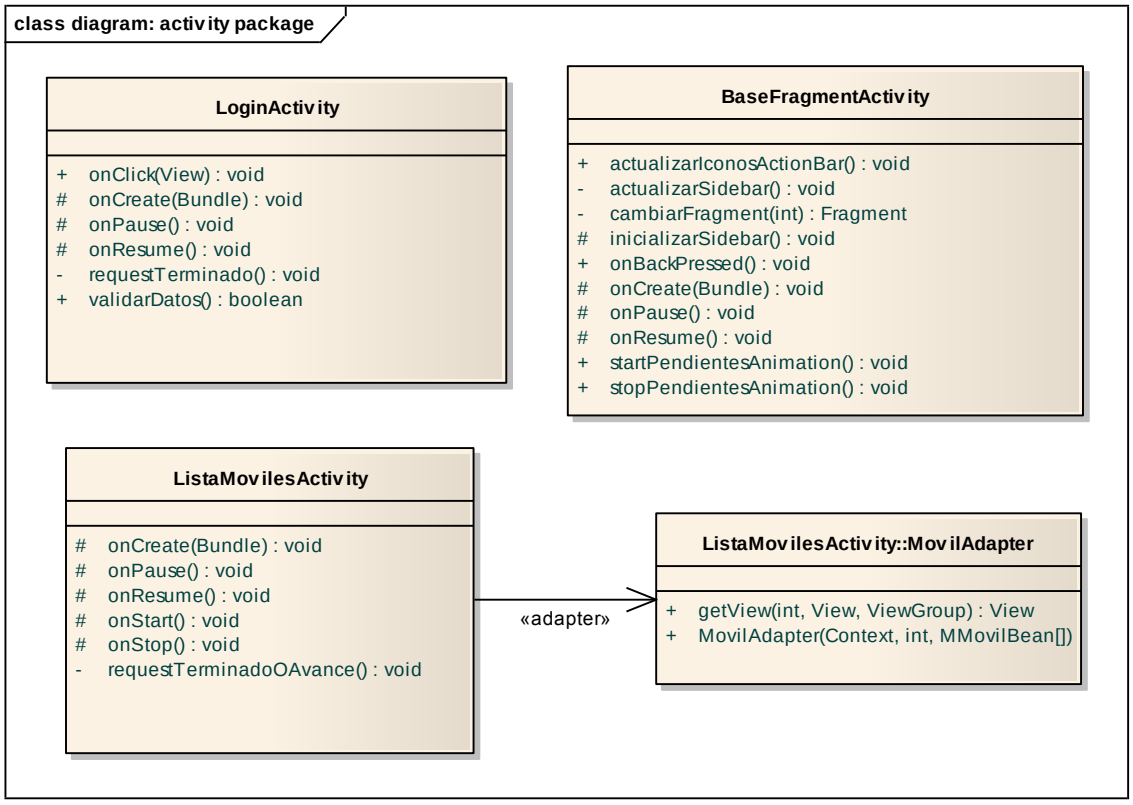


Ilustración 44: Diagrama de clases - Activities

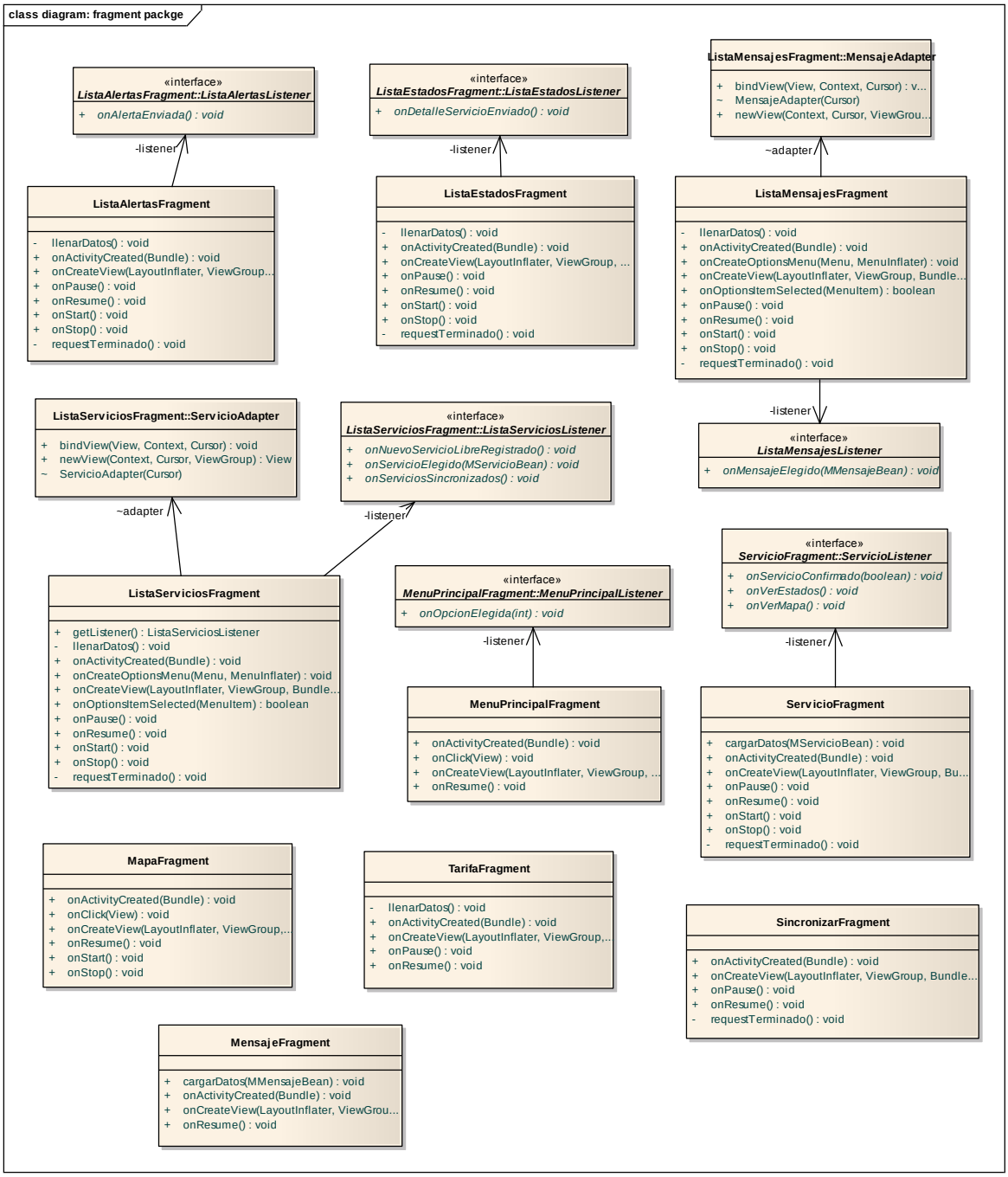


Ilustración 45: Diagrama de clases - Fragments

Ilustración 46: Diagrama de clases - Beans

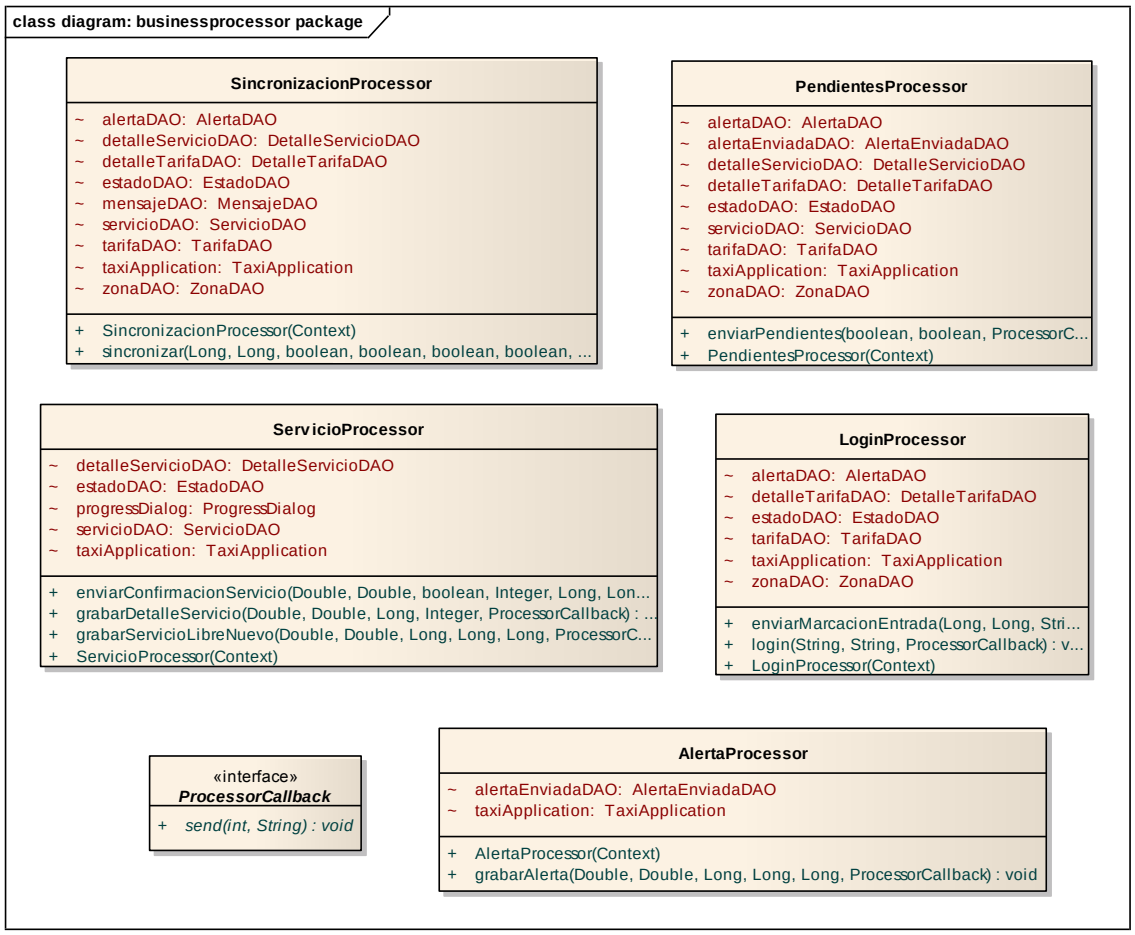


Ilustración 47: Diagrama de clases - Processors

Ilustración 48: Diagrama de clases - DAOs

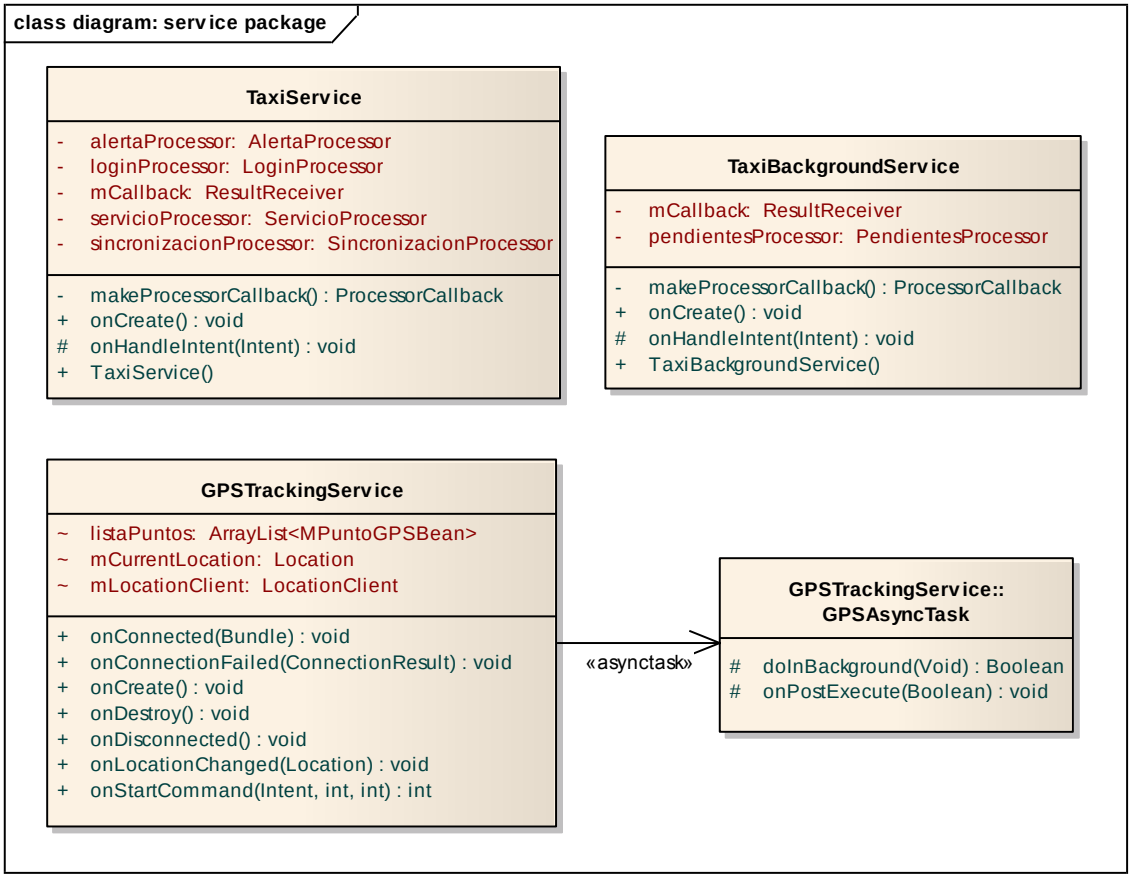


Ilustración 49: Diagrama de clases - Services

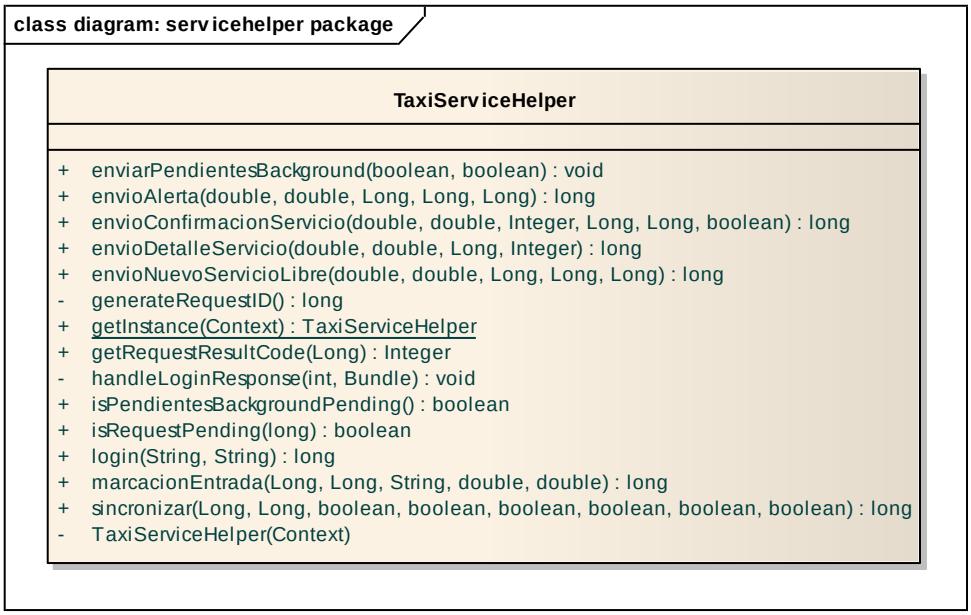


Ilustración 50: Diagrama de clases - Helper

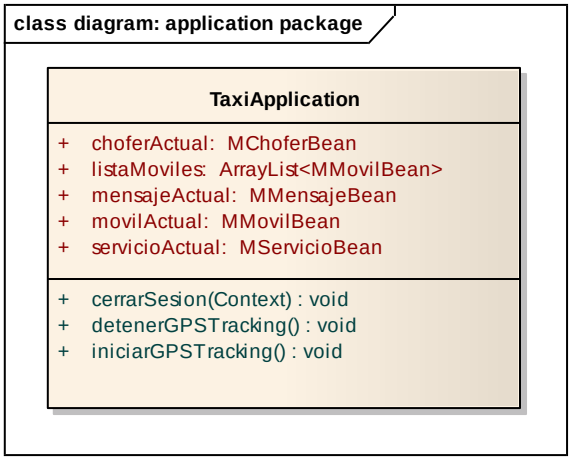


Ilustración 51: Diagrama de clases - Application

3.3.8. Diagrama de despliegue

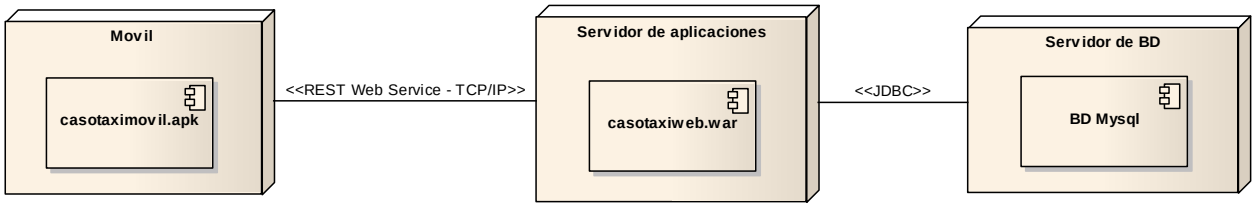


Ilustración 52: Diagrama de despliegue del caso

3.3.9. Herramientas de prueba

Para este proyecto se recomienda utilizar Android Testing Framework de manera conjunta con el DDM para observar y validar el funcionamiento del código que se implementará.

CAPITULO IV: DISCUSIÓN

La presente tesis es una alternativa de arquitectura para el diseño de aplicaciones móviles en Android. Es deseable probar la validez de la hipótesis propuesta, por lo que la culminación del presente tesis radica en la comprobación de los supuestos sobre la base de los cuales se emprendió este trabajo de investigación son aceptables una vez confrontados con la realidad en la cual se pretende aplicar. Esto implica la planificación y ejecución de un conjunto de actividades desarrolladas en este trabajo que tiendan a hacer efectivo este propósito. Para realizar dicha comprobación se utilizarán técnicas de evaluación de una arquitectura de software.

4.1. Definición del Problema

¿Cómo mejorar el diseño de las aplicaciones móviles en Android?

4.2. Hipótesis

Una arquitectura adaptada mejorará el diseño de las aplicaciones móviles en Android.

4.3. Definición de Variables

VI--------X---------VD

- Variable Independiente (VI): Arquitectura adaptada
- Variable Dependiente (VD): Diseño de las aplicaciones móviles en Android

4.4. Indicadores

En la hipótesis se busca demostrar la mejora en el diseño de las aplicaciones móviles en Android, para lo cual se consideran los siguientes indicadores, que son atributos de calidad de software:

- Mantenibilidad (M): Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas
- Fiabilidad (F): Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados
- Eficiencia (E): Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones
- Integridad Conceptual (I): Es el tema subyacente o la visión que unifica el diseño del sistema en todos los niveles

4.5. Contrastación de la Hipótesis

Para efectivamente demostrar el cumplimiento de la hipótesis en el presente trabajo, se tendrá en cuenta como instrumento de demostración de la hipótesis el grado de aceptación de la arquitectura.

Si mediante la utilización de una encuesta aplicada a un grupo de expertos se detecta que el grado de aceptación de la arquitectura propuesta es mayor que el no uso de una arquitectura, se considerará que el uso de la arquitectura propuesta mejorará el diseño de aplicaciones móviles en Android, comprobándose así la hipótesis.

La prueba que se empleará para contrastar las hipótesis se basa en los métodos existentes para la evaluación de una arquitectura de software, las cuales están basadas en el cumplimiento de los atributos de calidad del software. Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces

es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos.

En el presente trabajo se realizará una evaluación basada en el juicio de expertos. Esta evaluación se basa en experiencias previas y conocimiento del dominio de desarrolladores y consultores. Las personas con experiencia pueden decir si la arquitectura de software será lo suficientemente buena y presentará una mejora en comparación con el no uso de la arquitectura.

4.5.1. Diseño de la hipótesis

Se utilizará la prueba de Pre Test y Post Test para poder realizar las comparaciones entre los valores obtenidos en el cuestionario de aceptación usando la documentación estándar oficial realizada por Google sobre la plataforma Android y el resultado luego de brindar la documentación de la arquitectura de diseño propuesta.

Para realizar las pruebas estadísticas se usara Prueba t Student. El test de hipótesis nula por el cual se demuestra que la diferencia entre dos respuestas medidas en las mismas unidades estadísticas es cero. En el trabajo tesis se usará esta prueba para poder comprobar que la hipótesis nula debe ser descartada para poder comprobar la hipótesis de mejora de gestión.
Las pruebas t de muestras dependientes o apareadas, consisten típicamente en una muestra de pares de valores con similares unidades estadísticas, o un grupo de unidades que han sido evaluadas en dos ocasiones diferentes (una prueba t de mediciones repetitivas). Un ejemplo típico de prueba t para mediciones repetitivas sería por ejemplo que los sujetos sean evaluados antes y después de un tratamiento. En el caso de esta tesis el cuestionario será aplicado dos veces a los expertos.

La forma de las hipótesis estadísticas planteadas para pruebas de muestras relacionadas son:

$$H_0: \mu_D = 0$$

$$H_1: \mu_D \neq 0$$

4.5.2. Universo de estudio

Actualmente SUNAT tiene registrados en la ciudad de Trujillo 10 empresas de software, de los cuales 7 realizan desarrollo de aplicaciones móviles, en las cuales se considera puede realizarse la encuesta creada; a partir de estos datos se tomará una muestra significativa para efectos de poder probar la hipótesis.

Para obtener la muestra (n) se utilizó la siguiente fórmula:

$$\mathbf{n} = \frac{N\sigma^2 Z^2}{(N-1)e^2 + \sigma^2 Z^2}$$

Donde:

n = el tamaño de la muestra.

N = tamaño de la población.

σ =Desviación estándar de la población.

Z = Valor obtenido mediante niveles de confianza.

e = Límite aceptable de error muestral.

Hallamos los valores de cada variable de la fórmula:

- Para estimar el valor de "Z": se considera un nivel de confianza de 95%, de que la muestra sea la adecuada para efectos de estudio, ya que ha trabajado con una población no permanente. Tomando el 95% como obtenemos que:

$$\mathbf{Z = 1.96}$$

- Para calcular el valor de "σ", suele utilizarse un valor constante de 0,5:

$$\boldsymbol{\sigma = 0.5}$$

- Para calcular "e", suele utilizarse un valor que varñia entre 1% y 9%, a criterio del investigador se utiliza 5%.

$$\boldsymbol{e = 0.05}$$

Reemplazando los valores obtenidos de las variables en la fórmula, tenemos:

$$\mathbf{n} = \frac{\boldsymbol{N\sigma^2 Z^2}}{\boldsymbol{(N-1)e^2 + \sigma^2 Z^2}}$$

$$\mathbf{n} = \frac{\boldsymbol{7 * 0.5^2 * 1.96^2}}{\boldsymbol{(7-1)0.05^2 + 0.5^2 * 196^2}}$$

$$\mathbf{n = 7}$$

4.5.3. Diseño y selección de la muestra

Según los cálculos anteriores, la muestra estuvo compuesta por 7 expertos, uno por cada empresa de desarrollo de aplicaciones móviles en la ciudad de Trujillo.

4.5.4. Recolección de datos

Para aplicar esta prueba se ha elaborado un cuestionario en el cual se presentan los atributos de calidad en los que tiene impacto la arquitectura formulada, y por cada uno de esos atributo de calidad se tiene items con aplicaciones de esos atributos de calidad dentro del contexto de una aplicación móvil. Cada uno de los ítems tendrá un peso de 1 a 5 según la

medida en que sea cumplido. El puntaje máximo para alcanzar fue de 40 puntos por atributo de calidad, con un total máximo de 160 puntos. Los atributos de calidad seleccionados para ser medidos son:

- Mantenibilidad (M)
- Fiabilidad (F)
- Eficiencia (E)
- Integridad Conceptual (I)

Este cuestionario será utilizado para medir el grado de aceptación en cuanto al cumplimiento de los atributos de calidad con la arquitectura y sin la arquitectura, para lo cual se contará con dos grupos, uno desarrollará el cuestionario conociendo la arquitectura propuesta y el otro conociendo sólo la documentación oficial de Google para su plataforma Android, a cada grupo se le alcanzará la documentación correspondiente. Con los resultados obtenidos empleando el cuestionario se podrá realizar una evaluación comparativa entre ambos enfoques.

De acuerdo al universo y muestra seleccionados en los puntos anteriores se ha tomado como muestra poblacional, para encuestar, a expertos de software que pertenecen a empresas desarrolladoras de software dentro de la ciudad de Trujillo, los resultados a la Encuesta de Grado de Aceptación, por pregunta, se resumen en la tabla siguiente.

Preg.	**Pre Test**			**Post Test**		
	Suma de ptos	**Prom.**	**Calificación**	**Suma de ptos**	**Prom.**	**Calificación**
M1	23	3,29	Regular	24	3,43	Regular
M2	22	3,14	Regular	24	3,43	Regular
M3	23	3,29	Regular	24	3,43	Regular
M4	21	3,00	Regular	22	3,14	Regular
M5	20	2,86	Regular	21	3,00	Regular

M6	17	2,43	Alto	18	2,57	Regular
M7	30	4,29	Fácil	30	4,29	Fácil
M8	22	3,14	Regular	22	3,14	Regular
F1	16	2,29	Poco	17	2,43	Poco
F2	17	2,43	Poco	19	2,71	Regular
F3	15	2,14	Poco	17	2,43	Poco
F4	21	3,00	Regular	21	3,00	Regular
F5	28	4,00	Alta	28	4,00	Alta
F6	22	3,14	Regular	22	3,14	Regular
F7	22	3,14	Regular	24	3,43	Regular
F8	16	2,29	Poco	16	2,29	Poco
E1	15	2,14	Poco	17	2,43	Poco
E2	21	3,00	Regular	23	3,29	Regular
E3	21	3,00	Regular	22	3,14	Regular
E4	25	3,57	Suficiente	25	3,57	Suficiente
E5	21	3,00	Regular	22	3,14	Regular
E6	22	3,14	Regular	23	3,29	Regular
E7	14	2,00	Poco	15	2,14	Poco
E8	21	3,00	Regular	21	3,00	Regular
I1	14	2,00	Poco	22	3,14	Regular
I2	21	3,00	Regular	22	3,14	Regular
I3	19	2,71	Regular	23	3,29	Regular
I4	28	4,00	Suficiente	28	4,00	Suficiente
I5	22	3,14	Regular	23	3,29	Regular
I6	23	3,29	Regular	23	3,29	Regular
I7	13	1,86	Poco	17	2,43	Poco
I8	22	3,14	Regular	23	3,29	Regular

Tabla 1: Resultados del cuestionario de grado de aceptación por pregunta

La siguiente tabla muestra los resultados obtenidos por encuestado:

Individuo	Grupo	Calificación
1	Pre Test	95
2	Pre Test	92
3	Pre Test	87
4	Pre Test	92
5	Pre Test	102
6	Pre Test	96
7	Pre Test	93
1	Post Test	102
2	Post Test	99
3	Post Test	95
4	Post Test	103
5	Post Test	105
6	Post Test	98
7	Post Test	96

Tabla 2: Resultados del cuestionario de grado de aceptación

El cuestionario aplicado puede encontrarse en el Anexo de la presente tesis.

Los puntajes obtenidos por los expertos del Grupo Experimental y Grupo Control, se muestran en tablas estadísticas acompañadas de sus promedios y varianzas para cada atributo de calidad.

Mantenibilidad

Puntajes obtenidos para la aceptación del atributo de calidad Mantenibilidad:

	Pre Test	Post Test
Media (X)	25,43	26,43
Varianza (S^2)	4,62	3,29
Desv. Estand. (S)	2,15	1,81

Tabla 3: Estadística de aceptación de Mantenibilidad

Fiabilidad

Puntajes obtenidos para la aceptación del atributo de calidad Fiabilidad:

	Pre Test	Post Test
Media (X)	22,43	23,43
Varianza (S^2)	6,62	4,29
Desv. Estand. (S)	2,57	2,07

Tabla 4: Estadísticas de aceptación de Fiabilidad

Eficiencia

Puntajes obtenidos para la aceptación del atributo de calidad Eficiencia:

	Pre Test	Post Test
Media (X)	22,86	24,00
Varianza (S^2)	2,48	3,33
Desv. Estand. (S)	1,57	1,83

Tabla 5: Estadística de aceptación de Eficiencia

Integridad Conceptual

Puntajes obtenidos para la aceptación del atributo de calidad Integridad Conceptual:

	Pre Test	Post Test
Media (X)	23,14	25,86
Varianza (S^2)	2,14	1,48
Desv. Estand. (S)	1,46	1,21

Tabla 6: Estadística de aceptación de Integridad Conceptual

Total

Puntajes obtenidos para la aceptación de la arquitectura, sumatoria de los puntajes obtenidos por atributo de calidad:

	Pre Test	Post Test
Media (X)	93,86	99,71
Varianza (S^2)	21,14	13,90
Desv. Estand. (S)	4,60	3,73

Tabla 7: Estadísticas de aceptación de la arqutiectura

4.5.5. Análisis estadístico

La prueba estadística que se aplicó para realizar la contrastación de la Hipótesis corresponde a la prueba de comprobación de 2 medias, observadas en grupos con datos dependientes para universos pequeños (Prueba T Student), para lo cual se utilizó el software estadístico SPSS.

Estadísticas de muestras relacionadas

		Media	N	Desviación estándar	Media de error estándar
Par 1	Post Test	99,71	7	3,729	1,409
	Pre Test	93,86	7	4,598	1,738

Tabla 8: Estadísticas de muestras relacionadas

En esta tabla se muestra los estadísticos descriptivos del pre test y el post test.

Prueba de muestras relacionadas

	Diferencias emparejada					t	gl	Sig. (bilateral)
	Media	Desviación estándar	Media de error estándar	95% de intervalo de confianza de la diferencia				
				Inferior	Superior			
Post Test - Pre Test	5,857	3,288	1,243	2,816	8,898	4,713	6	,003

Tabla 9: Prueba de muestras relacionadas

El valor de alfa fue de 5%, por lo tanto para poder descartar la hipótesis nula se requería que el valor de significancia sea menor que 0.05. Como se ve en la tabla, el valor de significancia (p) es de 0.003. Según lo anterior, p=0.003 y α=0.05, entonces $p < \alpha$. El criterio de decisión nos dice que se rechaza H_0 y se acepta H_1; es decir, se rechaza la hipótesis nula de igualdad de medias, y sí hay diferencia entre las medias y los resultados obtenidos.

4.5.6. Resultados de la Contrastación

Después de realizar la contrastación del criterio de aceptación del uso de la documentación estándar de Google, contra el uso de la arquitectura adaptada propuesta, con los datos obtenidos después de aplicar la encuesta a la muestra de expertos se obtuvieron los siguientes resultados:

- Mantenibilidad (M): Se mejoró la media del criterio de aceptación en el atributo de calidad mantenibilidad de 22,43 a 23,43.
- Fiabilidad (F): Se mejoró la media del criterio de aceptación en el atributo de calidad fiabilidad de 22,43 a 23,43.
- Eficiencia (E): Se mejoró la media del criterio de aceptación en el atributo de calidad Eficiencia de 22,86 a 24,00.
- Integridad Conceptual (i): Se mejoró la media del criterio de aceptación en el atributo de calidad Integridad Conceptual de 23,14 a 25,86.

En conclusión, después de aplicar la prueba T-Student se demuestra que **una arquitectura adaptada mejorará el diseño de las aplicaciones móviles en Android**.

CAPITULO V: CONCLUSIONES

1. Con el uso de la "Arquitectura adaptada para el desarrollo de aplicaciones móviles en Android" propuesta por la presente investigación se puede mejorar el diseño de las aplicaciones móviles en dicha plataforma.
2. La investigación bibliográfica sirvió para conocer a profundidad los diferentes componentes que brinda la plataforma Android para el desarrollo de aplicaciones y las buenas prácticas en el desarrollo, así como la forma de estructurar y documentar una arquitectura de software.
3. Para la documentación de la arquitectura de software se utilizó un enfoque por vistas, cada una de las cuales brinda un aspecto de la arquitectura. Las vistas consideradas adecuadas fueron la vista de módulos y la vista de ejecución, las cuales permitieron describir a detalle la arquitectura, las decisiones significativas acerca de la organización y estructuración del sistema, los componentes a utilizarse y la manera en la que estos se comunican.
4. El uso de UML como notación para la documentación de la arquitectura resulta de gran utilidad, ya que es una notación semiformal ampliamente conocida, considerada de uso general, con elementos y relaciones genéricos aplicable a muchos escenarios y soportada por la gran mayoría de herramientas CASE.
5. Se ha presentado un caso de prueba como aplicación de la arquitectura formulada para complementar la exposición y el entendimiento de la misma. El caso consiste en un sistema web móvil para taxis, enfocándose en el diseño de los 13 casos de uso propios del módulo móvil de dicho sistema.
6. A través del cuestionario aplicado a 7 expertos en el desarrollo de aplicaciones móviles en Android se pudo comprobar la validez de la hipótesis en base a la mejora de los atributos de calidad de software: mantenibilidad, fiabilidad, eficiencia e integridad conceptual.

CAPITULO VI: RECOMENDACIONES

1. Para la documentación de una arquitectura de software se cuenta con un gran número de modelos y vistas; no todas aplican para todos los tipos de proyectos de software por lo que se debe investigar y seleccionar las vistas que mejor contribuyan a la documentación de la arquitectura que se está desarrollando.
2. Parte importante al momento de documentar la arquitectura de software es incluir la fundamentación de las decisiones arquitectónicas tomadas, ya que esto permite la revisión de la arquitectura, mejora el entendimiento y facilita la posible extensión de la arquitectura.
3. La plataforma Android presenta varios bloques de construcción básicos para implementar una aplicación, es importante conocer las ventajas y desventajas que cada uno de éstos presenta para poder construir un producto de calidad.
4. Al momento de documentar la arquitectura se debe considerar hacia quién esta dirigida dicha documentación, diferenciando los aspectos técnicos de los no técnicos para mejorar el entendimiento.
5. Al realizar el diseño de una aplicación móvil se debe tratar de producir un diseño detallado y lo más realista posible para maximizar la trazabilidad que habrá con el código que se escriba, de esta forma se logra que el programador entienda y aplique todas las decisiones arquitectónicas que se hayan tomado.
6. Considerar aspectos referidos a las prestaciones de los teléfonos móviles donde va a ser ejecutada la aplicación al momento de realizar el diseño, de manera que se respeten las limitaciones que pueda haber de batería, red, capacidad de cómputo, etc.
7. La arquitectura presentada puede servir de base para nuevos proyectos, donde se busque refinarla, extenderla o especializarla para su uso en un producto específico o una línea de productos.
8. Se recomienda la utilización de la herramienta Enterprise Architect para las tareas de modelamiento ya que cuenta con un amplio número de diagramas y artefactos, además de ser muy fácil de usar.

CAPITULO VII: REFERENCIAS BIBLIOGRÁFICAS

Ableson, F., Sen, R., & King, C. (2011). Android in Action. Stamford: Manning Publications.

Ambler, S. (12 de Julio de 2008). The Agile Unified Process (AUP). Recuperado el 15 de Marzo de 2012, de The Agile Unified Process (AUP): http://www.ambysoft.com/unifiedprocess/agileUP.html

Bass, L. (1997). Recommended Best Industrial Practice for Software Architecture Evaluation. Pittsburgh: Georgia Institute of Technology.

Bass, L., Clements, P., & Kazman, R. (2002). Software Architecture in Practice. Nueva York: Addison-Wesley.

Bosch, J. (2000). Design and Use of Software Architectures. Nueva York: Addison-Wesley.

Burnette, E. (2010). Hello, Android: Introducing Google's Mobile Development Platform. Los Angeles, Estados Unidos: Pragmatic Bookshelf.

Conder, S., & Lauren, D. (2011). Android Wireless Application Development. Michigan: Addison-Wesley.

de San Martín Oliva, C. R. (s.f.). Metodología Iconix. Recuperado el 5 de 1 de 2013, de http://www.portalhuarpe.com.ar/Seminario09/archivos/MetodologiaICONIX.pdf

Dobjanschi, V. (20 de Mayo de 2010). Google IO 2010. Recuperado el 2 de Julio de 2014, de https://dl.google.com/googleio/2010/android-developing-RESTful-android-apps.pdf

Felker, D. (2011). Android Application Development for Dummies. Indiana: Wiley Publishing.

Gargenta, M. (2011). Learning Android. Nueva York: O'Reilly.

Jordan, L., & Greyling, P. (2011). Practical Android Projects. New York: Apress.

López-Hermoso Agius, J. J., Montero Navarro, A., Martín.Romo Romero, S., de Pablos Heredero, C., Izquierdo Loyola, V. M., & Nájera Sánchez, J. J. (2000). Informática Aplicada a la Gestión de Empresas. Madrid, España: ESIC Editorial.

Martensson, F. (2006). Software architecture quality evaluation. Karlskrona, Suecia: Blekinge Institute of Technology.

Mattsson, M. (2008). Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability. Ronneby: Blekinge Institute of Technology.

Nicolas, S. (22 de Junio de 2014). Robospice Wiki. Recuperado el 8 de Julio de 2014, de https://github.com/stephanenicolas/robospice/wiki

Nicolas, S. (2014 de Junio de 22). Robospice Wiki. Recuperado el 8 de Julio de 2014, de https://github.com/stephanenicolas/robospice/wiki

Opcion Consultores. (5 de Octubre de 2012). Opcion Consultores. Recuperado el 20 de Septiembre de 2012, de http://opcion.com.uy/: http://camadiro.weebly.com/uploads/1/3/0/5/13054866/s.i._de_recursos_humanos.pdf

Oz, E. (2006). Administración de los Sistemas de Información (5a ed.). México D.F., México: Cengage Learning.

Pressman, R. (2002). Ingeniería de Software, Un enfoque práctico (5ta ed.). Madrid: McGrall-Hill.

Rosenberg, D. (2009). Use case driven object modeling with UML: a practical approach. Nueva York, Estados Unidos: Addison Wesley.

Rosenberg, D., Stephens, M., & Collins-Cope, M. (2005). Agile Development with ICONIX Process: people, process and pragmatism. Nueva York, Estados Unidos: Apress.

Sommerville, I. (2005). Ingeniería del Software (7ma ed.). Madrid: Pearson Addison Wesley.

CAPITULO VIII: ANEXOS

8.1. Cuestionario para conocer el Grado de Aceptación de la Arquitectura

Objetivo: Recolectar información sobre el grado de aceptación del diseño de una aplicación móvil en Android

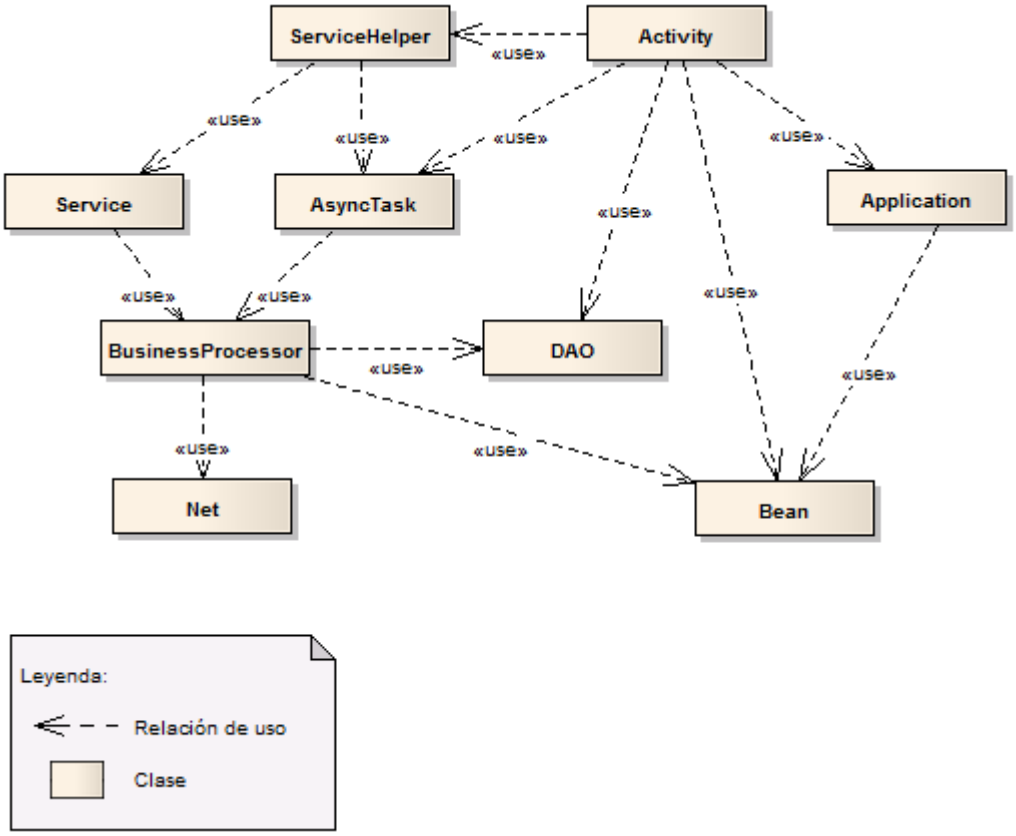
Indicaciones: Con el fin de conocer el grado de aceptación de una arquitectura móvil en Android solicitamos su colaboración para el llenado de la siguiente encuesta marcando la respuesta de su elección. El cuestionario está conformado por 4 grupos de preguntas que corresponden a 4 atributos de calidad que se espera tenga una arquitectura de software, estos atributos conforman los indicadores del grado de aceptación de la arquitectura.

Datos Generales:

Nombre : _____
Cargo :_____
Empresa :_____

Resumen de la arquitectura

La arquitectura propuesta puede resumirse en estructurar los proyectos Android en los módulos mostrados en la figura siguiente:



Activity: Los Activity representan las vistas dentro de la aplicación móvil. Es a través de éstos que se interactúa con el usuario, mostrándole información y capturando sus datos de entrada.

ServiceHelper: El ServiceHelper funciona como un facade para todos los servicios brindados a los Activity, canalizando todas las invocaciones, viene a ser un API o la interfaz a utilizarse. Sus métodos son asíncronos de manera que no bloquean el flujo de los Activity al invocarlos. La arquitectura propuesta recomienda que toda aplicación cuente con un ServiceHelper. La interfaz que presenta este elemento para llegar a los Service es sencilla, simplificando la forma en la que los Activitys la usan y encargándose de de hacer las invocaciones más complejas.

Service: Los Service son los elementos que aprovechan el potencial de los componentes Service de Android, permitiendo que los procesos sean ejecutados en background y con una prioridad superior a la que se logra por otros medios, con esto aseguramos que el sistema operativo no elimine el proceso por falta de recursos, lo que aumenta la robustez de la aplicación.

AsyncTask: Los AsyncTask son una alternativa a los Service para ejecutar procesos en background aunque no comparten su nivel de prioridad. Estos componentes suelen estar muy ligados a los Activity ya que cuentan con métodos que facilitan la tarea de actualización de la interfaz de usuario, la cual se complica por el hecho que un AsyncTask se ejecute en un hilo independiente del hilo principal. Como en el caso de los Service, su principal tarea es crear el hilo independiente (lo cual hace implícitamente el AsyncTask) para evitar que tareas largas bloqueen la aplicación, impidiendo al usuario interactuar con la interfaz, pero la tarea en sí es delegada al BusinessProcessor. El uso de AsyncTask es más sencillo que el de los Service, por eso puede decidirse usar esta opción pero hay que considerar que como se mencionó no tiene su nivel de prioridad.

BusinessProcessor: Los BusinessProcessor son los elementos que contienen la lógica de negocio de la aplicación, alejándose de temas de componentes propios de la plataforma Android para concentrarse en las tareas propias de los requerimientos de la aplicación. Estos elementos pueden ser traídos fácilmente de la capa de negocio de otras plataformas como la web sin tener que realizar muchas modificaciones para adaptarlas a la plataforma móvil.

Application: El elemento Application se encarga de mantener el estado global actual de la aplicación, en este se puede agregar variables como por ejemplo el usuario actual, el último producto elegido, el cliente actual, etc. según los requerimientos específicos de la aplicación a desarrollarse. Trabajar de esta manera nos permite contar con algo similar a una sesión, como se trabaja en las aplicaciones web. Mantener aquí variables como las mencionadas facilita compartir información entre los componentes de la aplicación, evitar información repetida y simplifica las invocaciones entre componentes.

DAO: El elemento DAO es el encargado de la persistencia de los datos utilizados en la aplicación en alguno de los medios brindados por Android. Como su nombre lo indica este elemento sigue el patrón de diseño DAO de acceso a datos, lo que significa que esta clase es la encargada de manejar la conexión al medio de almacenamiento utilizado y la modificación de los datos.

Bean: Los Beans son los encargados de representar las entidades de negocio según el proyecto, ellas encapsulan los datos y facilitan su manejo e intercambio entre los diferentes componentes. Estos Beans suelen ser replicas de los Beans que se tiene en otras plataformas como en el backend web, en estos casos no siempre es necesario que se manejen todos los atributos en el móvil sino un subconjunto de ellos, puesto que las funcionalidades de la aplicación web suele ser más limitada, así se evita aumentar el volumen de datos manejados innecesariamente.

Net: Este elemento es el encargado de realizar las comunicaciones de la aplicación a través de la red. Normalmente esto significa enviar y recibir datos hacia y desde un servidor en la web. Esta es una tarea que requiere ciertas configuraciones según el mecanismo a utilizar por lo cual esta clase encapsula un trabajo que probablemente tendría que ser replicado en varios puntos. Además debe encargarse de manejar los múltiples escenarios que pueden presentarse al querer hacer peticiones a través de la red, permitiendo que la aplicación siga su funcionamiento normal aunque dichas peticiones no sean exitosas, por ejemplo si se encuentra en una zona de no cobertura.

Para acceder a la documentación completa de la arquitectura adaptada propuesta, por favor ingrese al siguiente enlace: goo.gl/L3prdE

Preguntas:

Marque correctamente según la medida en la que la arquitectura presentada favorece el cumplimiento de los atributos siguientes:

MANTENIBILIDAD (Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas)

M1 - ¿Qué nivel de modularidad tienen las aplicaciones (elementos con alta cohesión y bajo acoplamiento)?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

M2- ¿Qué tan claro es el uso que se le da a cada componente de la plataforma Android, con sus ventajas y desventajas?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

M3 - ¿Los componentes que contienen la lógica de negocio de la aplicación pueden ser reutilizados en otras aplicaciones?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

M4 - ¿Los componentes pueden ser reutilizados dentro de la misma aplicación?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

M5 - ¿Con qué nivel de facilidad que se puede evaluar el impacto de un determinado cambio sobre el resto del software?

a) Muy difícil b) Difícil c) Regular d) Fácil e) Muy fácil

M6 - ¿Cuál es el nivel de conocimiento requerido de la plataforma Android para realizar modificaciones?

a) Muy alto b) Alto c) Regular d) Bajo e) Muy bajo

M7 - ¿Con qué nivel de facilidad se pueden realizar pruebas para un componente?

a) Muy difícil b) Difícil c) Regular d) Fácil e) Muy fácil

M8 - ¿Cuál es el nivel de analizabilidad para identificar causas de fallos en el software?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

FIABILIDAD (Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados)

F1 - ¿Se considera el funcionamiento de la aplicación cuando se encuentra fuera de cobertura?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

F2 - ¿Qué tanto se considera el reenvío de información al servidor cuando la comunicación inicial falla?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

F3 - ¿Se mantiene en cache el resultado de operaciones realizadas para cuando el recurso no esté disponible?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

F4 - ¿Se considera la priorización de hilos para evitar eliminación de componentes por el SO por falta de memoria?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

F5 - ¿Cuál es el nivel de transparencia para el usuario de tareas largas, sobre todo en conexiones al servidor?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

F6 - ¿Qué nivel de robustez de la aplicación se considera ante eventos como giros de pantalla o destrucción de Activities?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

F7 - ¿Qué nivel de desacoplamiento hay entre Activities y los hilos que ejecutan los procesos largos?

a) Muy baja b) Baja c) Regular d) Alta e) Muy alta

F8 - ¿Se considera el mantenimiento del estado global de la aplicación durante su ciclo de vida?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

EFICIENCIA (Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones)

E1 - ¿Se considera llevar un control del número de hilos ejecutándose por la aplicación?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E2 - ¿Se considera el consumo de recursos del teléfono al realizar envío de datos por la red?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E3 - ¿En qué nivel los procesos largos permiten ser ordenados para realizarse en secuencia o en paralelo según la naturaleza de la aplicación?

a) Muy difícil b) Difícil c) Regular d) Fácil e) Muy fácil

E4 - ¿Permite que procesos largos no interfieren con el funcionamiento de la aplicación?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E5 - ¿Se Favorece la separación de tareas en hilos separados para mejorar el performance?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E6 - ¿Se considera claramente la distinción entre las diferentes formas en las que la plataforma Android permite el manejo de hilos?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E7 - ¿Se considera la reducción de la cantidad de recursos utilizados por cada Activity?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

E8 - ¿En qué grado se presenta feedback inmediato ante tareas complejas de larga duración?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

INTEGRIDAD CONCEPTUAL (Es el tema subyacente o la visión que unifica el diseño del sistema en todos los niveles)

I1 - ¿En qué nivel se muestra un enfoque de desarrollo completo de una aplicación móvil en Android?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I2 - ¿Se considera el uso de patrones de diseño dentro de la aplicación móvil en Android?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I3 - ¿En qué nivel la documentación incluye aspectos de diseño de una aplicación móvil en Android?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I4 - ¿En qué nivel se permite el entendimiento y diferenciación de cada componentes de la plataforma, así como el uso que debería tener en una aplicación?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I5 - ¿Se brinda una estructura replicable en diferentes proyectos móviles en Android?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I6 - ¿Se facilita la comunicación y aprendizaje de las decisiones arquitectónicas tomadas dentro de la aplicación?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I7 - ¿En qué nivel la estructuración de los proyectos considera la división de tareas entre desarrolladores?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

I8 - ¿En qué nivel puede ser usada como la base para el análisis y construcción de la aplicación móvil?

a) Muy poco b) Poco c) Regular d) Suficiente e) Mucho

8.2. Resultados de la Encuesta

A continuación se presentan los puntajes obtenidos en la encuesta aplicada:

Pre Test							
Pregunta	experto 1	experto 2	experto 3	experto 4	experto 5	experto 6	experto 7
M1	3	4	3	4	3	4	2
M2	4	3	2	3	4	3	3
M3	3	3	3	4	4	2	4
M4	3	3	3	2	4	3	3
M5	3	3	2	2	4	3	3
M6	3	2	2	2	3	3	2
M7	4	4	4	5	4	4	5
M8	3	2	3	4	3	3	4
	26	**24**	**22**	**26**	**29**	**25**	**26**
F1	2	2	2	3	2	3	2
F2	2	3	2	2	3	3	2
F3	2	2	2	2	2	3	2
F4	3	3	2	4	3	3	3
F5	3	4	4	4	5	4	4
F6	3	4	1	3	4	3	4
F7	3	4	3	1	4	4	3
F8	2	3	3	2	2	2	2
	20	**25**	**19**	**21**	**25**	**25**	**22**
E1	2	2	2	2	2	3	2
E2	3	2	4	3	3	3	3
E3	4	3	3	3	3	2	3
E4	4	3	3	4	5	3	3
E5	3	3	2	3	4	3	3
E6	4	3	2	3	3	4	3
E7	2	2	3	2	2	1	2
E8	2	3	4	4	3	2	3
	24	**21**	**23**	**24**	**25**	**21**	**22**
I1	3	2	2	2	1	2	2
I2	2	3	4	4	3	2	3
I3	4	3	2	3	2	3	2
I4	4	4	3	4	4	5	4
I5	3	3	3	2	4	3	4
I6	4	3	3	2	4	4	3
I7	2	2	2	1	2	2	2
I8	3	2	4	3	3	4	3
	25	**22**	**23**	**21**	**23**	**25**	**23**
TOTAL	**95**	**92**	**87**	**92**	**102**	**96**	**93**

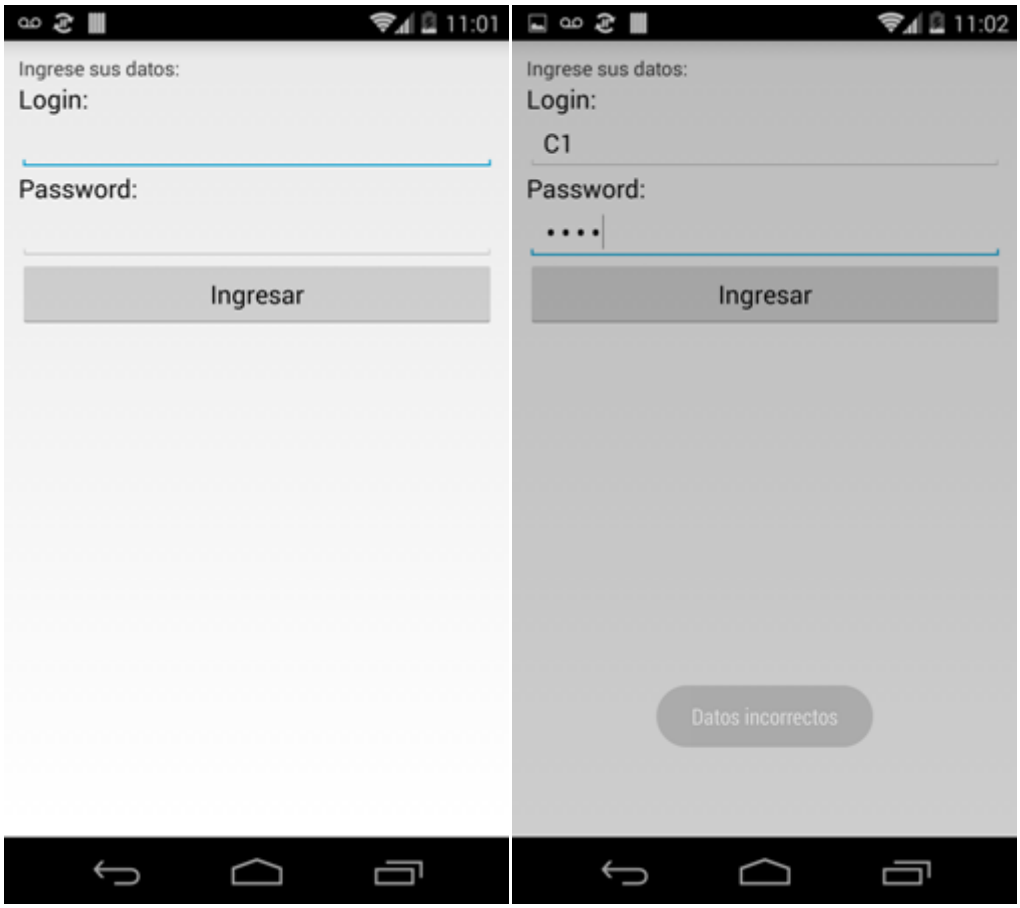
Post Test							
Pregunta	experto 1	experto 2	experto 3	experto 4	experto 5	experto 6	experto 7
M1	3	4	3	4	3	4	3
M2	4	3	4	3	4	3	3
M3	3	4	3	4	4	2	4
M4	4	3	3	2	4	3	3
M5	3	3	2	3	4	3	3
M6	4	2	2	2	3	3	2
M7	4	4	4	5	4	4	5
M8	3	2	3	4	3	3	4
	28	**25**	**24**	**27**	**29**	**25**	**27**
F1	3	2	2	3	2	3	2
F2	2	3	3	3	3	3	2
F3	3	2	2	3	2	3	2
F4	3	3	2	4	3	3	3
F5	3	4	4	4	5	4	4
F6	3	4	1	3	4	3	4
F7	3	4	3	3	4	4	3
F8	2	3	3	2	2	2	2
	22	**25**	**20**	**25**	**25**	**25**	**22**
E1	2	3	2	3	2	3	2
E2	3	3	4	4	3	3	3
E3	4	3	3	4	3	2	3
E4	4	3	3	4	5	3	3
E5	3	3	3	3	4	3	3
E6	4	3	3	3	3	4	3
E7	2	2	3	2	2	2	2
E8	2	3	4	4	3	2	3
	24	**23**	**25**	**27**	**25**	**22**	**22**
I1	4	3	3	3	3	3	3
I2	3	3	4	4	3	2	3
I3	4	4	3	3	3	3	3
I4	4	4	3	4	4	5	4
I5	3	3	4	2	4	3	4
I6	4	3	3	2	4	4	3
I7	3	3	2	3	2	2	2
I8	3	3	4	3	3	4	3
	28	**26**	**26**	**24**	**26**	**26**	**25**
TOTAL	**102**	**99**	**95**	**103**	**105**	**98**	**96**

8.3. Capturas de pantalla de la Aplicación de Taxis

Se presentan las siguientes capturas tomadas de la aplicación de ejemplo de uso de la arquitectura propuesta:

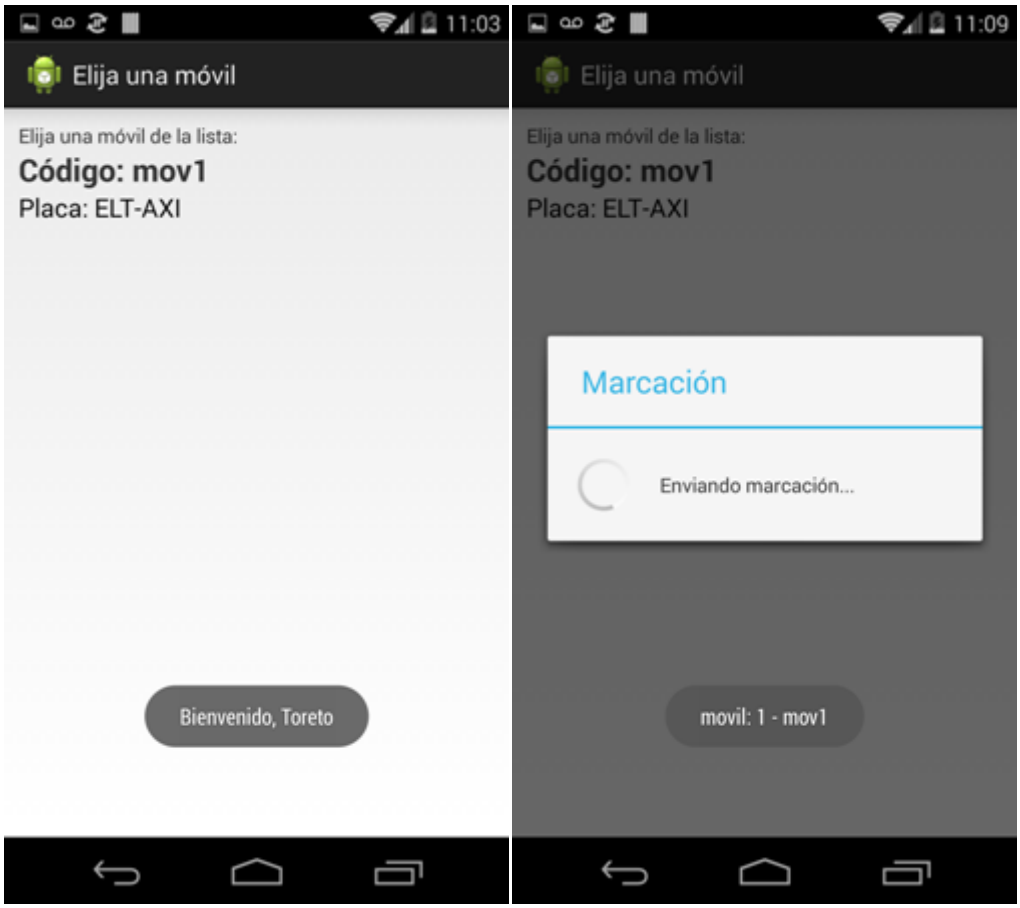
Login

Pantalla de ingreso a la aplicación móvil.



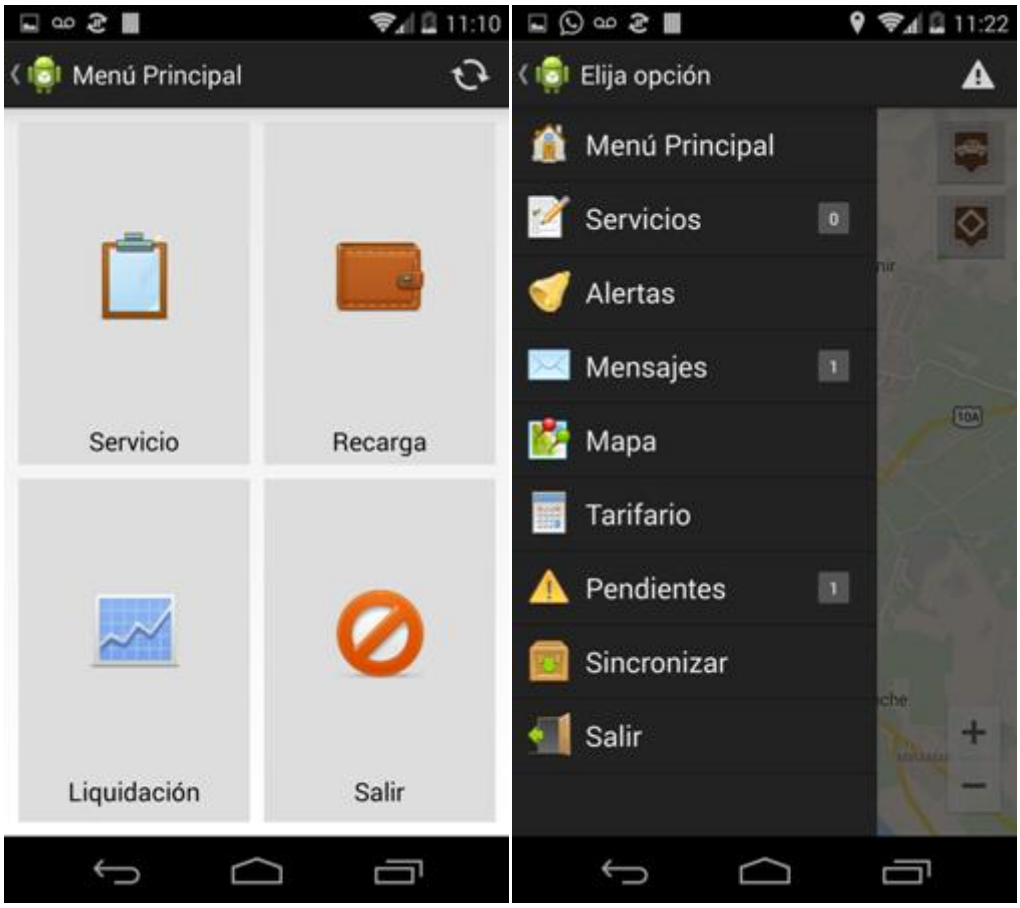
Marcación

Se elige la móvil con la cual el usuario está ingresando al sistema.



Menú Principal

Pantalla principal donde se muestran las principales opciones de la aplicación y el resto mediante una pantalla deslizable.



Servicios

Pantalla donde se puede elegir el servicio a atender.

Servicio Asignado

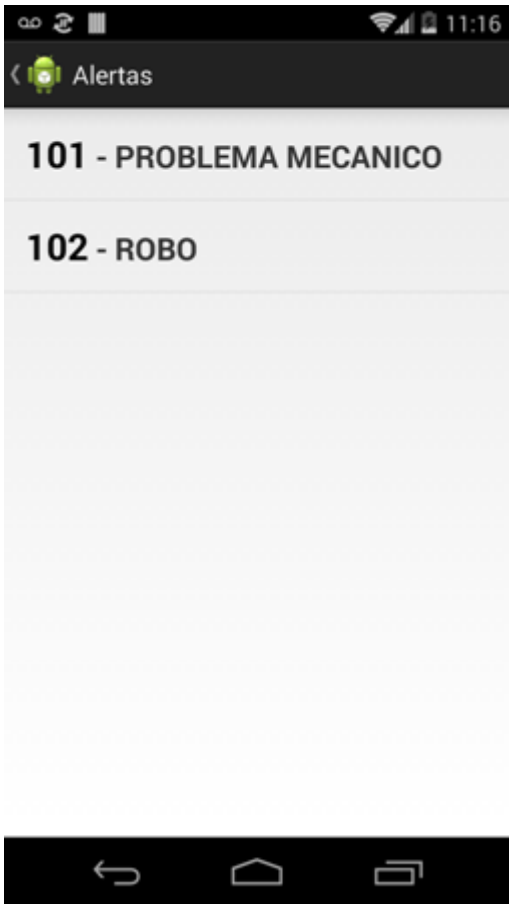
Pantalla con los datos del servicio asignado y las acciones que se pueden realizar.

Servicio Libre

Pantalla con los datos del servicio libre y las acciones que se pueden realizar.

Alertas

Pantalla que lista las alertas que pueden ser enviadas.

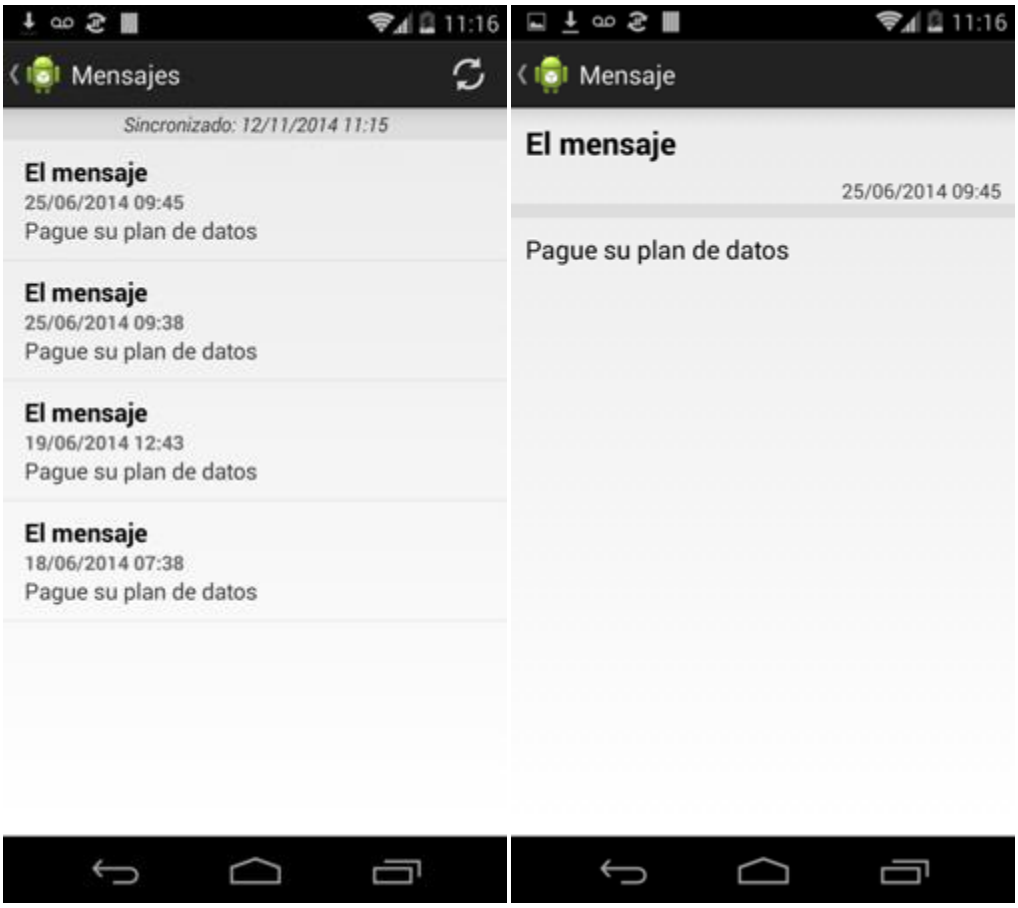


Mapa

Pantalla que muestra el mapa con la ubicación de la móvil

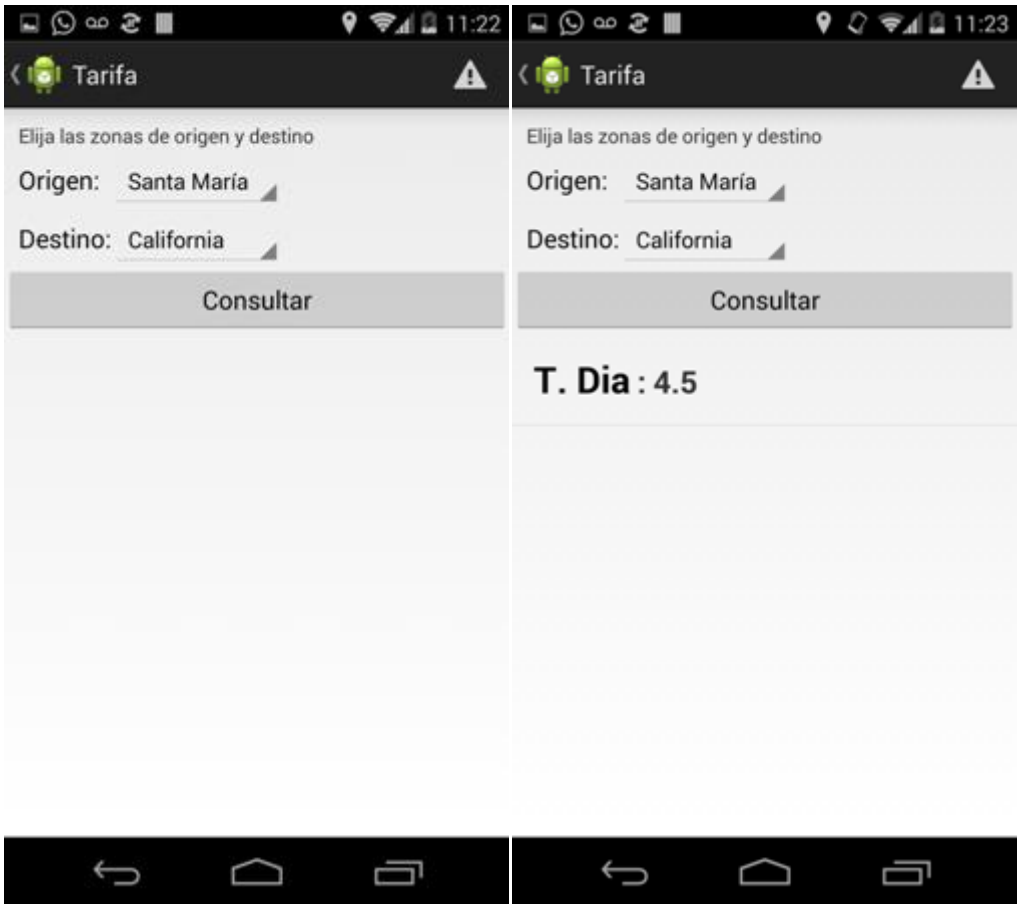
Mensajes

Se listan los mensajes que se han recibido y se puede ver el detalle de cada uno de ellos.



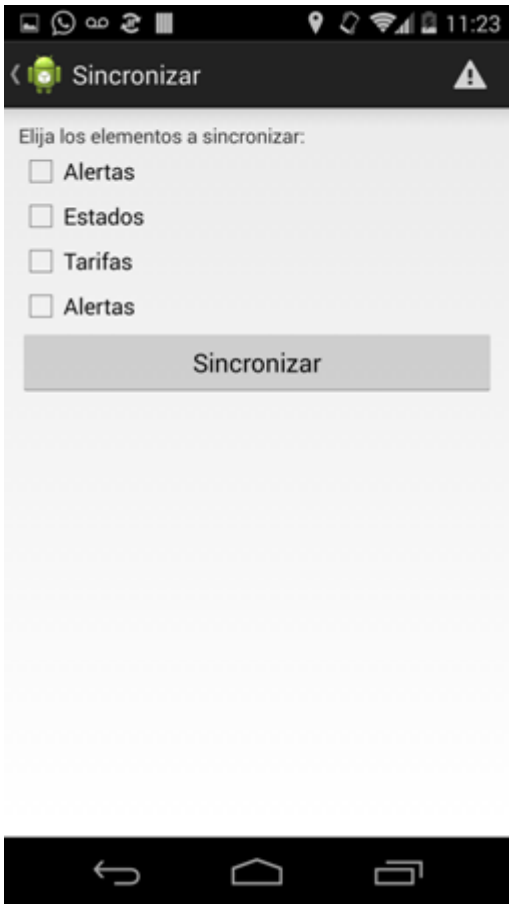
Tarifario

Se muestra el precio de la tarifa dependiendo del origen y el destino



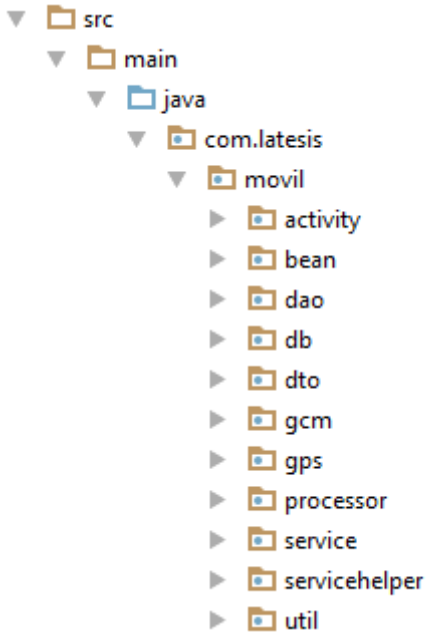
Sincronización

Pantalla donde se muestra las opciones de sincronización

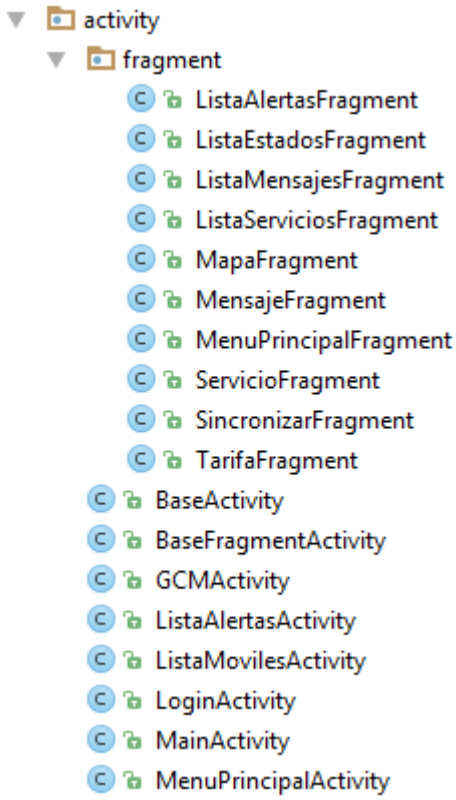


8.4. Estructura de Aplicación de Taxis

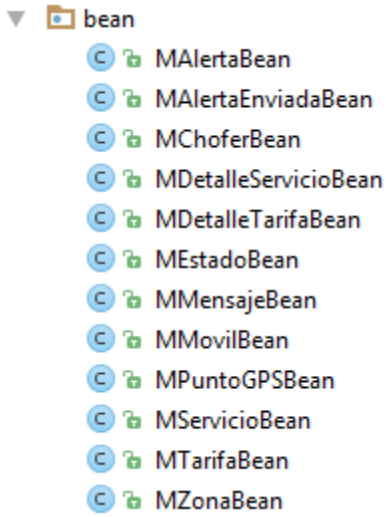
Se presenta la estructura de carpetas utilizada en la aplicación de ejemplo de taxis, la cual sigue las directrices indicadas en la arquitectura propuesta.



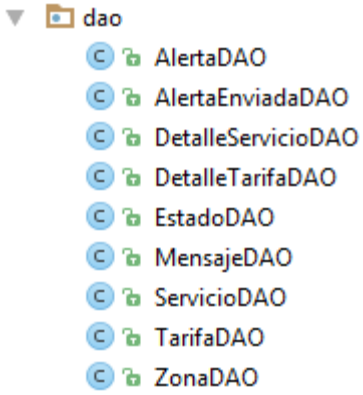
Paquete activity



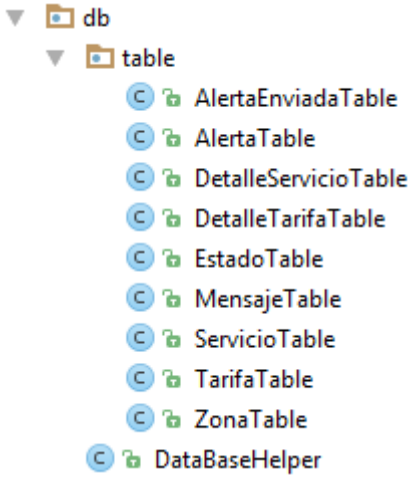
Paquete bean



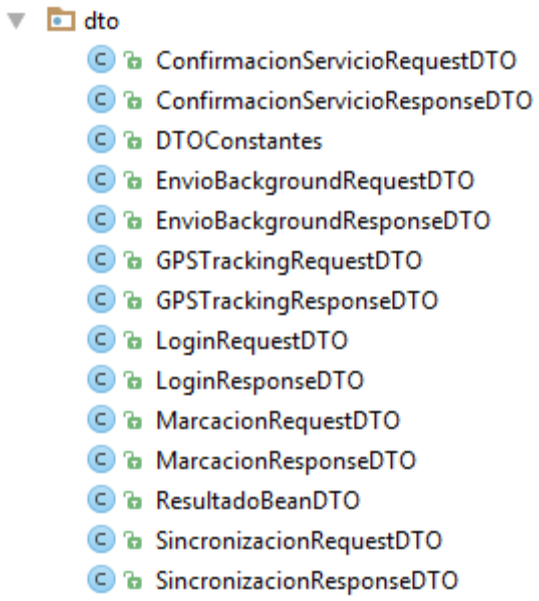
Paquete dao



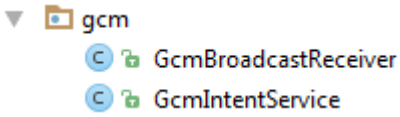
Paquete db



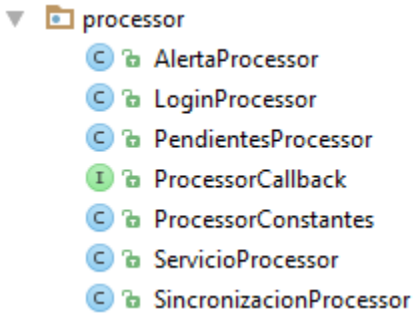
Paquete dto



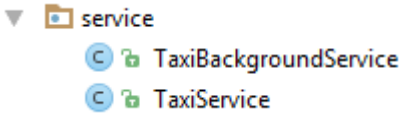
Paquete gcm



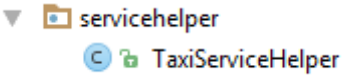
Paquete processor



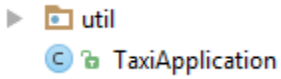
Paquete service



Paquete servicehelper



Paquete util



8.5.Fragmentos de Código de Elementos Principales

Se presenta algunos fragmentos de código significativos para ejemplificar la implementación de los elementos presentados en la arquitectura propuesta.

LoginActivity - Activity

```
public class LoginActivity extends Activity implements View.OnClickListener{

    private static final String TAG = LoginActivity.class.getSimpleName();
    private Long requestId;
    private BroadcastReceiver requestReceiver;

    Button btnIngresar;
    TextView txtUsername,txtPassword;
    private TaxiServiceHelper taxiServiceHelper;

    ProgressDialog progressDialog;
    TaxiApplication taxiApplication;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.activity_login);
        btnIngresar=(Button)findViewById(R.id.btnIngresar);
        txtUsername=(TextView)findViewById(R.id.txtUsername);
        txtPassword=(TextView)findViewById(R.id.txtPassword);
        btnIngresar.setOnClickListener(this);

        progressDialog=new ProgressDialog(this);
        progressDialog.setTitle("Login");
        progressDialog.setMessage("Autenticando datos...");

        taxiApplication=(TaxiApplication)getApplication();


    }

    @Override
    protected void onResume() {
        super.onResume();

        IntentFilter filter = new IntentFilter(Constantes.ACTION_REQUEST_RESULT);
        requestReceiver = new BroadcastReceiver() {

            @Override
            public void onReceive(Context context, Intent intent) {

                long resultRequestId = intent.getLongExtra(Constantes.EXTRA_REQUEST_ID, 0);
                Log.d(TAG, "Received intent " + intent.getAction() + ", request ID "  + resultRequestId);

                if (resultRequestId == requestId) {
                    Log.d(TAG, "Result is for our request ID");
                    requestTerminado();
                } else {
                    Log.d(TAG, "Result is NOT for our request ID");
                }

            }
        };
```

```
        taxiServiceHelper=TaxiServiceHelper.getInstance(this);
        this.registerReceiver(requestReceiver, filter);

        if (requestId == null) {
        } else if (taxiServiceHelper.isRequestPending(requestId)) {
            progressDialog.show();
        } else {
            requestTerminado();
        }
    }


    @Override
    protected void onPause() {
        super.onPause();
        if (requestReceiver != null) {
            try {
                this.unregisterReceiver(requestReceiver);
            } catch (IllegalArgumentException e) {
                Log.e(TAG, e.getLocalizedMessage(), e);
            }
        }
    }


     @Override
     public void onClick(View view) {
         final LocationManager manager = (LocationManager) getSystemService( Context.LOCATION_SERVICE );

         String login=txtUsername.getText().toString();
         String password=txtPassword.getText().toString();
         if("".equals(login.trim())||"".equals(password.trim())){
             Toast.makeText(this,"Debe ingresar los datos",Toast.LENGTH_SHORT).show();
             return;
         }

         progressDialog.show();
         requestId = taxiServiceHelper.login(login,password);
     }


    private void requestTerminado(){
        int resultCode = taxiServiceHelper.getRequestResultCode(requestId);
        Log.d(TAG, "Result code = " + resultCode);
        if (resultCode == ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_OK) {
            progressDialog.dismiss();
            if(taxiApplication.choferActual!=null){
                Toast.makeText(this,"Bienvenido, "+taxiApplication.choferActual.getNombres(),Toast.LENGTH_SHORT).show();
                Intent i=new Intent(this,ListaMovilesActivity.class);
                startActivity(i);
            }
        } else if (resultCode == ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_NO_HAY_MOVILES) {
            Toast.makeText(LoginActivity.this,"No hay moviles libres",Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        } else if (resultCode == ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_INCORRECTO) {
            Toast.makeText(LoginActivity.this,"Datos incorrectos",Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        } else if (resultCode == ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_ERROR_GCM) {
            Toast.makeText(LoginActivity.this,"Error configurando cuenta de GCM",Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        } else if (resultCode == ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_ERROR_GOOGLE_PLAY_SERVICES) {
            Toast.makeText(LoginActivity.this,"Error con el uso de Google Play Services",Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        } else {
            Toast.makeText(LoginActivity.this,"Ocurrió un error...",Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        }
        requestId=null;
    }
}
```

TaxiServiceHelper - ServiceHelper

```
public class TaxiServiceHelper {

    private static final String TAG = TaxiServiceHelper.class.getName();
    private static TaxiServiceHelper instance;
    private static long c=0;

    private static final String REQUEST_ID = "REQUEST_ID";
    public static final String loginHashkey = "LOGIN";

    private static Object lock = new Object();
    private Map<String,Long> pendingRequests = new HashMap<String,Long>();
    private Map<Long,Integer> requestsResultCode = new HashMap<Long,Integer>();

    private Context ctx;

    private TaxiServiceHelper(Context ctx){
        this.ctx = ctx.getApplicationContext();
    }

    public static TaxiServiceHelper getInstance(Context ctx){
        synchronized (lock) {
            if(instance == null){
                instance = new TaxiServiceHelper(ctx);
            }
        }
        return instance;
    }


     public long login(String login,String password) {
         Log.d(TAG, ">>login...");
         long requestId = generateRequestID();
         pendingRequests.put(loginHashkey, requestId);

         ResultReceiver serviceCallback = new ResultReceiver(null){
             @Override
             protected void onReceiveResult(int resultCode, Bundle resultData) {
                 handleLoginResponse(resultCode, resultData);
             }
         };

         Intent intent = new Intent(this.ctx, TaxiService.class);
         intent.putExtra(EXTRA_ACTION, TAXI_LOGIN_ACTION);
         intent.putExtra(EXTRA_SERVICE_CALLBACK, serviceCallback);
         intent.putExtra(EXTRA_LOGIN, login);
         intent.putExtra(EXTRA_PASSWORD, password);
         intent.putExtra(REQUEST_ID, requestId);

         this.ctx.startService(intent);

         return requestId;
     }


    private void handleLoginResponse(int resultCode, Bundle resultData){

        Intent origIntent = (Intent)resultData.getParcelable(EXTRA_ORIGINAL_INTENT);

        if(origIntent != null){
            long requestId = origIntent.getLongExtra(REQUEST_ID, 0);

            requestsResultCode.put(pendingRequests.get(loginHashkey),resultCode);
            pendingRequests.remove(loginHashkey);

            Intent resultBroadcast = new Intent(ACTION_REQUEST_RESULT);
            resultBroadcast.putExtra(EXTRA_REQUEST_ID, requestId);
            resultBroadcast.putExtra(EXTRA_RESULT_CODE, resultCode);

            ctx.sendBroadcast(resultBroadcast);
        }
    }
}
```

TaxiService - Service

```
public class TaxiService extends IntentService {

    protected static final String TAG = TaxiService.class.getSimpleName();

    private static final int REQUEST_INVALID = -1;

    private ResultReceiver mCallback;

    private Intent mOriginalRequestIntent;
    private LoginProcessor loginProcessor;
    private SincronizacionProcessor sincronizacionProcessor;
    private AlertaProcessor alertaProcessor;
    private ServicioProcessor servicioProcessor;

    public TaxiService(){
        super("TaxiService");
    }

    @Override
    public void onCreate() {
        super.onCreate();

        if(loginProcessor==null)
            loginProcessor=new LoginProcessor(getApplicationContext());
        if(sincronizacionProcessor==null)
            sincronizacionProcessor=new SincronizacionProcessor(getApplicationContext());
        if(alertaProcessor==null)
            alertaProcessor=new AlertaProcessor(getApplicationContext());
        if(servicioProcessor==null)
            servicioProcessor=new ServicioProcessor(getApplicationContext());

    }


    @Override
    protected void onHandleIntent(Intent requestIntent) {
        Log.d(TAG,">>onHandleIntent...");
        mOriginalRequestIntent = requestIntent;

        int action = requestIntent.getIntExtra(EXTRA_ACTION, -1);
        mCallback = requestIntent.getParcelableExtra(EXTRA_SERVICE_CALLBACK);
        switch (action){
            case TAXI_LOGIN_ACTION:
                loginProcessor.login(requestIntent.getStringExtra(EXTRA_LOGIN),
                        requestIntent.getStringExtra(EXTRA_PASSWORD),
                        makeProcessorCallback());
                break;
        }
    }


    private ProcessorCallback makeProcessorCallback() {
        ProcessorCallback callback = new ProcessorCallback() {

            @Override
            public void send(int resultCode, String resultData) {
                Log.d(TAG,"send() resultCode: "+resultCode);
                if (mCallback != null) {
                    Bundle data= new Bundle();
                    data.putParcelable(EXTRA_ORIGINAL_INTENT, mOriginalRequestIntent);
                    data.putString(EXTRA_RESULT_DATA, resultData);
                    mCallback.send(resultCode,data);
                    //mCallback.send(resultCode, getOriginalIntentBundle());
                }
            }
        };
        return callback;
    }

    protected Bundle getOriginalIntentBundle() {
        Bundle originalRequest = new Bundle();
        originalRequest.putParcelable(EXTRA_ORIGINAL_INTENT, mOriginalRequestIntent);
        return originalRequest;
    }
}
```

TaxiApplication - Application

```
public class TaxiApplication extends Application{

    public MChoferBean choferActual;
    public MMovilBean movilActual;
    public ArrayList<MMovilBean> listaMoviles;
    public boolean gpsActivo=false;
    public MMensajeBean mensajeActual;
    public MServicioBean servicioActual;
    private static final long GPS_SEND_INTERVAL_IN_SECONDS=30;

    public void cerrarSesion(Context context){
        choferActual=null;
        movilActual=null;
        listaMoviles=null;
        mensajeActual=null;
        servicioActual=null;
    }

    public void iniciarGPSTracking(){
        gpsActivo=true;
        Calendar cal = Calendar.getInstance();
        Intent intent = new Intent(this, GPSTrackingService.class);
        intent.putExtra("enviarGPS",true);
        PendingIntent pintent = PendingIntent.getService(this, 0, intent, 0);
        AlarmManager alarm = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
        alarm.setRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(), GPS_SEND_INTERVAL_IN_SECONDS*1000, pintent);
    }
```

LoginProcessor – BusinessProcessor

```
public class LoginProcessor {

    protected static final String TAG = LoginProcessor.class.getSimpleName();

    private Context mContext;
    TaxiApplication taxiApplication;

    AlertaDAO alertaDAO;
    DetalleTarifaDAO detalleTarifaDAO;
    EstadoDAO estadoDAO;
    TarifaDAO tarifaDAO;
    ZonaDAO zonaDAO;

    public LoginProcessor(Context context) {
        mContext = context;
        taxiApplication=(TaxiApplication) context.getApplicationContext();

        alertaDAO=new AlertaDAO(context.getApplicationContext());
        detalleTarifaDAO=new DetalleTarifaDAO(context.getApplicationContext());
        estadoDAO=new EstadoDAO(context.getApplicationContext());
        tarifaDAO=new TarifaDAO(context.getApplicationContext());
        zonaDAO=new ZonaDAO(context.getApplicationContext());

    }
```

```
    public void login(String login, String password,ProcessorCallback callback)  {
        Gson gson=new Gson();
        LoginRequestDTO loginRequestDTO=new LoginRequestDTO();
        loginRequestDTO.setLogin(login);
        loginRequestDTO.setPassword(password);
        String data=gson.toJson(loginRequestDTO);
        Log.d(TAG,"data : "+data);

        String regid;
        try{
            GoogleCloudMessaging gcm;
            gcm = GoogleCloudMessaging.getInstance(mContext);
            regid = getRegistrationId(mContext);

            if (regid.isEmpty()) {
                Log.d(TAG,"registrar en GCM");
                if (gcm == null) {
                    gcm = GoogleCloudMessaging.getInstance(mContext);
                }
                regid = gcm.register(Constantes.GCM_SENDER_ID);
                Log.d(TAG,"...registro completo - regid:"+regid);
                storeRegistrationId(mContext, regid);

            }else{
                Log.d(TAG,"...previamente registrado - regid:"+regid);
            }

        }catch(Exception e){
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_ERROR_GCM, "");
            return;
        }


        LoginResponseDTO loginResponseDTO=null;
        try{
            String url = Constantes.URL + "login";
            Log.d(TAG,"url : "+url);
            RestTemplate restTemplate = new RestTemplate();
            restTemplate.getMessageConverters().add(new StringHttpMessageConverter());
            String result = restTemplate.postForObject(url, data, String.class);
            loginResponseDTO=gson.fromJson(result,LoginResponseDTO.class);

        }catch(Exception e){
            Log.e(TAG,e.getMessage());
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_ERROR, "");
            return;
        }

        if(DTOConstantes.RESULTADO_LOGIN_OK==loginResponseDTO.getResultadoLogin()){
            taxiApplication.choferActual=loginResponseDTO.getChofer();
            taxiApplication.listaMoviles=loginResponseDTO.getListaMoviles();
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_OK,"");
        }else if(DTOConstantes.RESULTADO_LOGIN_INCORRECTO==loginResponseDTO.getResultadoLogin()){
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_INCORRECTO,"");
        }else if(DTOConstantes.RESULTADO_LOGIN_NO_HAY_MOVILES==loginResponseDTO.getResultadoLogin()){
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_NO_HAY_MOVILES,"");
        }else{
            callback.send(ProcessorConstantes.PROCESSOR_RESULTADO_LOGIN_ERROR, "");
        }
    }
}
```

AlertaDAO - DAO

```
public class AlertaDAO {
    final DataBaseHelper dbHelper;
    SQLiteDatabase db;

    public AlertaDAO(Context context) {
        dbHelper = DataBaseHelper.getInstance(context);
        this.db= dbHelper.getWritableDatabase();
    }

    public void close() {
        this.dbHelper.close();
    }

    public long insertar(MAlertaBean alerta) {
        ContentValues cv = new ContentValues();
        cv.put(AlertaTable.C_ID_BD, alerta.getIdBD());
        cv.put(AlertaTable.C_CODIGO, alerta.getCodigo());
        cv.put(AlertaTable.C_NOMBRE, alerta.getNombre());
        cv.put(AlertaTable.C_DESCRIPCION, alerta.getDescripcion());
        cv.put(AlertaTable.C_IMPORTANCIA, alerta.getImportancia());
        cv.put(AlertaTable.C_COLOR, alerta.getColor());

        long id = -1;
        try {
            id = db.insertWithOnConflict(AlertaTable.TABLE_NAME, // tabla donde se va a insertar
                    null, cv, // valores a insertar
                    SQLiteDatabase.CONFLICT_IGNORE); // si hay un problema ignorarlo
        } catch(Exception e){

        }
        return id;
    }


    public Cursor listarTodos() {
        return db.query(AlertaTable.TABLE_NAME, null, null, null, null, null, null);
    }

    public void eliminarTodos() {
        try {
            db.delete(AlertaTable.TABLE_NAME, // tabla
                    null, // where
                    null);
        } catch(Exception e){
        }

    }


    public MAlertaBean buscarAlertaPorIdBD(Long idBD){
        MAlertaBean bean=null;

        try {

            String[] args={""+idBD};
            Cursor cursor=db.rawQuery("SELECT "+
                    AlertaTable.C_ID_MOVIL+","+
                    AlertaTable.C_ID_BD+","+
                    AlertaTable.C_CODIGO+","+
                    AlertaTable.C_NOMBRE+","+
                    AlertaTable.C_DESCRIPCION+","+
                    AlertaTable.C_IMPORTANCIA+","+
                    AlertaTable.C_COLOR+
                    " FROM "+AlertaTable.TABLE_NAME+
                    " WHERE "+AlertaTable.C_ID_BD+"=?",args);

            try {
                if(cursor.moveToNext()){
                    bean=new MAlertaBean();
                    bean.setIdMovil(cursor.getInt(cursor.getColumnIndex(AlertaTable.C_ID_MOVIL)));
                    bean.setIdBD(cursor.getLong(cursor.getColumnIndex(AlertaTable.C_ID_BD)));
                    bean.setCodigo(cursor.getString(cursor.getColumnIndex(AlertaTable.C_CODIGO)));
                    bean.setNombre(cursor.getString(cursor.getColumnIndex(AlertaTable.C_NOMBRE)));
                    bean.setDescripcion(cursor.getString(cursor.getColumnIndex(AlertaTable.C_DESCRIPCION)));
                    bean.setImportancia(cursor.getInt(cursor.getColumnIndex(AlertaTable.C_IMPORTANCIA)));
                    bean.setColor(cursor.getString(cursor.getColumnIndex(AlertaTable.C_COLOR)));
```

```
                }
            } finally {
                cursor.close();
            }
        } catch(Exception e){

        }
        return bean;
    }
}
```

MAlertaBean - Bean

```
public class MAlertaBean {
    private Integer idMovil;

    private Long idBD;
    private String codigo;
    private String nombre;
    private String descripcion;
    private Integer importancia;
    private String color;

    public Integer getIdMovil() {
        return idMovil;
    }
    public void setIdMovil(Integer idMovil) {
        this.idMovil = idMovil;
    }
    public Long getIdBD() {
        return idBD;
    }
    public void setIdBD(Long idBD) {
        this.idBD = idBD;
    }
    public String getCodigo() {
        return codigo;
    }


    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDescripcion() {
        return descripcion;
    }
    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
    public Integer getImportancia() {
        return importancia;
    }
    public void setImportancia(Integer importancia) {
        this.importancia = importancia;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
}
```