Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Localización de personas y objetos a través de realidad aumentada en sistemas móviles

Autor:

D. Héctor Del Campo Pando

Tutores:

Dr. Diego R. Llanos Ferraris

D. Daniel Barba Gutiérrez

Agradecimientos

A mi familia, sin cuyo apoyo y comprensión este proyecto no habría sido posible.

A mis sujetos de prueba, aguantando mis continuas interrupciones, ayudándome a generar la mejor interfaz de usuario posible.

A mis compañeros de RDNest, logrando hacerme sentir como uno más del equipo desde el primer día y echándome una mano cuando ha hecho falta, perdiendo parte de su tiempo.

A Diego y a Daniel, por darme la oportunidad de desarrollar este proyecto y orientarme estos últimos meses.

Y finalmente a la Escuela de Ingeniería Informática en su totalidad, cuyo ambiente y consejos me han formado profesionalmente y personalmente.

Resumen

Este Trabajo De Fin de Grado presenta *Skywalker*, una aplicación para dispositivos móviles con sistema operativo Android o iOS, que permite la localización de objetos y personas en entornos cerrados a través de la realidad aumentada, sirviéndose de los distintos sensores que estos dispositivos móviles poseen.

En paralelo a este proyecto, se ha desarrollado XtremeLoc, un sistema externo cuyo objetivo es la localización de elementos en entornos cerrados, y que facilita una API pública para su utilización.

Utilizando los datos proporcionados por XtremeLoc, *Skywalker* proporciona una fácil localización de los elementos registrados en XtremeLoc, e incluso sirve como elemento localizable para XtremeLoc.

Abstract

This Bachelor's thesis presents *Skywalker*, an application for Android and iOS systems, which allows locating objects and users in closed environments through augmented reality, making use of device's sensors.

Alongside this project, XtremeLoc has been developed, an external system whose objective is to locate elements in closed environments; XtremeLoc publishes an API so other systems can use it.

Using the data provided by XtremeLoc, *Skywalker* allows an easy location of the registered elements in XtremeLoc, even working as a trackable element for XtremeLoc.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde la aparición de los SmartPhones, tal y como los conocemos hoy en día, en el año 2007 con el primer iPhone, la potencia computacional y las capacidades de estos dispositivos no ha parado de crecer, además de la expansión generada en el desarrollo de aplicaciones para estos dispositivos, cuyo crecimiento ha servido para que nuestros móviles puedan hacer casi cualquier cosa, y hoy en día no es difícil ver gente cuyo único ordenador es su dispositivo móvil.

Sin embargo, las únicas aplicaciones que sirven para guiarnos en el mundo real, son las clásicas aplicaciones de mapas en dos dimensiones, y cuyo funcionamiento se limita a espacios exteriores con buena cobertura de los clásicos satélites.

Aprovechando este poder computacional, y para presentar un concepto de geolocalización diferente, se ha desarrollado el sistema descrito en este documento.

1.1. Contexto e Historia

A pesar de que los sistemas móviles ya están asentados, la aparición de los dos principales sistemas operativos, Android e iOS, es reciente. A continuación se describen estos sistemas operativos, así como los dispositivos que los utilizan.

iOS Sistema operativo móvil, propiedad de Apple Inc., aparece por primera vez junto al primer Smartphone de esta compañía en el año 2007, únicamente disponible en arquitecturas ARM, y basado en otro sistema operativo de la misma compañía, macOS, los cuales comparten el núcleo del mismo. Escrito en C, C++, Objective-C y Swift, no sólo se encarga de la gestión del dispositivo, sino que ofrece a desarrolladores APIs con los que crear aplicaciones, las cuales son distribuidas a través de la tienda oficial App Store.

Hoy en día, se utiliza tanto en móviles, tabletas y reproductores de música del tipo iPod Touch. La base para cualquiera de estos dispositivos es la misma, sin embargo, en cada plataforma se puede disponer de funcionalidades únicas al tipo de dispositivo.

Android Sistema operativo móvil, propiedad de Google Inc., cuyo desarrollo es de código abierto, surgió como respuesta a iOS, saliendo al mercado en el año 2008. A lo largo de los años, ha crecido en uso, llegando a acaparar el 80 % del mercado mundial. Escrito en C, C++ y Java, puede ser utilizado por cualquier fabricante, modificándolo a su gusto, tanto en interfaz de usuario como en funcionalidades.

Cada año se amplían sus posibilidades, actualmente, disponible en teléfonos móviles, tabletas, coches, televisores, relojes y para el Internet de las Cosas. Como sucede en iOS, dependiendo de la plataforma de ejecución, se puede disponer de unas capacidades u otras, incluso de hardware distinto. Disponible en arquitecturas ARM, x86, MIPS e IBM Power.

1.2. Motivación

Hoy en día, la geolocalización, a pesar de estar muy expandida y perfeccionada, tiene varios problemas:

1. La precisión de los sistemas actuales en entornos indoor es irrisoria, llegando hasta el punto de no disponer de localización en estos entornos.
2. La no inmediata relación del espacio físico real con una posición señalada en un mapa multidimensional, debido a la dificultad de interpretación de dichos planos.

Este Trabajo de Final de Grado representa una solución para ambos problemas, sirviendose de un sistema externo al sistema aquí propuesto, XtremeLoc, el cual permite la localización en entornos equipados con dicho sistema, sin necesidad de disponer del sistema GPS tradicional, con un grado de precisión de pocos metros.

Por otro lado, este trabajo implementará un sistema de realidad aumentada para eliminar la necesidad de mapas bidimensionales o tridimensionales, usando así la propia realidad como asociación localización física-objeto.

El sistema desarrollado propone una forma diferente de orientación en el mundo real y localización de elementos de una forma sencilla y precisa, además de ser una solución económica al utilizar hardware de bajo coste, por parte del sistema externo a utilizar, y de dispositivos móviles, como teléfonos inteligentes o tabletas, para la localización por parte de los usuarios.

1.3. Objetivos

Skywalker tratará de dar respuesta a las necesidades antes mencionadas. Se intentará la utilización de los dispositivos ya existentes, sin necesidad de componentes adicionales -salvo aquellos necesarios para XtremeLoc-, y de las APIs ya publicadas por los distintos fabricantes y librerías públicas para su uso libre.

Este proyecto no trata de ser únicamente un documento académico, sino de ser un producto comercial, dando solución a necesidades de gobiernos y empresas, siendo un complemento al sistema de desarrollo externo XtremeLoc.

1.4. Definiciones y Acrónimos

A lo largo del documento se utilizarán los siguientes acrónimos y definiciones.

UI User Interface, Interfaz de Usuario.

FPS Fotogramas Por Segundo, medida del rendimiento para el dibujado de interfaces.

Closure Función evaluada en un entorno que depende de una o más variables de otro entorno.

TDD Test Driven Development, tipo de desarrollo donde primero se generan los tests a pasar y en último lugar se desarrolla el código.

1.5. Estructura del documento

La estructura del resto de la memoria es la siguiente:

El capítulo dos presenta el Estado del arte, proveniente de la expresión inglesa *state-of-the-art*, y que hace referencia a temas relativos a realidad aumentada, desarrollo para sistemas móviles, y localización espacial.

El capítulo tres trata sobre el análisis extraído del trabajo previo al desarrollo propiamente dicho.

El capítulo cuatro contiene la planificación para el desarrollo del trabajo presentado, tratando temas como alcance, presupuesto, planificación o riesgos.

El capítulo cinco detalla el diseño de la aplicación, empezando por un punto de visto global, para acabar hablando del diseño especifico a cada plataforma objetivo.

El capítulo seis contiene información relativa a la implementación del sistema, al igual que el capítulo cinco, desde los puntos de vista globales y específicos. Este capítulo puede tomarse como un manual para desarrolladores.

El capítulo siete detalla las pruebas llevadas a cabo en el sistema, con el fin de comprobar su correcto funcionamiento, aquí se tratan tanto pruebas automáticas del sistema, como pruebas en dispositivos físicos reales, con integración con XtremeLoc.

El capítulo ocho se corresponde con el manual de usuario.

El último capítulo, el noveno, expone las conclusiones extraídas de este desarrollo, a la vez que presenta ideas sobre trabajos futuros realizables, concluyendo este Trabajo de Fin de Grado.

Capítulo 2

Estado del arte

Este capitulo describe el Estado del Arte respecto a la realidad aumentada, la localización física de elementos y el desarrollo móvil.

2.1. Geolocalización y orientación espacial

La cartografía existe desde el milenio VII a. C [39], y desde entonces no ha cambiado mucho, ha mejorado en técnicas y herramientas, pero la idea sigue siendo la misma, y aún estando en pleno siglo XXI d.C mucha gente es incapaz de orientarse con un mapa con facilidad, muestra de ello es el turismo en grandes ciudades, donde se puede observar como mucha gente se pierde y ha de preguntar a pesar de llevar consigo un mapa simplificado del terreno.

Hoy en día, gracias a los móviles podemos usar el GPS fácilmente, ver hacia donde estamos orientados, y no solo eso, sino que las señales WiFi y Bluetooth que pueblan las ciudades y edificios son utilizadas por algunas aplicaciones y sistemas para geolocalizar a los usuarios de éstas. Mejorando la percepción y capacidad de los usuarios para orientarse, aunque siguen teniendo problemas de interpretación, y lo peor de todo, la ausencia de cobertura deja a muchos usuarios completamente perdidos intentando orientarse en terreno desconocido, y aunque parezca inverosímil, en grandes ciudades como Londres o Nueva York, sus propios ciudadanos se pierden en una ciudad que debería serles familiar.

Con una rápida búsqueda en Internet, se puede ver como no hay nuevas ideas en desarrollo para la geolocalización, algunos proyectos Open Source utilizados como entrenamiento por sus creadores tienen vagas ideas para solucionar este problema. Pero localizar alguna solución comercial, si es que la hay, no es tarea fácil.

El problema que muchos usuarios tienen para interpretar un mapa, radica en la proyección de un sistema bidimensional a uno tridimiensional, de un mapa plano al mundo real. *SkyWalker* plantea la eliminación de esta barrera, plantea la eliminación de la transformación de distintos sistemas de coordenadas, y ser capaces de localizar personas, objetos y lugares con un simple vistazo al mundo real.

2.2. Realidad aumentada, virtual y mixta

Hoy en día, el desarrollo de dispositivos móviles de todo tipo, ha propiciado la aparición de nuevos paradigmas de visualización, entretenimiento y comunicación, cada día surgen nuevos dispositivos con

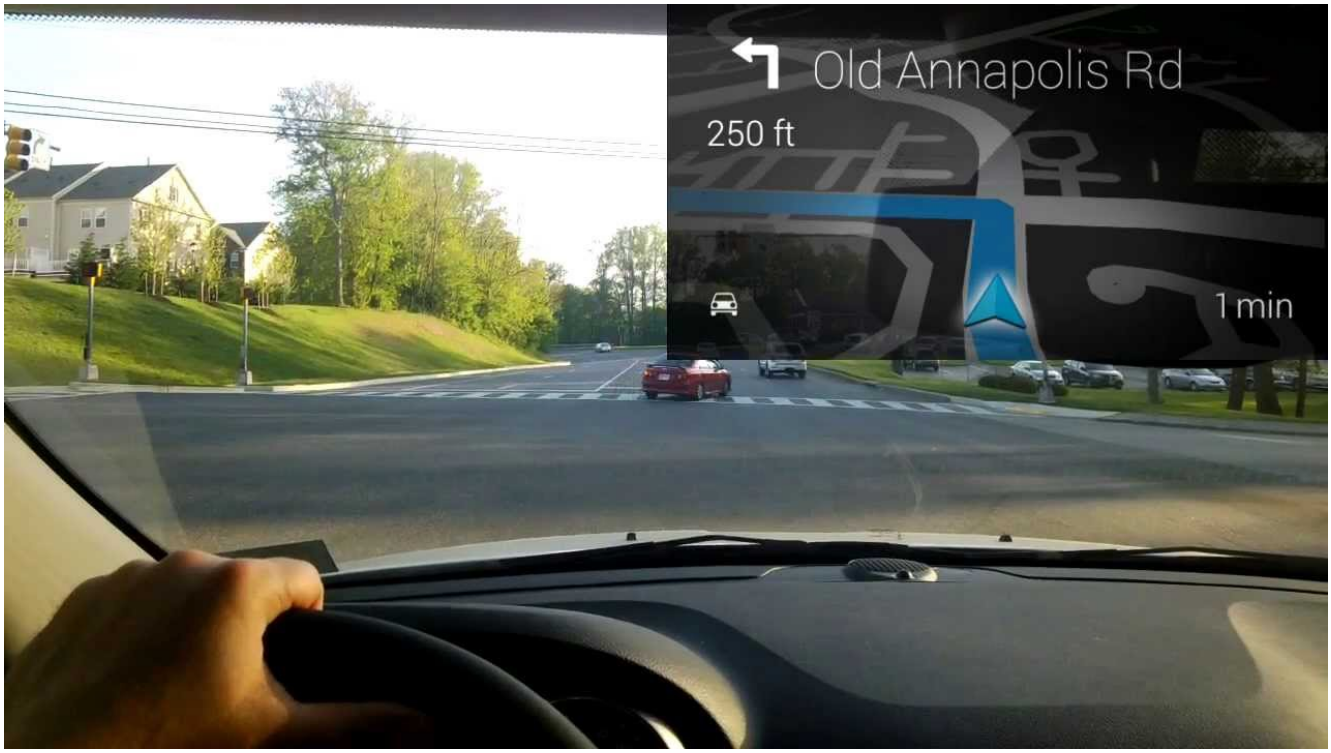


Figura 2.1: Interfaz de usuario en mapas usando Google Glass [29]

mejores capacidades computacionales y posibilidades funcionales, tales como Smartphones, Tabletas, Smartwatches, o Gafas inteligentes.

La taxonomía actual de los nuevos paradigmas de realidad es la siguiente:

1. Realidad virtual: Sustituye la realidad a través de dispositivos que nos permitan "sentir"que nos encontramos en otro lugar, sumergirnos en una realidad que no existe, transportarnos a una realidad construida, una realidad virtual.[31]

2. Realidad aumentada: Intenta es perfeccionar la realidad, agregar cosas a la misma a través de nuestros sentidos. Superponer otras realidades artificiales que se combinen con la realidad para ofrecer un mejor trato con la misma.[31]

3. Realidad mixta: Combinar mundos virtuales con el mundo real (físico) a tiempo real[40]

A nivel comercial, se están comercializando productos de realidad virtual pura, cómo *Oculus Rift* o *HTC Vive*, sin aparente aplicación para el objetivo de este trabajo, pero con un fuerte impulso en el sector de la medicina y en la industria del entretenimiento. Otros productos, como *Google Glass* o *Microsoft Hololens* se centran en la realidad aumentada, con aplicaciones ya propuestas en medicina, arquitectura o videollamadas. Con utilidad en el aprendizaje o en procesos industriales, pero ninguna de estas herramientas aprovecha el potencial de la realidad aumentada para la geolocalización, hay aplicaciones de mapas, pero son una simple translación de las aplicaciones ya existentes y sus interfaces de usuario, superponiendo una interfaz artificial y no integrada con la visión real del usuario.

2.3. Desarrollo móvil

Aunque en el mercado existen dos principales sistemas operativos móviles, iOS y Android, los cuales acaparan más del 90 % del mercado actual [27], otras empresas, han presentado y comercializado sus

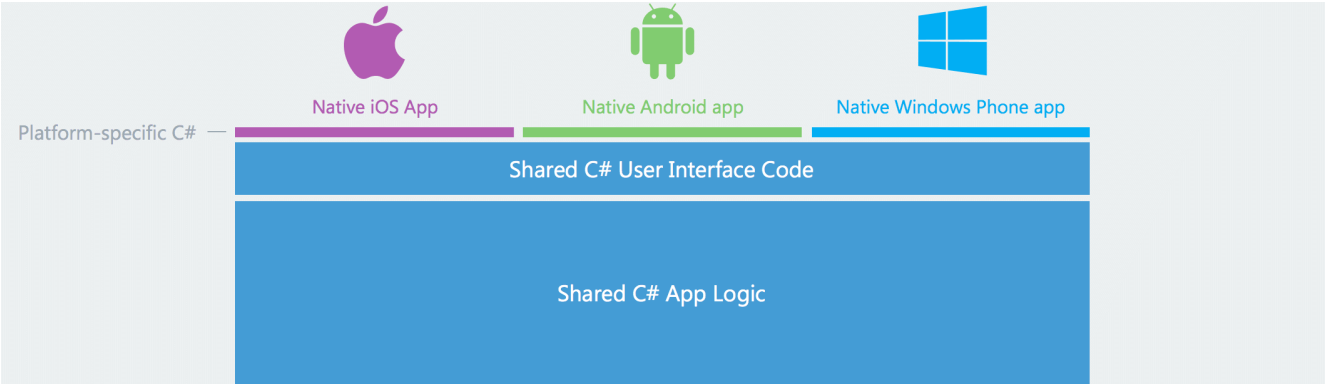


Figura 2.2: Modelo de desarrollo en Xamarin

propuestas, tales como FirefoxOS, Windows 10, o Ubuntu Touch.

No sólo el alto número de plataformas de desarrollo, cada una con sus peculiaridades, y la diferencia entre ellas retrasan el desarrollo de aplicaciones y sistemas nuevos, sino que la alta dificultad de algunas de las propuestas y sistemas desarrollados retrasan la entrada al mercado, y ponen trabas a el mantenimiento que se requiere en estas soluciones.

Para solucionar estos problemas y acelerar el desarrollo de nuevos sistemas, se han ideado soluciones de desarrollo multiplataforma, basadas en otras tecnologías ya existentes, tales como *Xamarin*, *Sencha*, *PhoneGap* para el desarrollo de aplicaciones tradicionales, y algunos motores gráficos comerciales se han ido transformando para permitir el desarrollo multiplataforma con un mínimo esfuerzo por parte de los desarrolladores, como *Unity3D* o *Unreal Engine*.

Antes de comenzar el desarrollo de *Skywalker*, se han investigado las distintas opciones a la hora de afrontar un desarrollo de este tipo.

2.3.1. Xamarin

La más importante de las soluciones, y la primera en ser estudiada, *Xamarin* permite el desarrollo en plataformas de *Microsoft*, *iOS*, *Android* y *Mac*, trabajando en el lenguaje *C#* y utilizando *Visual Studio* de *Microsoft*, compartiendo el mismo código para la lógica de negocio, pero necesitando una implementación única por plataforma de las vistas.

En el momento de este estudio, *Xamarin* no es viable para *Skywalker*, debido al alto coste que plantea la adquisición de una licencia de *Visual Studio*, y la imposibilidad de acceder a todas las APIs nativas de las plataformas objetivo, las cuales, precisamente, necesitaré para el desarrollo.

Sin embargo, *Xamarin* promete acelerar el desarrollo de ciertas aplicaciones con un fuerte componente en su lógica de negocio, ofreciendo a desarrolladores herramientas para comprobación de errores y rendimiento, y de distribución, además de que su desarrollo es continuo y cuenta con el apoyo de un gigante tecnológico como es *Microsoft* al ser este de su propiedad. Como muestra de sus posibilidades, destacar que empresas como *Slack*, *Kellogg's* o *Bosh* utilizan esta solución. [41]

Por ultimo, destacar el rendimiento que prometen sus desarrolladores, ya que el código generado es nativo a cada plataforma, y no se realiza ninguna interpretación por un motor en ejecución en el dispositivo.

2.3.2. Tecnologías web

En esta sección se engloban todas las plataformas que utilizan tecnologías web para la generación de aplicaciones multiplataforma. Entre ellas *PhoneGap*[30] y *Sencha*[32].

La mayoría de las soluciones de desarrollo multiplataforma que existen en el mercado utilizan tecnologías y lenguajes, que fueron creadas para el desarrollo web, tales como JavaScript, HTML5 y CSS, la idea que ha llevado a estas soluciones es simple, muchos desarrolladores están acostumbrados y tienen cierta experiencia con estas tecnologías, así que, ¿por que no llevarlas al desarrollo de aplicaciones?[33]
Algunas de estas soluciones simplemente permiten el desarrollo de las interfaces de usuario, otras más complejas permiten el desarrollo completo de las aplicaciones en estas tecnologías. Aquellas herramientas que simplifican solamente el desarrollo de las interfaces de usuario son de poca utilidad para *SkyWalker* debido a que las vistas tienen poca complejidad y son pocas.
Por otra parte, tenemos aquellas herramientas y frameworks que permiten el desarrollo completo en tecnologías web, esto lleva a una gran ventaja, la similitud entre escribir páginas web y escribir aplicaciones móviles, sin embargo, esta abstracción y facilidad conlleva grandes penalizaciones para aplicaciones con altas necesidades computacionales, entre ellas encontramos:

1. Una penalización en el uso de caches y un bajo control del uso de ésta.
2. Bajo control en el uso de hilos de ejecución.
3. Algunas soluciones insertan un visor web como vista, con un bajo rendimiento.

No hay datos exactos sobre la penalización en el rendimiento ni Benchmarks, pero la gente que tiene experiencia con estas herramientas indican que un desarrollo nativo es lo más indicado en muchos casos. [1, 10, 34]

Por lo que, el desarrollo de *Skywalker* mediante estas tecnologías, queda completamente descartado, debido a la necesidad de un rendimiento excelente.

2.3.3. Desarrollo nativo

La última opción considerada es el desarrollo nativo a cada plataforma, esta es la única opción que proporciona un alto control sobre la ejecución, y un acceso pleno a las distintas APIs, a costa de mantener dos ramas separadas de código fuente. No sólo los distintos lenguajes, *Java* y *Swift* están diferenciados, sino que las distintas arquitecturas de cada plataforma objetivo, y las guías de diseño, tanto de código, diseño visual, y arquitectonico que *Apple* y *Google* aconsejan y aprueban tendrán un fuerte impacto en el diseño y desarrollo.

Sin embargo, aunque parezcan mayores las desventajas, esta es la opción elegida, debido a que es la única libre de pagos, y libre de restricciones de rendimiento y con pleno aprovechamiento de las capacidades que ofrece cada plataforma objetivo.

Figura 2.3: Ventajas del desarrollo nativo e híbrido [1]

Capítulo 3

Análisis

3.1. Requisitos

El proceso de elicitación de requisitos arroja la siguiente lista de requisitos:

3.1.1. Requisitos funcionales

RF-01: Selección de elementos

El sistema permitirá la selección de elementos de interés para el usuario.

RF-02: Visualización de elementos

El sistema deberá permitir al usuario localizar cualquier objeto de interés, mostrando identificador, piso y distancia entre el usuario y dicho punto.

RF-03: Ayuda a la orientación

El sistema deberá indicar al usuario hacia donde ha de dirigir el dispositivo para la localización de cada uno de los objetos que se desee visualizar.

RF-04: Visualización de la cámara trasera

El sistema deberá utilizar la imagen de la cámara trasera como fondo para la información mostrada.

RF-05: Conexión externa

El sistema deberá permitir al usuario conectarse a un servidor en red para recibir información sobre los distintos elementos localizables.

RF-06: Localización del usuario

El sistema deberá utilizar el dispositivo como sistema de localización del usuario, transmitiendo el identificador asociado al usuario que utilize el sistema en dicho momento al servidor conectado.

3.1.2. Requisitos de información

RI-01: Posicionamiento

Por cada punto disponible, incluido el usuario:

- Identificador
- Nombre
- Posición

RI-02: Centro

Por cada centro:

- Identificador
- Escala del mapa
- Norte mágnetico

3.1.3. Requisitos no funcionales

RNF-01: Disponibilidad Android

El sistema deberá funcionar en dispositivos Android 4.3 o superior, nivel de API 18 y que cuenten con cámara trasera, sensores de rotación y bluetooth LE.

RNF-02: Disponibilidad iOS

El sistema deberá funcionar en dispositivos iOS 8.0 o superior, y que cuenten con cámara trasera, sensores de rotación y bluetooth LE.

RNF-03: Tipo de conexión

El sistema deberá utilizar una conexión de red, mediante HTTPS y peticiones tipo REST, para comunicarse con el servidor.

RNF-04: Transmisión de posición

El sistema deberá transmitir su localización e identificador a antenas receptoras de paquetes bluetooth LE.

3.2. Casos de uso

Los casos de uso que *Skywalker* ha de cumplir son escasos, aunque no por ello su dificultad sea menor.

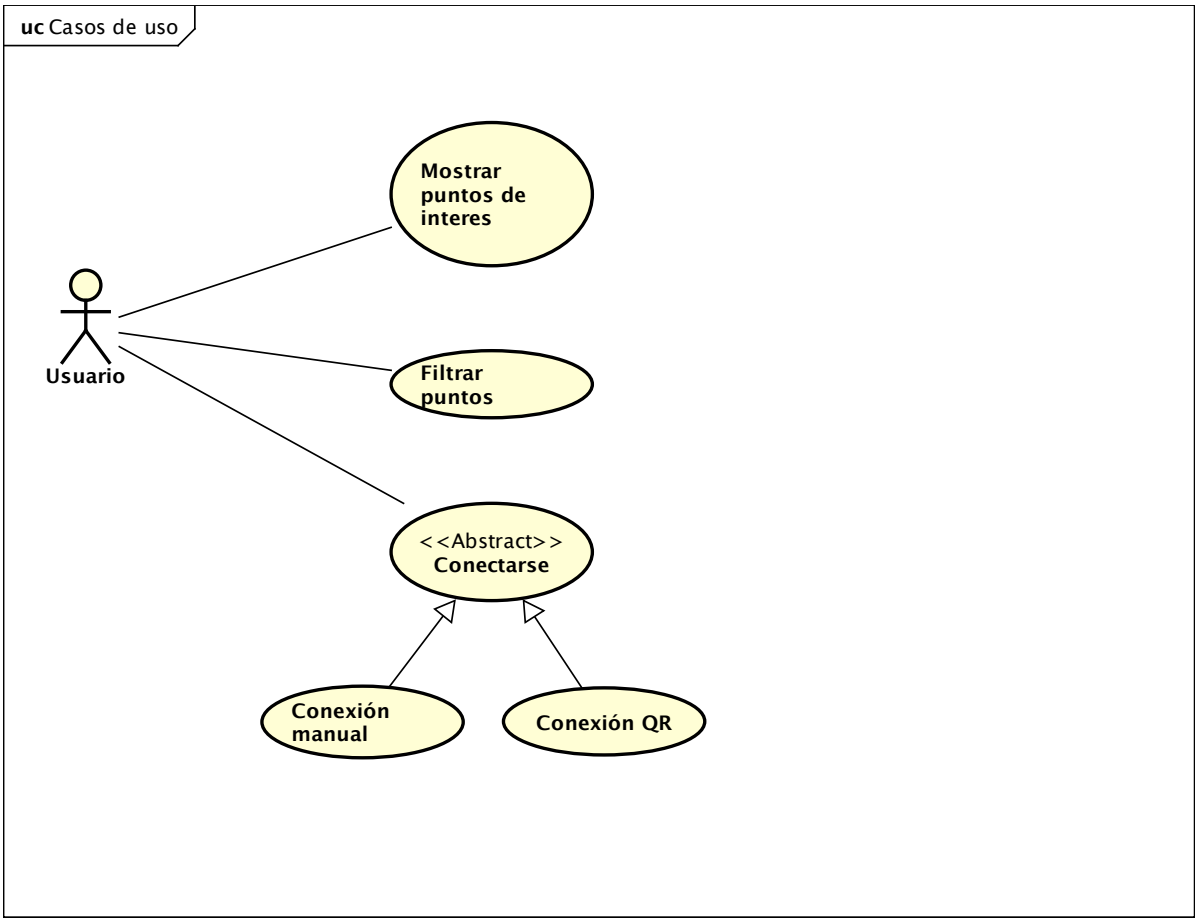


Figura 3.1: Casos de uso detectados.

3.2.1. Mostrar puntos

ITEM	VALUE
UseCase	Mostrar puntos de interes
Summary	Detectar puntos de interes a traves de la vision de realidad aumentada.
Actor	Usuario
Precondition	El usuario se ha conectado al servidor remoto.
Base Sequence	1 - El usuario indica que quiere observar los puntos de interes. 2 - El sistema muestra la visión de la camara trasera, a su vez, muestra superpuestos los puntos de interes registrados como observables, en su posición correspondiente en pantalla, y diferenciando los elemento s que entran en el campo de visión de la cámara, de aquellos que no en tran en dicho campo de visión, en un bucle infinito.
Exception Sequence	2 - El usuario cancela, el caso de uso termina.

Figura 3.2: Descripción del caso de uso mostrar elementos.

3.2.2. Filtrar puntos

ITEM	VALUE
UseCase	Filtrar puntos
Summary	Seleccionar y filtrar los puntos a observar.
Actor	Usuario
Precondition	El usuario se ha conectado a un servidor.
Postcondition	Los puntos de interes a observar han quedado registrados en el sistem a.
Base Sequence	1 - El usuario indica que quiere seleccionar los puntos de interes a obs ervar. 2 - El sistema muestra una lista de todos los puntos de interes observ ables para dicho usuario. 3 - El usuario selecciona los elementos a observar y acepta. 4 - El sistema guarda los puntos de interes a mostrar.
Exception Sequence	3 - El usuario cancela, el caso de uso queda sin efecto.

Figura 3.3: Descripción del caso de uso filtrar elementos.

3.2.3. Conectarse

ITEM	VALUE
UseCase	[Abstract]Conectarse
Summary	Conectarse a una base de datos para obtener la información de los puntos de interes a observar.
Actor	Usuario
Postcondition	El sistema registra la conexión con el servidor.
Base Sequence	1 - El usuario desea iniciar una nueva conexión manual
Branch Sequence	1' - El usuario desea iniciar una nueva conexión mediante código QR

Figura 3.4: Descripción del caso de uso iniciar nueva conexión.

3.2.4. Conexión manual

ITEM	VALUE
UseCase	Conexión manual
Summary	Conectarse a una base de datos para obtener la información de los puntos de interes a observar.
Postcondition	El sistema registra la conexión con el servidor.
Base Sequence	1 - El usuario indica que quiere realizar una nueva conexión de forma manual. 2 - El sistema solicita al usuario la información de acceso al servidor. 3 - El usuario introduce sus credenciales. 4 - El sistema comprueba las credenciales y conecta el servidor.
Exception Sequence	3 - El usuario cancela, el caso de uso termina. 4 - Las credenciales son incorrectas, el sistema muestra un mensaje de error y el caso de uso continua en el paso 2.

Figura 3.5: Descripción del caso de uso iniciar nueva conexión manual.

3.2.5. Conexión QR

ITEM	VALUE
UseCase	Conexión QR
Summary	Conectarse a una base de datos para obtener la información de los puntos de interes a observar.
Postcondition	El sistema registra la conexión con el servidor.
Base Sequence	1 - El usuario indica que quiere realizar una nueva conexión mediante QR. 2 - El sistema muestra la visión de la camara trasera. 3 - El usuario centra un código QR en la visión de la camara. 4 - El sistema escanea dicho código y conecta con el servidor.
Exception Sequence	3 - El usuario cancela, el caso de uso termina. 4 - El código QR no contiene información correspondiente a un servidor valido, el sistema muestra un mensaje de error y el caso de uso continua en el paso 2.

Figura 3.6: Descripción del caso de uso iniciar nueva conexión mediante código QR.

3.3. Modelo de dominio

El análisis del dominio lleva al siguiente diagrama:

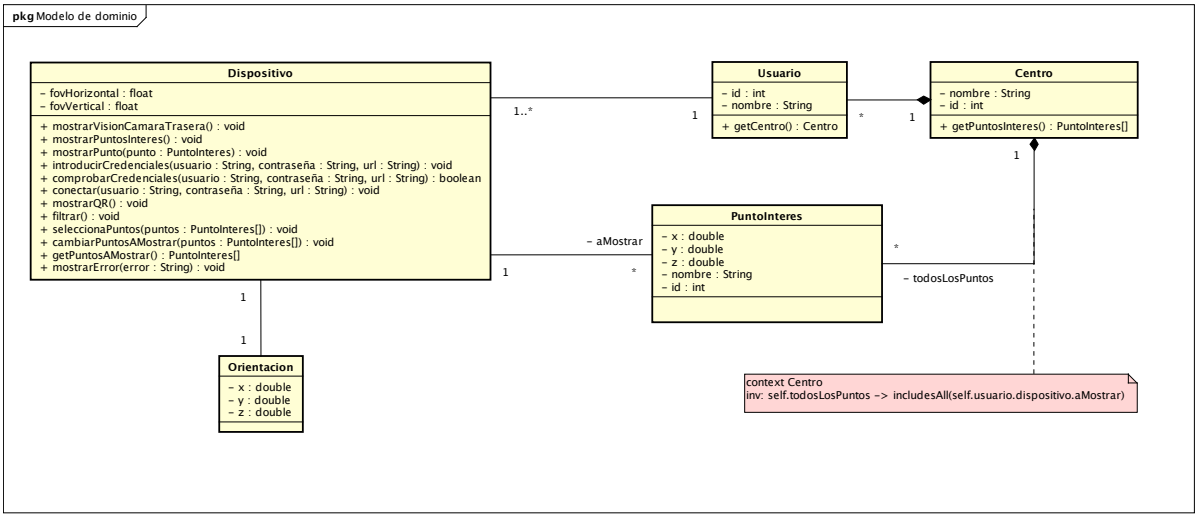


Figura 3.7: Modelo de dominio de Skywalker.

3.4. Diagramas de secuencia

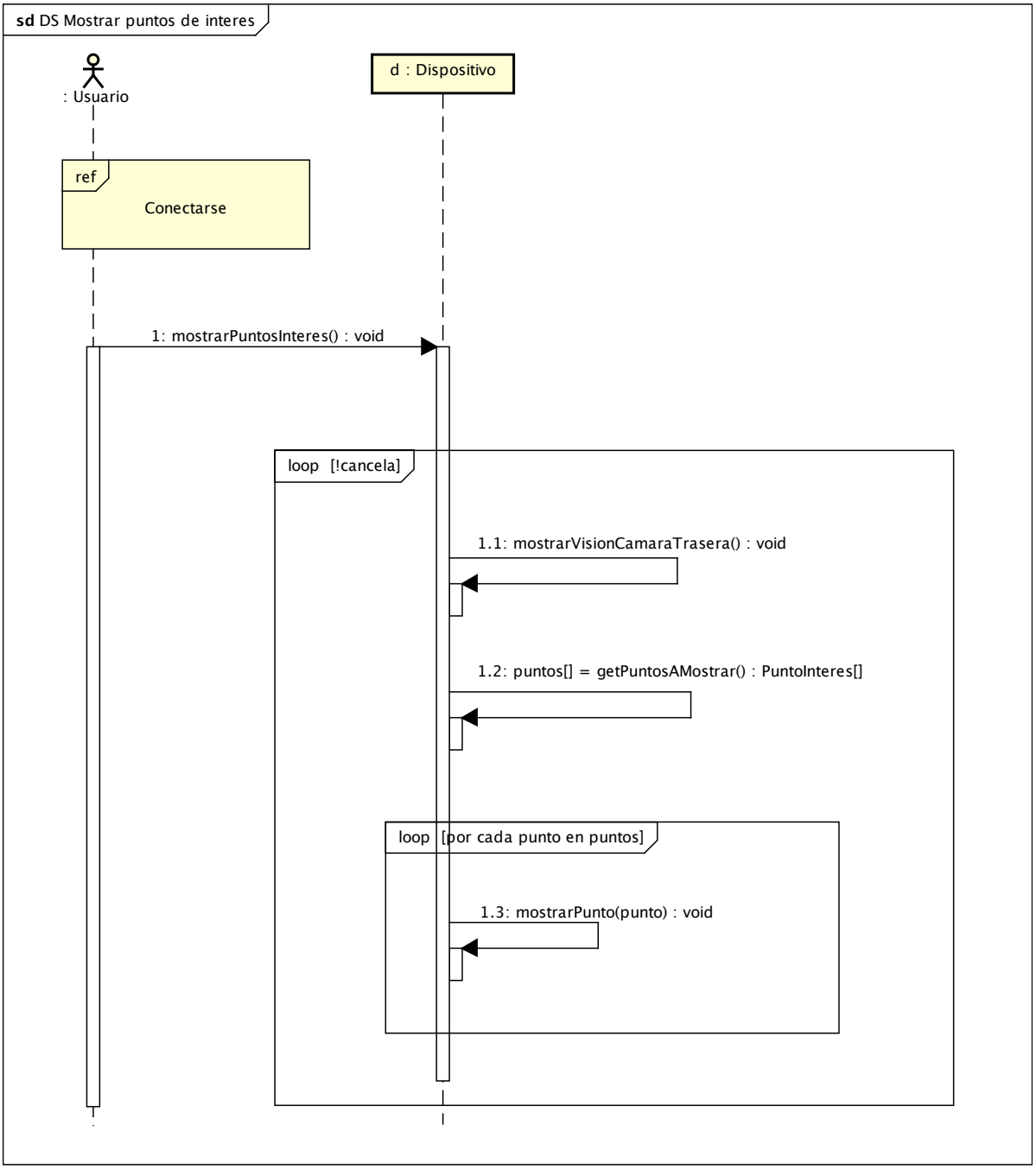


Figura 3.8: Diagrama de secuencia para el caso de uso mostrar puntos.

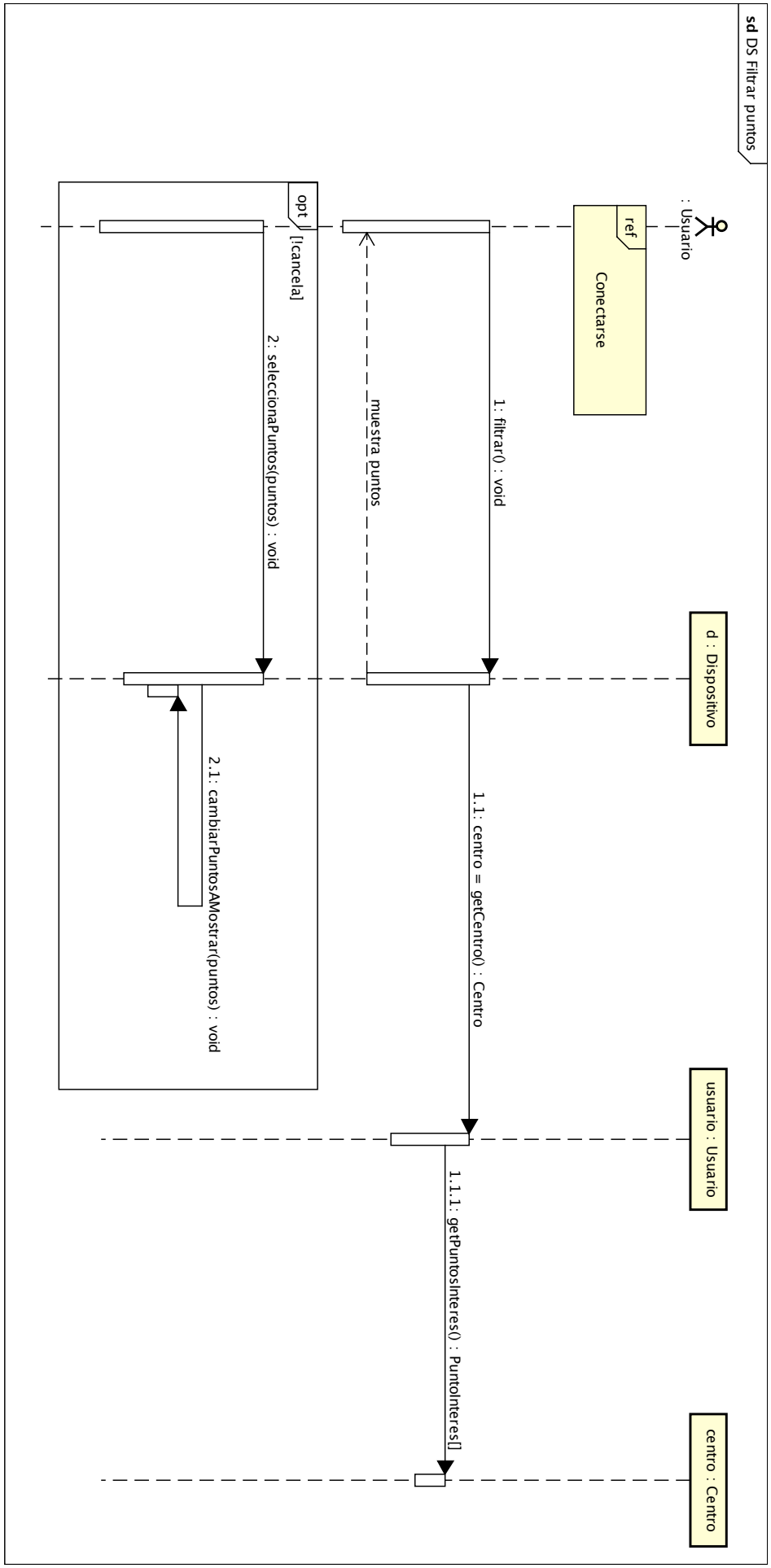


Figura 3.9: Diagrama de secuencia para el caso de uso filtrar elementos.

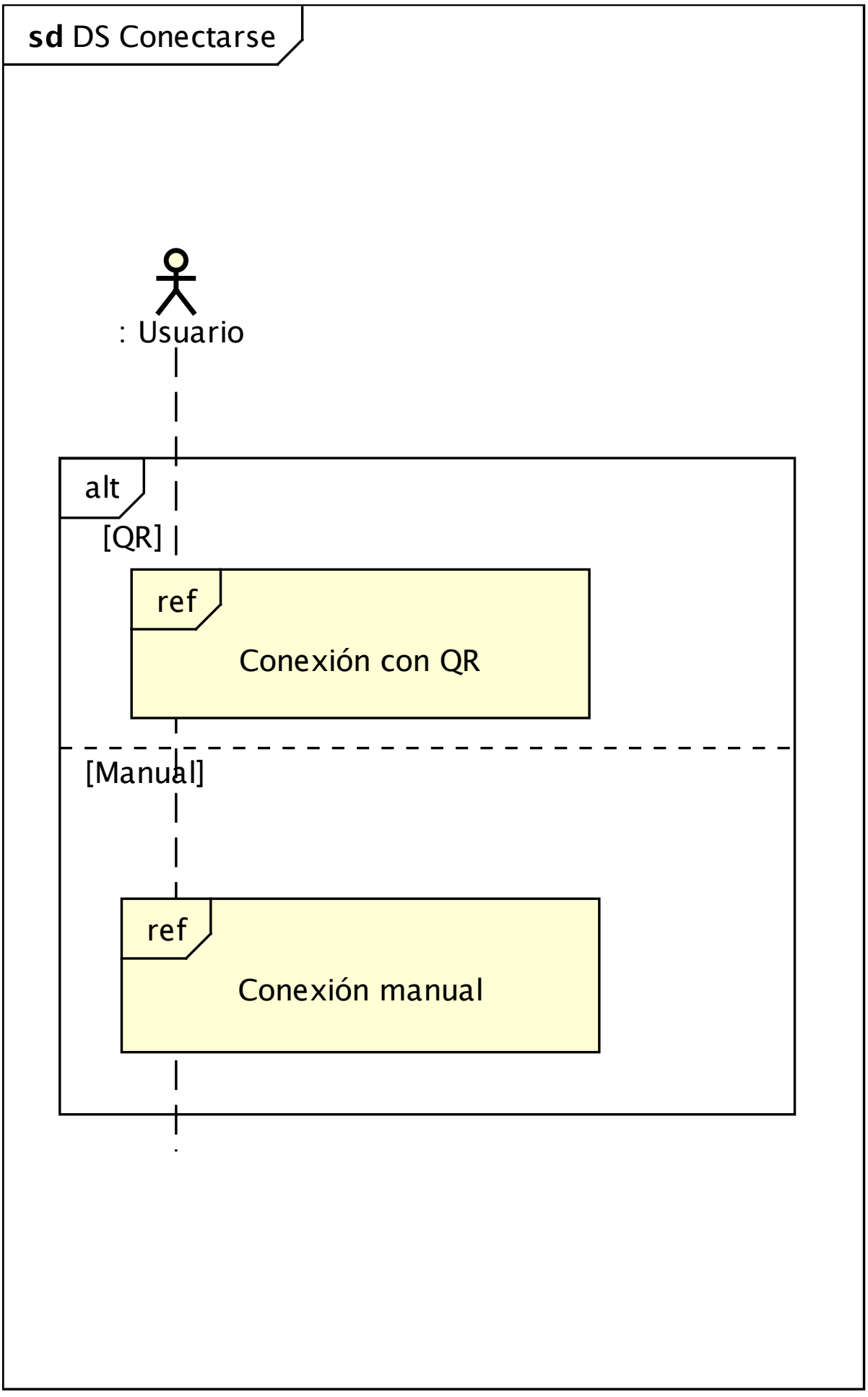


Figura 3.10: Diagrama de secuencia para el caso de uso conectarse.

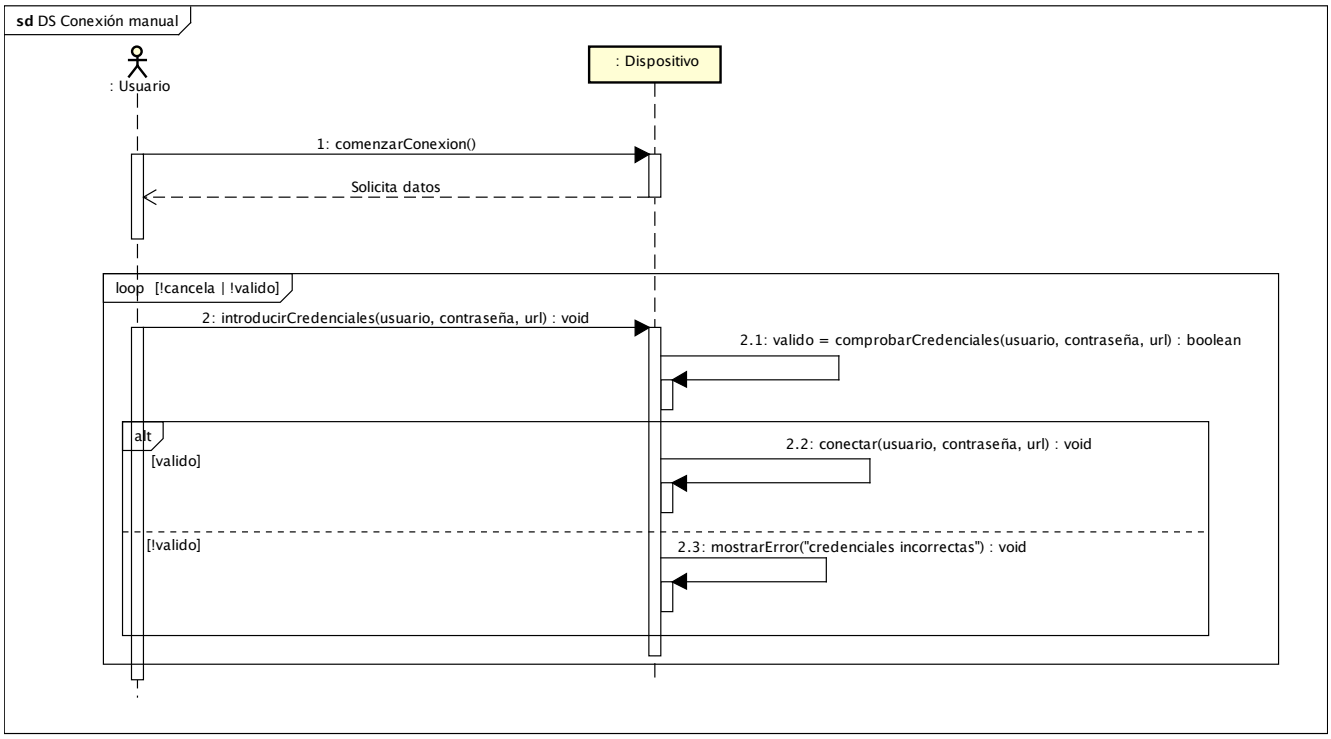


Figura 3.11: Diagrama de secuencia para el caso de uso nueva conexión manual.

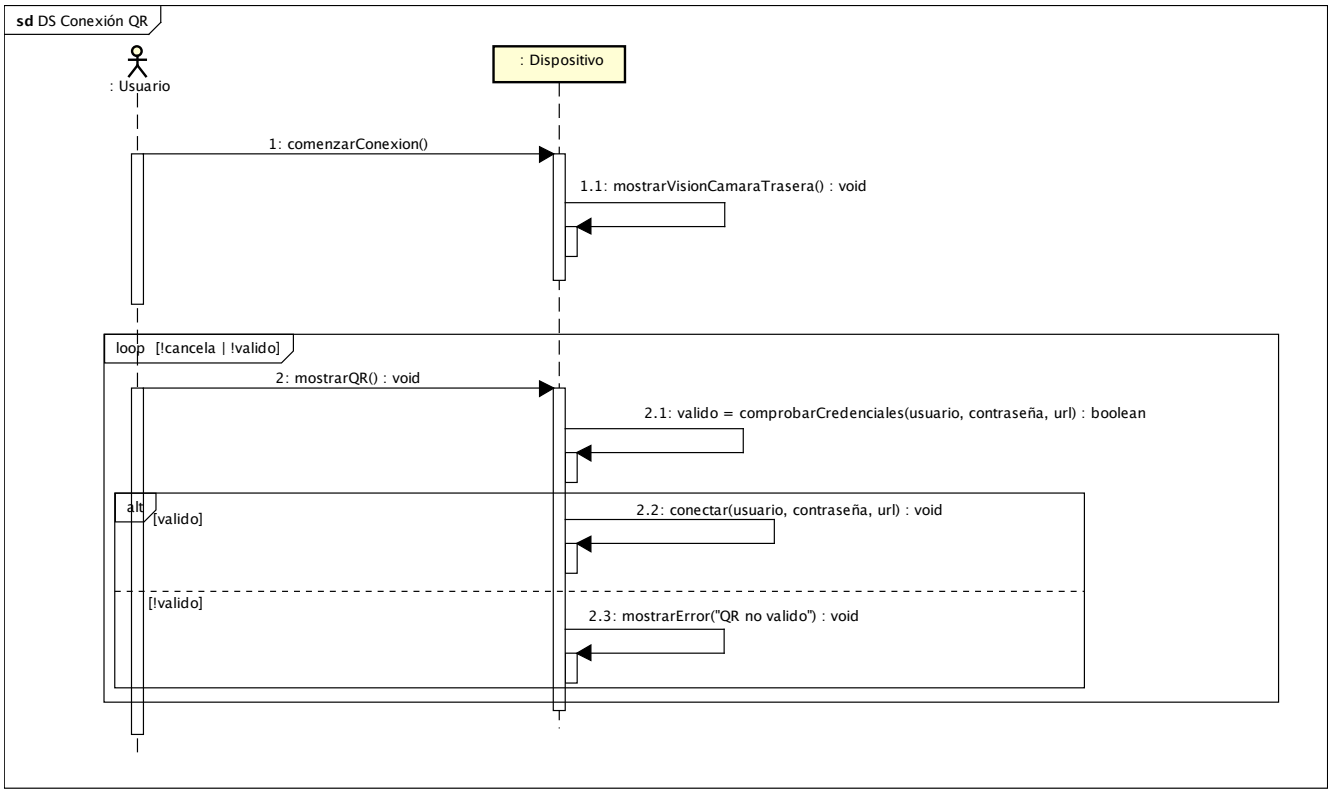


Figura 3.12: Diagrama de secuencia para el caso de uso nueva conexión mediante código QR.

3.5. Prototipado

El prototipado inicial de la interfaz, cumpliendo los requisitos y los casos de uso, ha llevado a la generación de los siguientes mockups, es importante tener en cuenta que estas imágenes representan interfaces genéricas, por lo que la implementación real de estas interfaces de usuario dependerán de las guías de estilo y diseño de cada plataforma, incluso viéndose modificadas según avance el desarrollo del sistema.

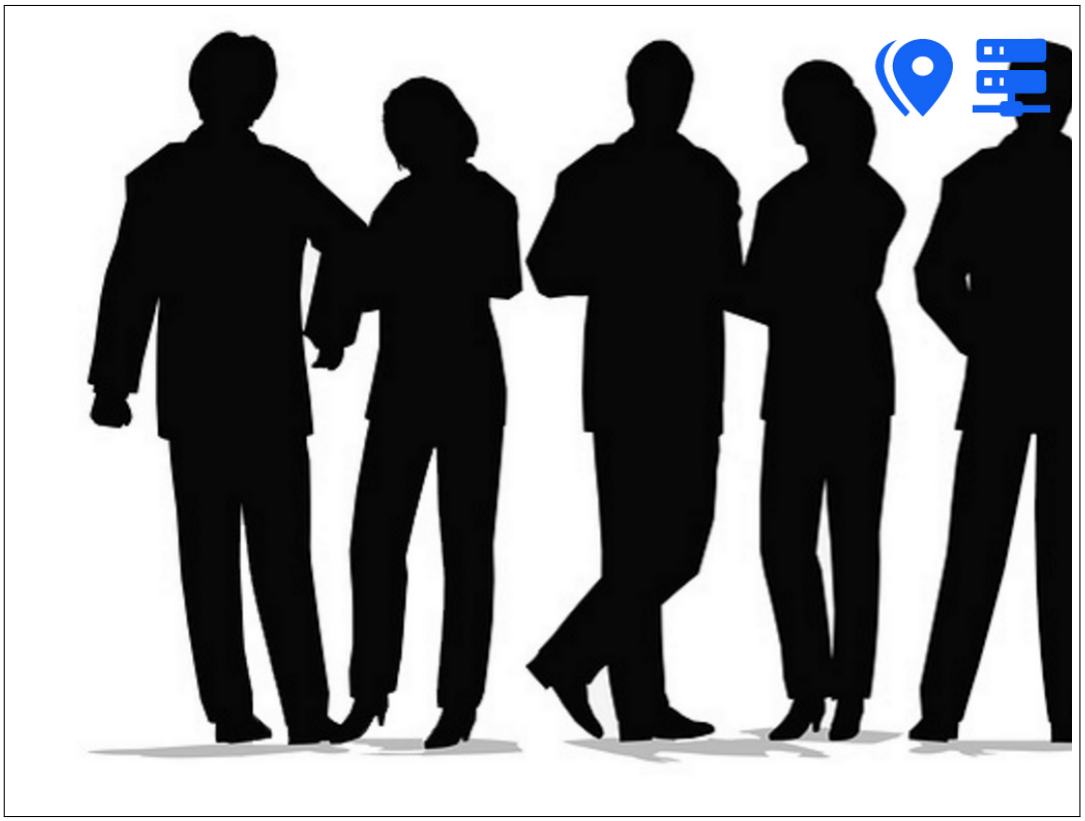


Figura 3.13: UI con botones.

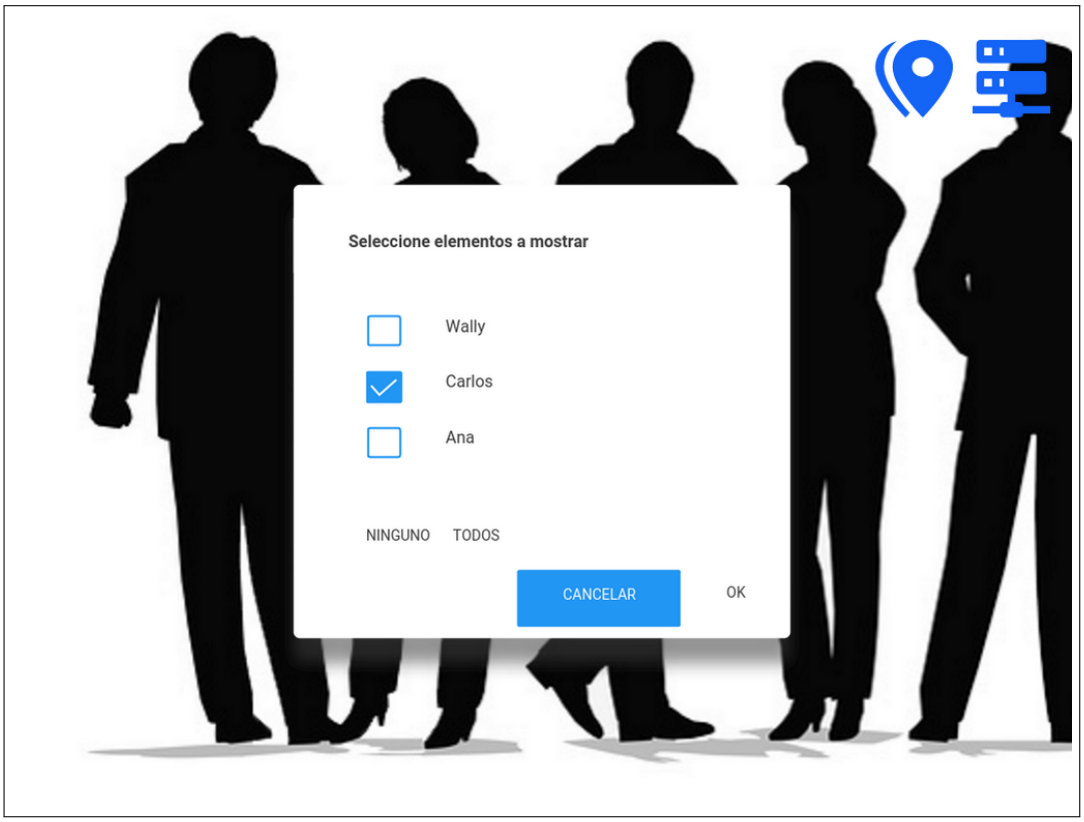


Figura 3.14: Dialogo de filtrado.

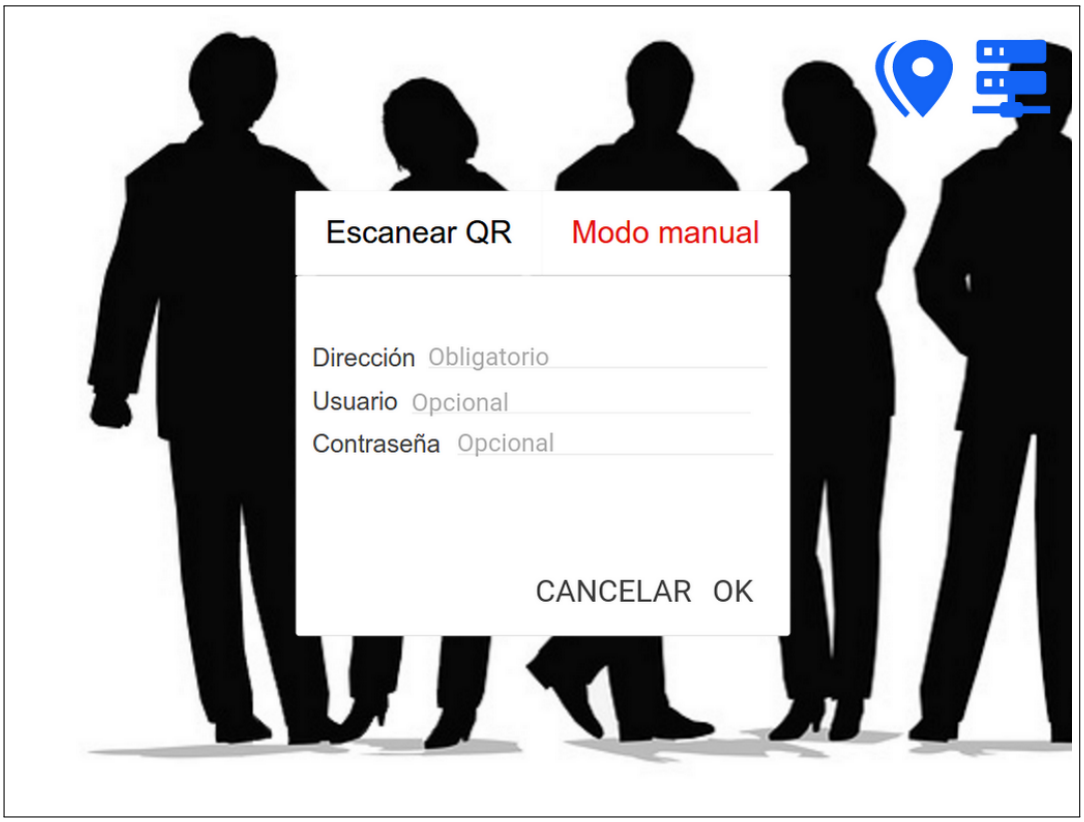


Figura 3.15: Dialogo de conexión con una BB.DD. de forma manual.

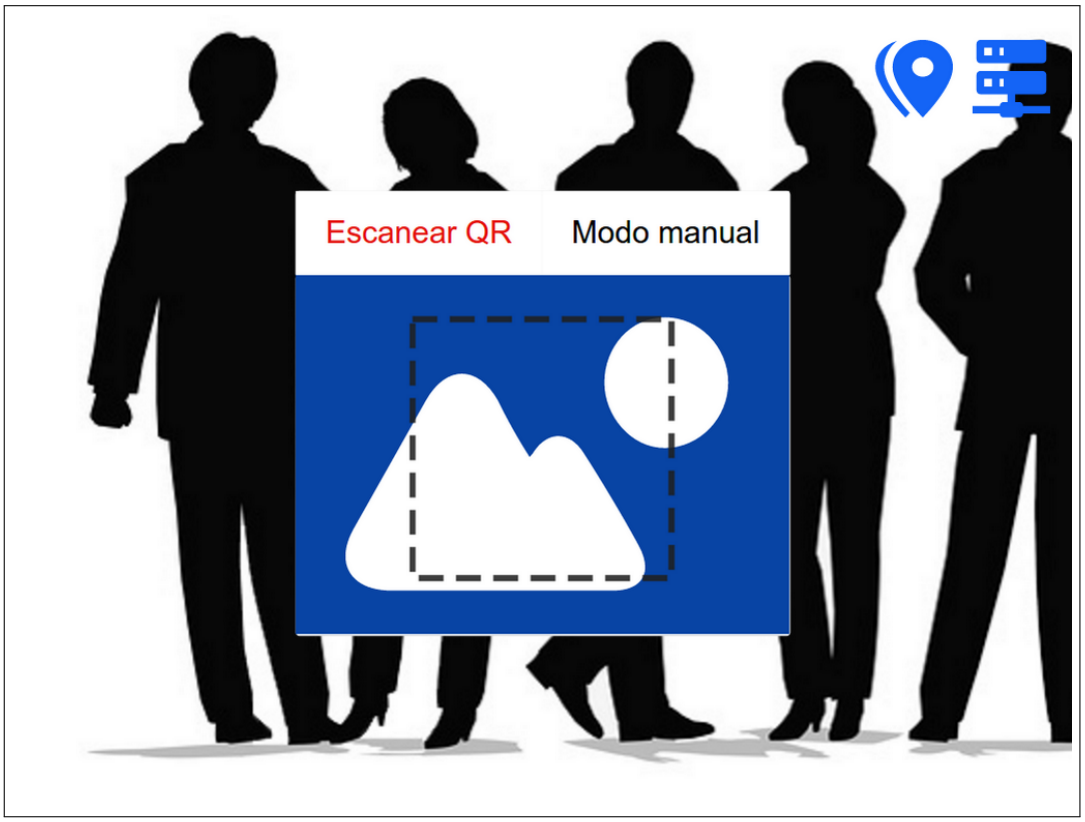


Figura 3.16: Dialogo de conexión con una BB.DD. mediante código QR.

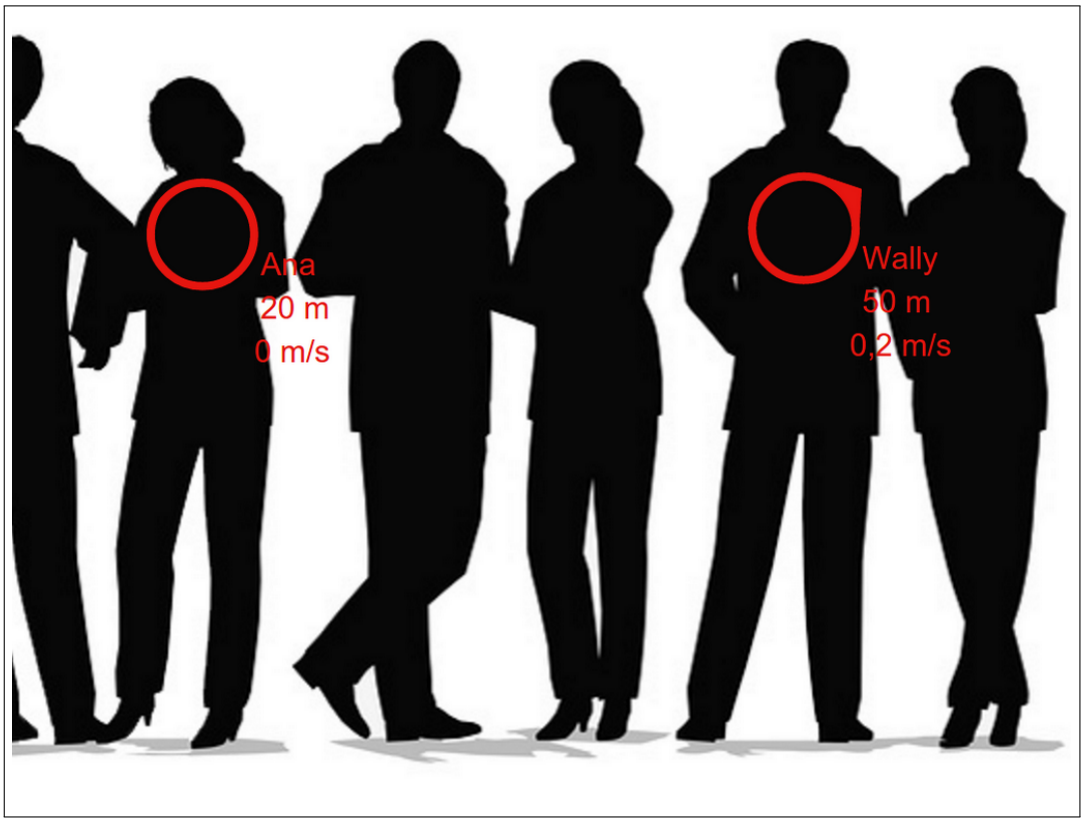


Figura 3.17: Identificación de los elementos dentro del campo de visión mediante círculos.

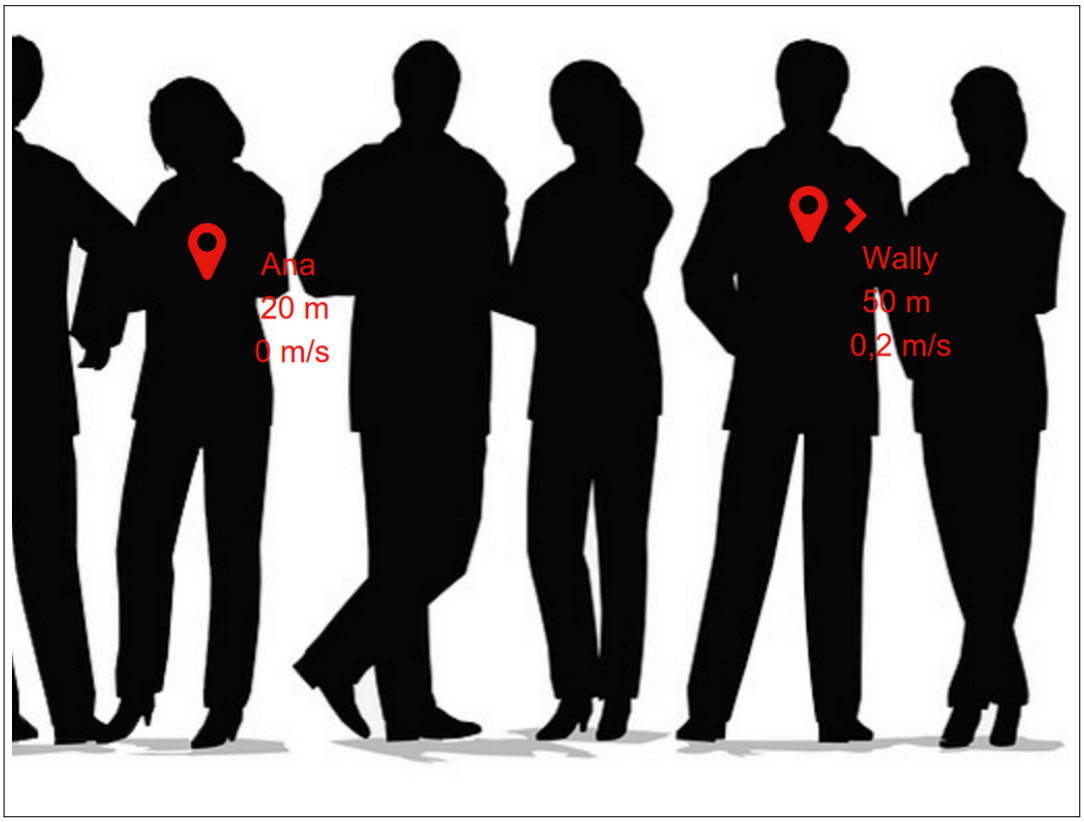


Figura 3.18: Identificación de los elementos dentro del campo de visión mediante círculos.

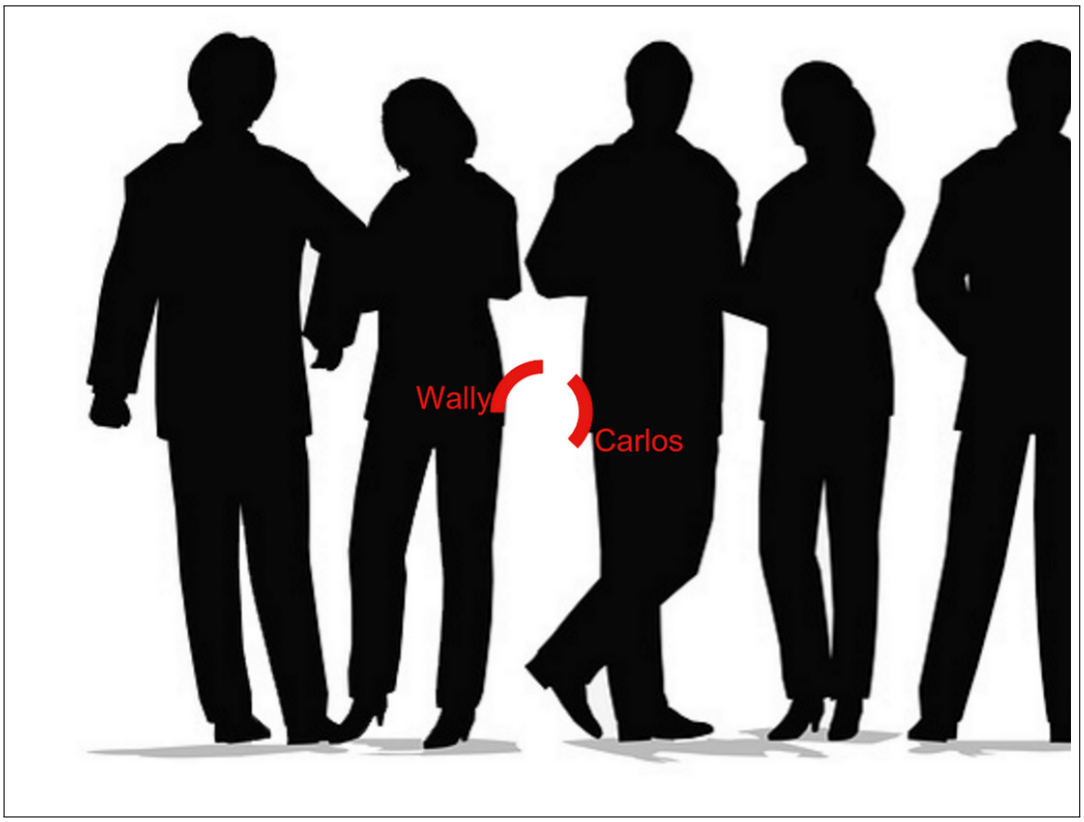


Figura 3.19: Identificación de elementos fuera de vista mediante arcos centrales a la UI.

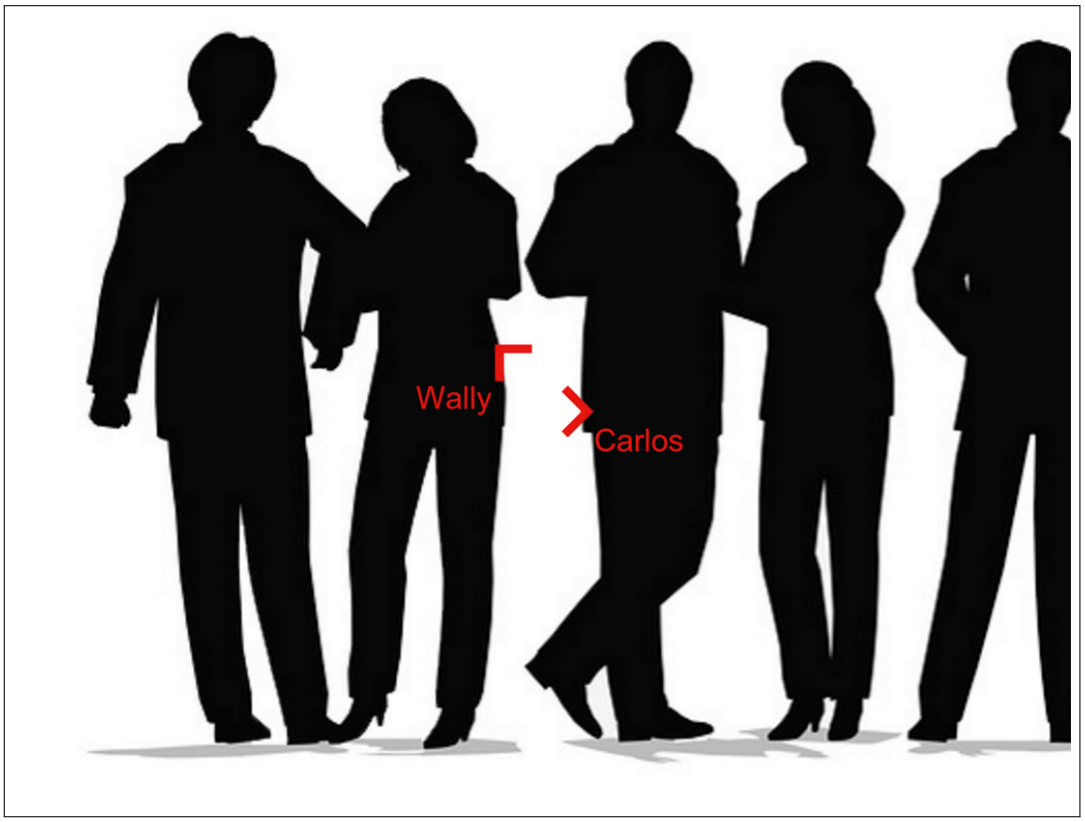


Figura 3.20: Identificación de elementos fuera de vista mediante iconos centrales a la UI.

Figura 3.21: Identificación de elementos fuera de vista mediante lineas circundantes a la UI.

Figura 3.22: Identificación de elementos fuera de vista mediante iconos circundantes a la UI.

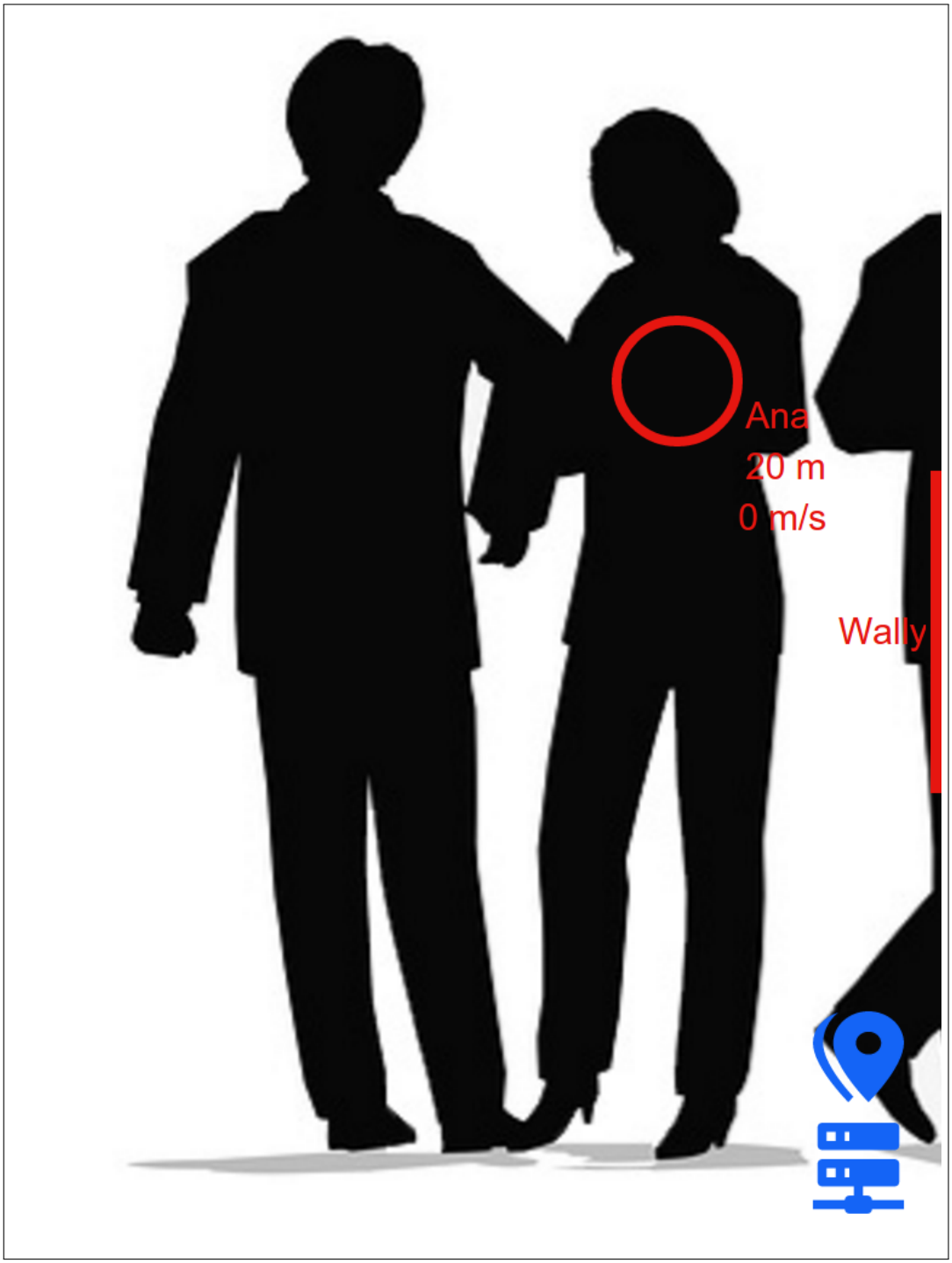


Figura 3.23: La interfaz simplemente rota si el dispositivo cambia de estado.

Capítulo 4

Planificación

4.1. Alcance de proyecto

El objetivo final del desarrollo aquí propuesto es la obtención de dos aplicaciones móviles *Skywalker*, una para sistemas *iOS* y otra para sistemas *Android*, que permitan la detección de objetos, personas y lugares de interés mediante una visión de realidad aumentada, usando *XtremeLoc* como sistema de apoyo. La aplicación debe ser compatible, tanto con tabletas como con móviles, y deberá cumplir los requisitos detallados en profundidad en la Sección 3.1.

4.2. Metodología

Para favorecer un desarrollo proactivo y facilitar la evolución de la App durante su desarrollo, además de la necesidad de arquitecturas emergentes debido a la inexperiencia con los Frameworks de desarrollo, se necesitará un enfoque de desarrollo ágil, entre las distintas opciones disponibles, la metodología que más se acomoda a las características del desarrollo es Scrum.
Sin embargo, Scrum está pensado para equipos de 7 ± 2 personas con ciertos roles bien definidos en personas distintas, por lo que se adaptará Scrum a las propiedades del desarrollo, adoptando así la metodología llamada Personal Scrum.

Los roles serán asignados de la siguiente forma:

Product Owner Representante de la voz del cliente, en este caso, uno de los tutores de *Skywalker*, Diego R. Llanos.

Scrum Master Cuyo trabajo consiste en eliminar los obstáculos que impiden alcanzar los objetivos del Sprint, rol asumido por mi, Héctor Del Campo.

Equipo de desarrollo Encargado de entregar el producto, y formado por una única persona, yo, Héctor Del Campo.

Caracterizado por los siguientes eventos:

Sprints Eventos con una duración de entre 1 y 4 semanas, durante los cuales se realizará el desarrollo de las historias de usuario elegidas del *Product Backlog*, para dicho intervalo de tiempo.

ID	Historia de usuario	Prioridad	Estimación (H/M)
0	Como usuario quiero ver la visión de la cámara trasera.	1	105
1	Como usuario quiero ver los elementos superpuestos sobre la visión de la cámara trasera.	2	210
2	Como usuario quiero poder elegir que elementos ver en la visión de realidad aumentada.	3	70
3	Como usuario quiero poder conectarme a un servidor para recibir información de los puntos de interes.	4	105
4	Como usuario quiero poder utilizar mi dispositivo móvil como dispositivo de transmisión de mi localización.	5	70

Cuadro 4.1: Product Backlog de *Skywalker*

Sprint planning, Sprint review y Sprint retrospective Los cuales serán realizados en presencia del tutor Diego R. Llanos, para evaluar el progreso del desarrollo y planificar las siguientes semanas.

Daily Scrum Este evento no existe como tal, al iniciar una jornada decidiré personalmente como planificar mi trabajo para ese día.

El resto de elementos se mantendrán intactos respecto de la metodología Scrum original.

El cuadro 4.1 muestra el *Product Backlog*, con las historias de usuario que habrá que haber cumplido al finalizar el desarrollo.

4.3. Plan temporal

Cada una de las siguientes tablas muestran los *Sprints Backlogs*, con el trabajo llevado a cabo en cada uno de estos *Sprints* durante el desarrollo. La duración de estos sprints se ha calculado teniendo en cuenta el trabajo a realizar, su complejidad y la experiencia personal, además, se ha incluido un trabajo previo a realiza, cuyo objetivo es adquirir las siguientes capacidades:

- Conocimiento de lenguaje Java 7.
- Conocimiento de lenguaje Swift 3.
- Conocimiento del Framework Android.
- Conocimiento del Framework iOS.
- Conocimiento matemático, en trigonometría y orientación espacial.
- Capacidades relativas a la automatización de pruebas.
- Conocimiento básico de la filosofía REST.

Aparte de este trabajo inicial, se ha incluido un tiempo extra final para la generación de este informe.

A continuación se incluyen los *Sprints* a realizar, habiendo considerado un trabajo de 5 horas diarias, los diagramas de Gantt con la calendarización de dichos *Sprints*, y los objetivos necesarios a cubrir en el trabajo previo.

Historia de usuario	SO	Tarea	Estimación (H/M)
0	Android	Diseño cámara abstracta	4
0	Android	Crear layout	1
0	Android	Programar cámara API menor a 21	30
0	Android	Programar cámara API 21 o mayor	60
0	iOS	Diseño	2
0	iOS	Crear layout	1
0	iOS	Programar cámara	7

Cuadro 4.2: 1^{er} Sprint Backlog - 3 semanas

Historia de usuario	SO	Tarea	Estimación (H/M)
1	Android	Diseño sensor	10
1	Android	Programar sensor orientación	25
1	Android	Diseñar dibujado de elementos	5
1	Android	Crear layout	1
1	Android	Programar clases de soporte matemáticas	15
1	Android	Programar clases del dominio	15
1	Android	Programar dibujado de elementos	25
1	iOS	Diseño sensor	10
1	iOS	Programar sensor orientación	25
1	iOS	Diseñar dibujado de elementos	10
1	iOS	Crear layout	1
1	iOS	Programar clases de soporte matemáticas	15
1	iOS	Programar clases del dominio	15
1	iOS	Programar dibujado de elementos	38

Cuadro 4.3: 2º Sprint Backlog - 6 semanas

Historia de usuario	SO	Tarea	Estimación (H/M)
2	Android	Diseño del filtrado de elementos	4
2	Android	Crear layout	1
2	Android	Implementación del filtrado de elementos	30
2	iOS	Diseño del filtrado de elementos	4
2	iOS	Crear layout	1
2	iOS	Implementación del filtrado de elementos	30

Cuadro 4.4: 3^{er} Sprint Backlog - 2 semanas

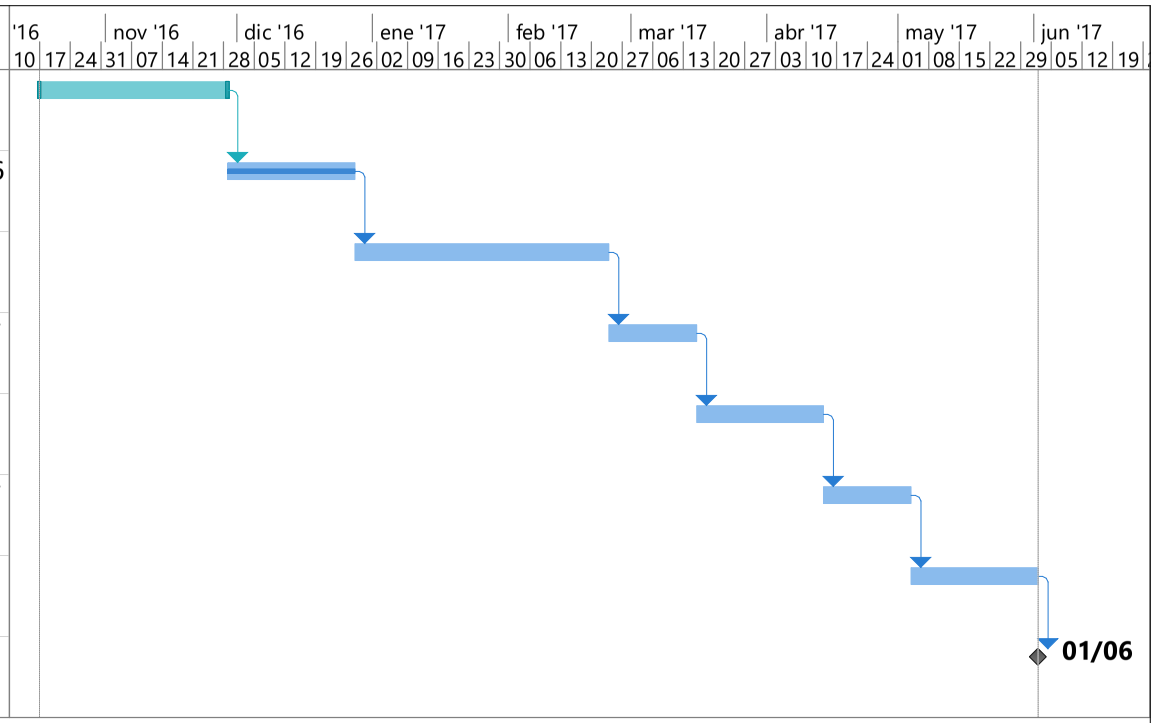
Historia de usuario	SO	Tarea	Estimación (H/M)
3	Android	Diseño de la conexión con el servidor	5
3	Android	Crear layout	5
3	Android	Implementación conexión manual	30
3	Android	Implementación conexión con código QR	15
3	iOS	Diseño de la conexión con el servidor	5
3	iOS	Crear layout	5
3	iOS	Implementación conexión manual	30
3	iOS	Implementación conexión con código QR	10

Cuadro 4.5: 4º Sprint Backlog - 3 semanas

Historia de usuario	SO	Tarea	Estimación (H/M)
4	Android	Diseño emisor bluetooth	5
4	Android	Implementación emisor bluetooth	40
4	iOS	Diseño emisor bluetooth	5
4	iOS	Implementación emisor bluetooth	20

Cuadro 4.6: 5º Sprint Backlog - 2 semanas

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Preparativos	31 días	lun 17/10/16	lun 28/11/16
2	Sprint 1	21 días	mar 29/11/1	mar 27/12/16
3	Sprint 2	42 días	mié 28/12/1	jue 23/02/17
4	Sprint 3	14 días	vie 24/02/17	mié 15/03/17
5	Sprint 4	21 días	jue 16/03/17	jue 13/04/17
6	Sprint 5	14 días	vie 14/04/17	mié 03/05/17
7	Informe	21 días	jue 04/05/17	jue 01/06/17
8	Fin	0 días	jue 01/06/17	jue 01/06/17



Proyecto: Skywalker

Tarea	Resumen inactivo	Tareas externas
División	Tarea manual	Hito externo
Hito	solo duración	Fecha límite
Resumen	Informe de resumen manual	Progreso
Resumen del proyecto	Resumen manual	Progreso manual
Tarea inactiva	solo el comienzo	
Hito inactivo	solo fin	

4.4. Presupuesto

En base al total de las horas hombres antes estimadas, junto con una suposición de coste de 17 €/hora-hombre, coste real en una empresa real consultada, y teniendo en cuenta el coste del Hardware para desarrollo, ya que *Apple* no permite el desarrollo para sus plataformas desde fuera de sus productos, y de la necesidad de dispositivos físicos, ya que los emuladores no tienen los sensores o emisores Bluetooth necesarios para la comprobación del correcto funcionamiento de *Skywalker*. Se ha realizado el presupuesto representado en la tabla 4.7.

Artefacto	Cantidad	Coste unitario (€)	Coste (€)
Trabajo (H/M)	820	17	13940
Plataforma Macintosh	1	1600	1600
Dispositivo Android	1	500	500
Dispositivo iOS	1	600	600
		Total	16640

Cuadro 4.7: Presupuesto de desarrollo para *Skywalker*

4.5. Plan de control

Aquí se detallan las acciones a realizar respecto al control en el desarrollo del proyecto.

Gestión de requisitos

Si una modificación propuesta entra a tiempo para su puesta en marcha en algún *Sprint* no será necesaria la modificación de la planificación. En el caso contrario, se introducirán nuevos *Sprints*, replanificando el calendario si fuera necesario.

Control de calendario

Como cualquier otro proceso *Scrum*, el control del cumplimiento del calendario planificado se basa en la entrega de los objetivos de cada *Sprint* a tiempo, con la aprobación del *Product Owner*.

4.6. Plan de gestión de riesgos

A continuación se exponen tanto los riesgos detectados, con una descripción de los mismos, su impacto, probabilidad, plan de protección y plan de contingencia, como la matriz de exposición mostrada en la tabla 4.8, la cual utilizaremos para obtener la exposición al riesgo.

R01 - Máquina personal averiada o perdida de datos

El desarrollo será realizado en una máquina MacBook Pro, en el caso de su avería, la continuación del desarrollo para iOS sería inviable

Impacto Catastrófico

Cuadro 4.8: Matriz Impacto/Probabilidad

Impacto\ Probabilidad	Muy Alto	Alto	Medio	Bajo	Muy Bajo
Catastrófico	Alto	Alto	Moderado	Moderado	Bajo
Crítico	Alto	Alto	Moderado	Bajo	Ninguno
Marginal	Moderado	Moderado	Bajo	Ninguno	Ninguno
Despreciable	Moderado	Bajo	Bajo	Ninguno	Ninguno

Probabilidad Muy bajo

Exposición Baja

Plan de Protección Utilizar un sistema de control de versiones, teniendo así una copia de seguridad.

Plan de Contingencia Dado que todo el trabajo estará guardado en el repositorio, se podrá continuar el trabajo de la versión Android en otras máquinas, en el caso de iOS el desarrollo debería ser detenido.

R02 - Dispositivos físicos averiados

Al igual que la máquina de desarrollo, los dispositivos de prueba también podrían averiarse.

Impacto Crítico

Probabilidad Bajo

Exposición Baja

Plan de Protección Ninguno.

Plan de Contingencia Intentar conseguir algún dispositivo físico de reemplazo, si no se pudiera, replanificar según el tiempo perdido.

R03 - Repositorio corrupto

No sería descabellado que el repositorio utilizado se corrompiera.

Impacto Crítico

Probabilidad Bajo

Exposición Baja

Plan de Protección Utilizar varios repositorios para el control de versiones.

Plan de Contingencia Utilizar aquellos repositorios disponibles, a la espera de la recuperación de aquellos corruptos.

R04 - Enfermedad

En el caso de enfermedad, podría no cumplirse la planificación establecida.

Impacto Crítico

Probabilidad Baja

Exposición Baja

Plan de Protección Ninguno.

Plan de Contingencia Evaluar el tiempo perdido, modificando el plan de trabajo.

R05 - Desconocimiento técnico de los Frameworks

Los Frameworks y plataformas de desarrollo se encuentran en cambio constante, esto podría provocar perdidas de tiempo buscando soluciones a ciertos problemas.

Impacto Marginal

Probabilidad Medio

Exposición Baja

Plan de Protección Planificar el proyecto con este hecho en mente, además de un diseño con patrones del tipo .estrategia.o "factoría abstracta".

Plan de Contingencia Evaluar el tiempo perdido, replanificando si fuera necesario.

R06 - Bajo rendimiento en la aplicación

El posible calentamiento del dispositivo, junto a operaciones computacionalmente intensivas, podría provocar una degradación en el dibujado de la interfaz.

Impacto Crítico

Probabilidad Medio

Exposición Moderado

Plan de Protección Diseñar e implementar para lograr un rendimiento óptimo.

Plan de Contingencia Utilizar las herramientas de profiling, mejorando aquellos puntos críticos, y replanificar si fuera necesario.

R07 - Lentitud en peticiones REST

La necesidad de una conexión lo suficientemente rápida podría ser un impedimento para la utilización de peticiones REST.

Impacto Marginal

Probabilidad Bajo

Exposición Bajo

Plan de Protección Mantener las peticiones lo más pequeñas posibles

Plan de Contingencia Cambiar el tipo de conexión con el servidor, replanificando el tiempo perdido.

R08 - Lentitud en el desarrollo de XtremeLoc

XtremeLoc, al ser un desarrollo paralelo a *Skywalker*, podría no entregar la funcionalidad requerida a tiempo.

Impacto Marginal

Probabilidad Medio

Exposición Baja

Plan de Protección Planificar el proyecto con este hecho en mente.

Plan de Contingencia Replanificar el proyecto, cambiando sprints que ya puedan comenzar por aquellos que necesitan la funcionalidad no disponible.

R09 - Bugs y errores en la aplicación

Es muy probable la aparición de errores y comportamientos no deseados en las aplicaciones.

Impacto Crítico

Probabilidad Alto

Exposición Alta

Plan de Protección Planificar el proyecto con este hecho en mente.

Plan de Contingencia Replanificar el proyecto, evaluando el tiempo perdido para su correción.

4.7. Planes de proceso soporte

Esta sección detalla los planes de soporte para el proyecto.

4.7.1. Plan de gestión de configuraciones

Se dispondrá de dos sistemas de control de version online para la documentación y código del proyecto.

Ambos repositorios cuentan con tecnología GIT, situados en GitHub y GitLab.

4.7.2. Revisiones de progreso del proyecto

Durante el *Daily Scrum* se analizará el cumplimiento de la planificación temporal, replanificandolo si fuera necesario.

Capítulo 5

Diseño

En este capítulo se presenta el Diseño realizado para *Skywalker*. Al ser un desarrollo software bastante tradicional, se emplearán los ya conocidos diagramas de clases y diagramas de secuencia.

Aparte de estos diagramas, se describirán otras técnicas utilizadas para el desarrollo, debido a las imposiciones que las arquitecturas de los sistemas utilizados imponen, conviene destacar el alto uso de llamadas asíncronas y de hilos de ejecución, además de las prácticas recomendadas.

5.1. Diseño global

En esta sección, se mostrarán la forma genérica de trabajo del sistema presentado, ya que, aún siendo bastante distintos en ciertos apartados, *iOS* y *Android* comparten ciertos elementos arquitectónicos fundamentales.

5.1.1. Arquitectura

Se ha elegido una arquitectura en tres capas estrictas, con el objetivo de aislar los cambios que puedan producirse en ellas. Existe una pequeña discusión sobre que es una vista y que es un controlador en *Android* y *iOS*, algunos autores indican que la vista son los layouts propiamente dichos, y los controladores las Actividades, Fragmentos y las clases descendientes de UIViewController; otros autores indican que estas clases son la vista en sí, y los controladores otra clase extra. Para este desarrollo, se ha decidido interpretar como vistas los layouts, y como controladores las clases antes citadas.

Además de tener las tres capas anteriores, se ha incluido una capa común, con utilidades de estilo matemático, tales como matrices y vectores, llamada servicios.

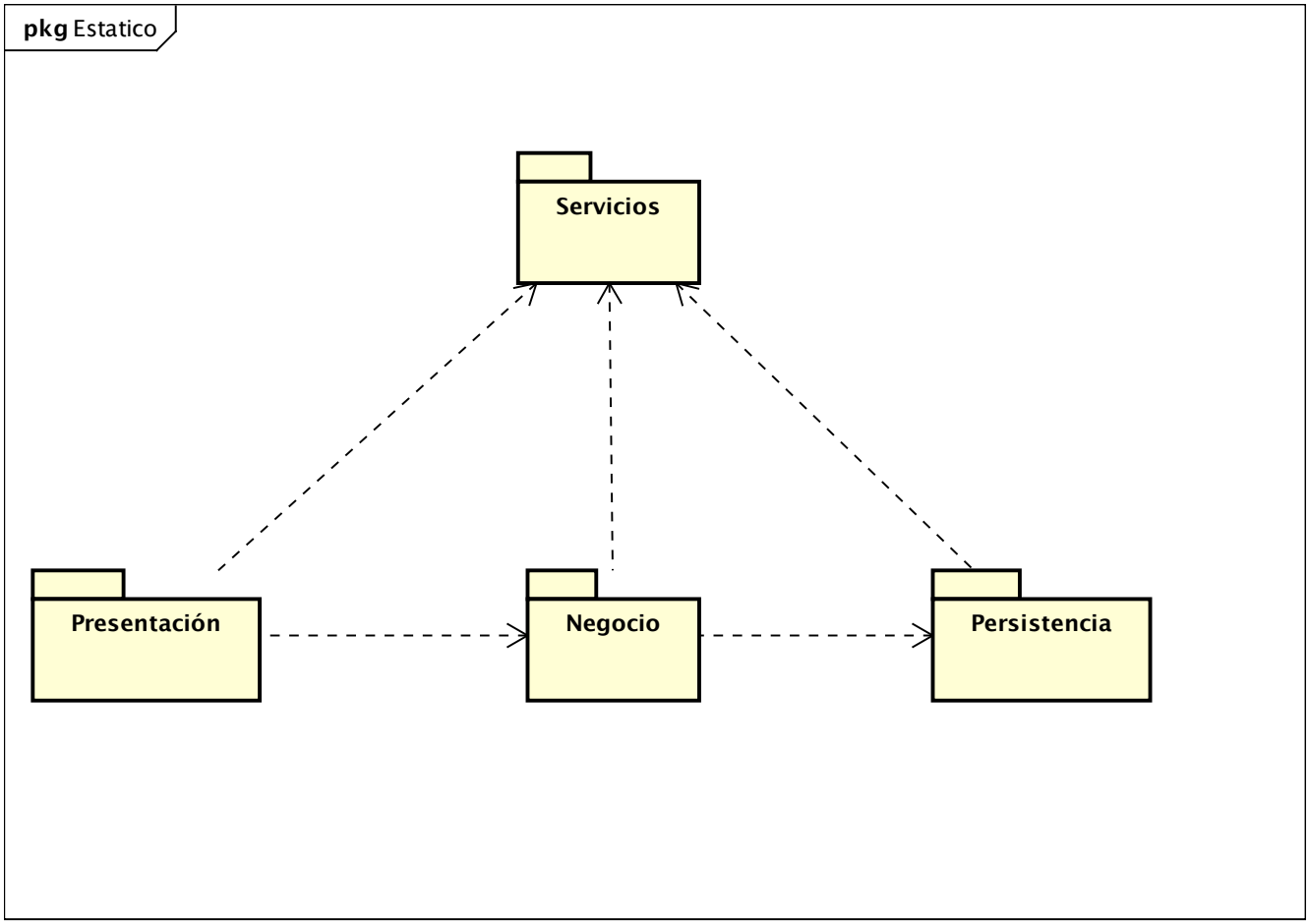


Figura 5.1: Diagrama arquitectónico común.

5.1.2. Interfaz de usuario

En ambos Frameworks de desarrollo, se promueve la reutilización y el aumento de la cohesión de las clases controladoras de la interfaz de usuario ya sea mediante *Fragmentos* en *Android* [12], o mediante *Contenedores* en *iOS* [7].

Aprovechando esta posibilidad, se ha diseñado la interfaz de *Skywalker* para no solo reutilizar elementos, sino para estar preparada ante nuevas vistas que puedan reutilizar aquellas ya creadas. Con esta idea en mente, se han extraído los siguientes elementos comunes:

1. Cada una de las páginas utilizadas en la introducción a la aplicación.
2. La interfaz de filtrado.
3. La interfaz de conexión manual.
4. La interfaz de conexión mediante código QR.
5. La interfaz de realidad aumentada, sin los controles correspondientes.
6. Los controles para la interfaz de realidad aumentada.

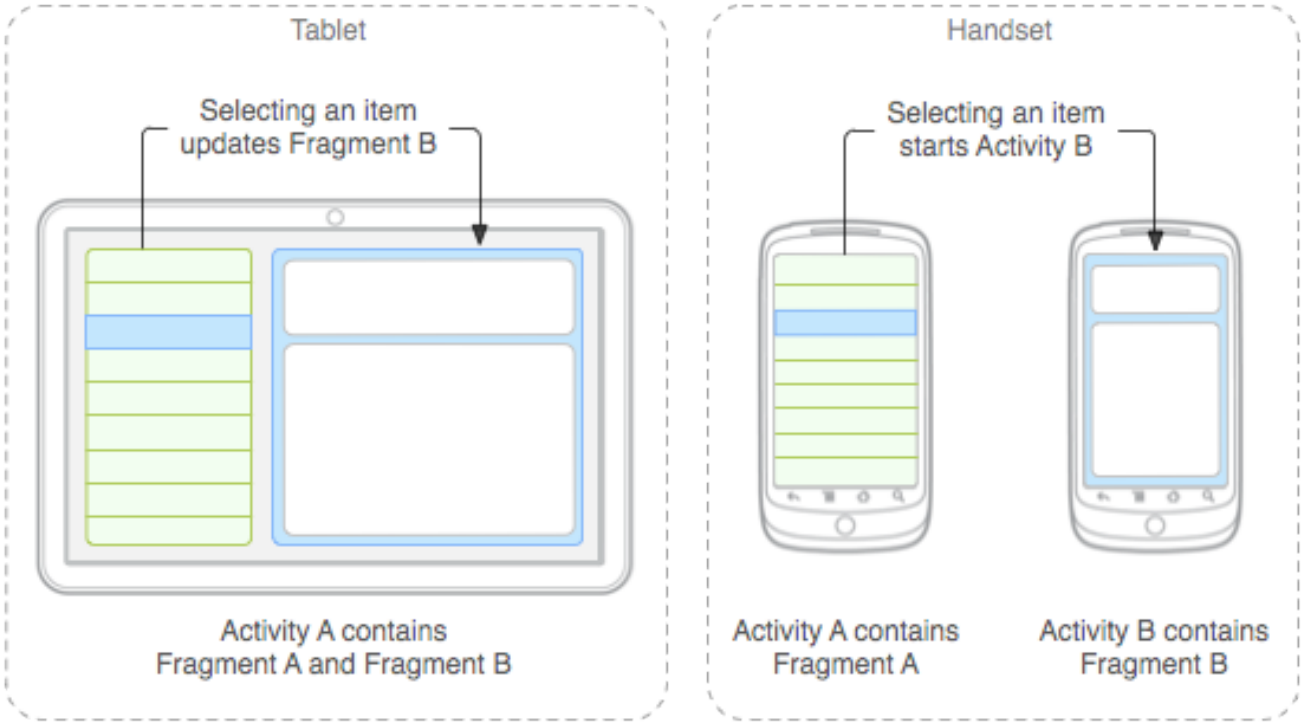


Figura 5.2: Fragmentos Android

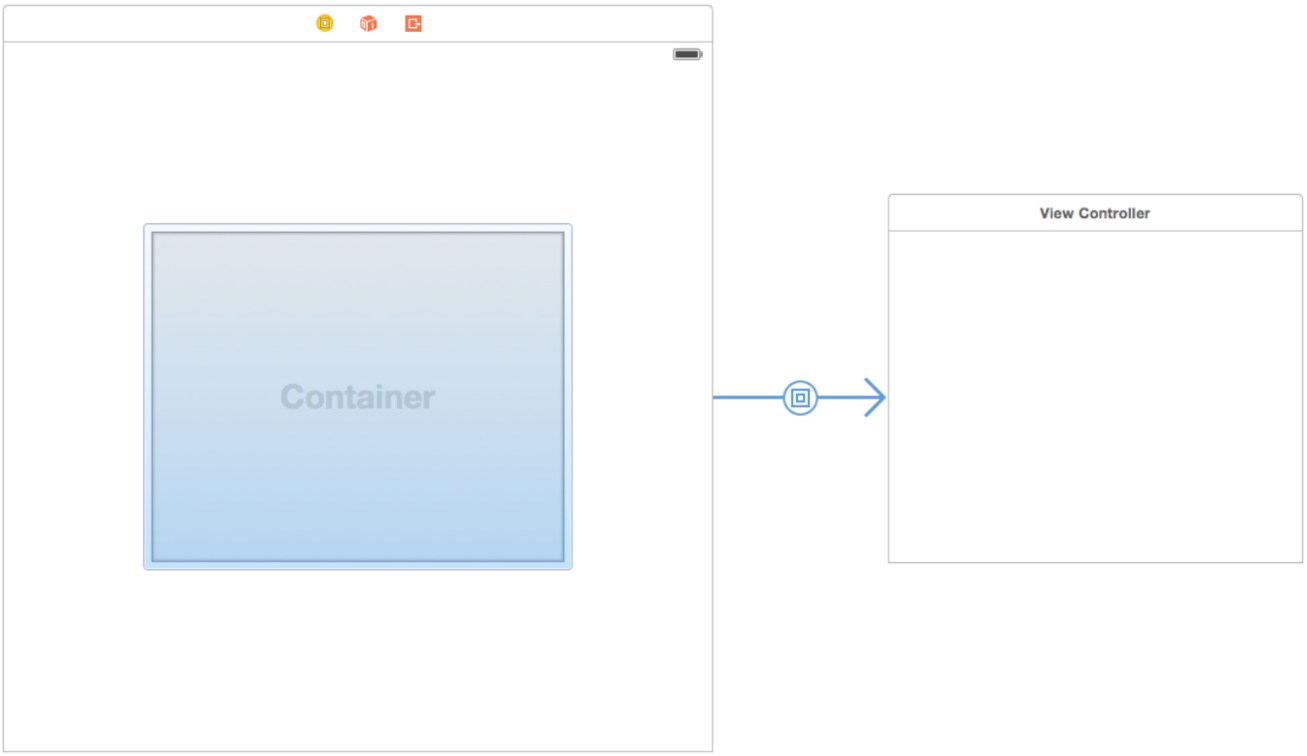


Figura 5.3: Contenedores iOS

La interfaz de usuario ha sido diseñada para ser nativa a cada plataforma y ser fiel a las guías de diseño propuestas por cada fabricante. De esta forma, las vistas con pestañas muestran la barra de pestañas en las posiciones superior o inferior de la vista, dependiendo de la plataforma destino [5, 15], otro ejemplo, son los diálogos, *Android* aconseja su uso [11], sin embargo, *iOS* no los considera como tal, sino que proporciona unas pocas formas de presentación para la vistas, siendo el framework *Cocoa Touch* quien maneja su forma final en pantalla, siendo lo único parecido a un dialogo Android, el estilo de presentación *Form Sheet*.

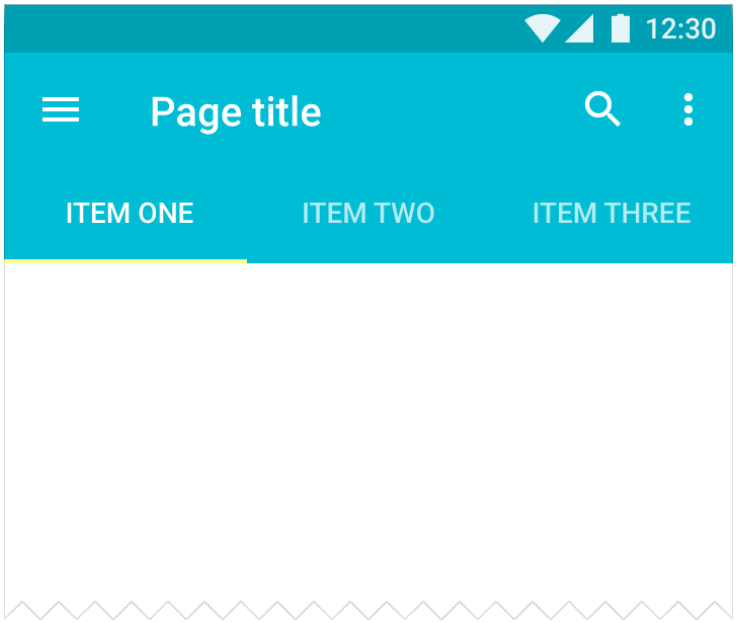


Figura 5.4: Pestañas Android

Figura 5.5: Pestañas iOS

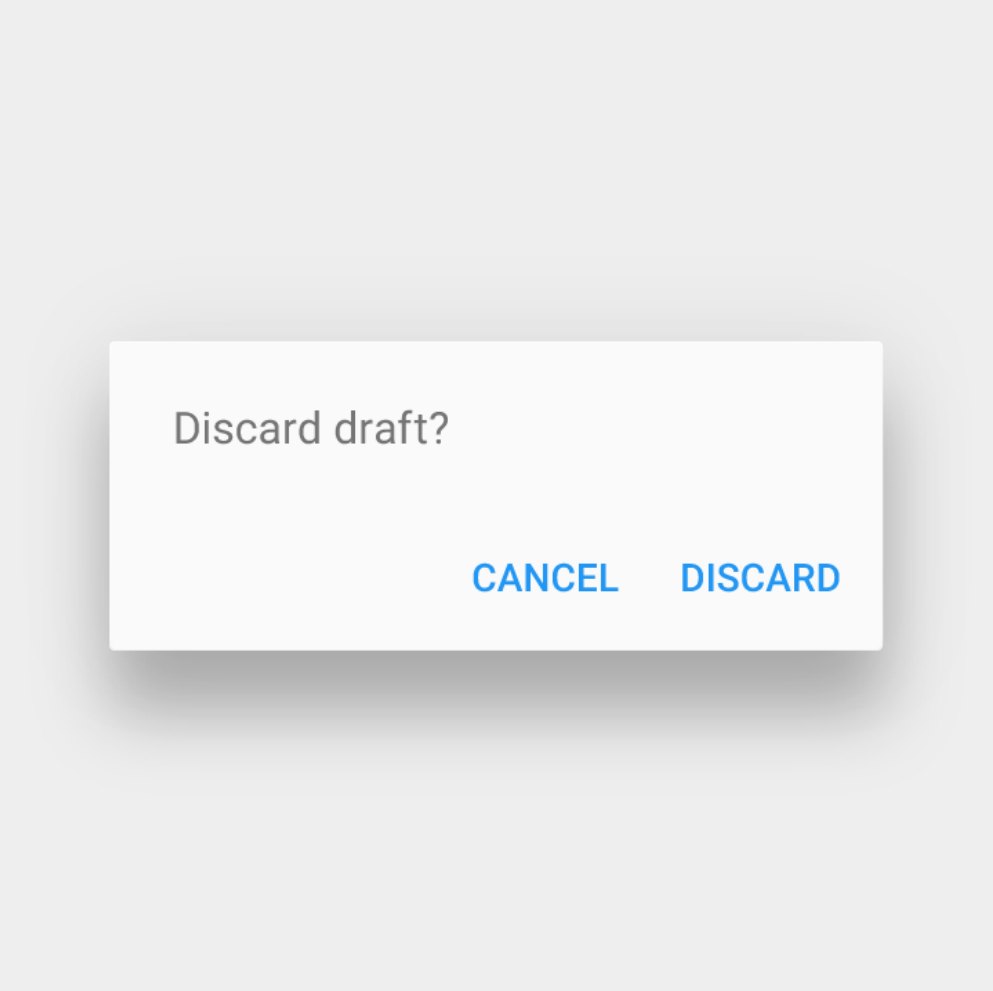


Figura 5.6: Dialogos Android

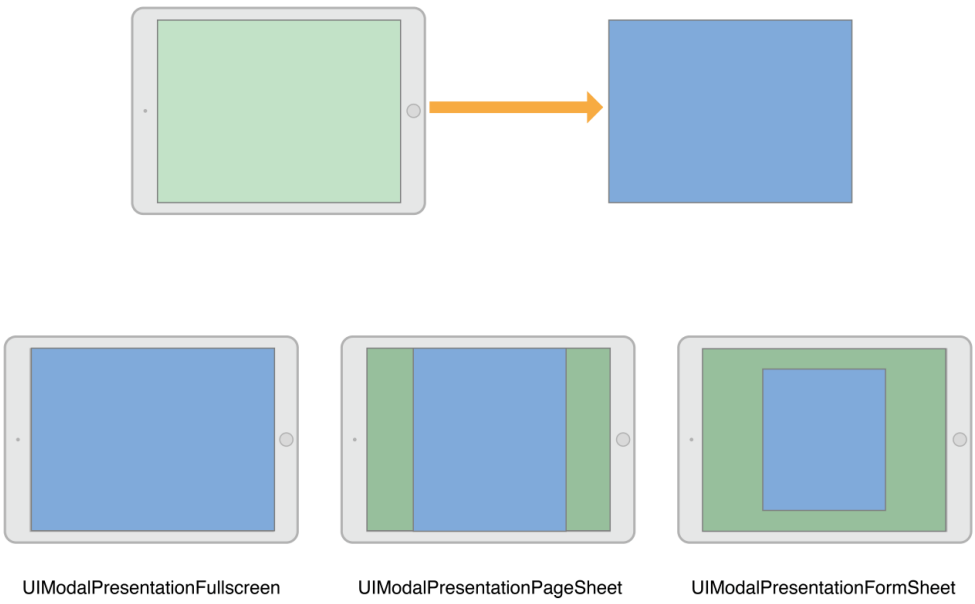


Figura 5.7: Posibles presentaciones para las vistas en iOS

5.1.3. Características de los lenguajes Swift y Java

Además de las diferencias entre los Frameworks de desarrollo, cada plataforma trabaja en unos lenguajes de programación. *Swift* y *Objetive-C* para *iOS*, y *Java* y *C++/C* para *Android*. Este trabajo ha sido desarrollado utilizando los lenguajes aconsejados por cada fabricante, *Swift 3* y *Java 7*.

```
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

```
static func sumOf(numbers: Int...) -> Int {
    var sum = 0
    for number in numbers {
        sum += number
    }
    return sum
}
```

Aparte de las diferencias claras en sintaxis, cada lenguaje tiene una filosofía y unas construcciones diferentes. Estas diferencias, serán aprovechadas para el diseño y desarrollo de *Skywalker*, favoreciendo un desarrollo natural a cada entorno, por ejemplo, mientras en *Java* una función no es un ciudadano de primer nivel, y por tanto no puede ser almacenado ni pasado como argumento de una función, *Swift* si lo permite. Esta característica, será aprovechada, entre otros, en el diseño de las peticiones al servidor, tal y como se muestra en el fragmento de código siguiente.

```
ServerHandler.getInstance(getActivity().getApplicationContext()).
              getAvaliableTags(
                      new ServerHandler.
                      OnServerResponse<List<PointOfInterest>>() {
          @Override
          public void onSuccess(List<PointOfInterest> response) {
              body...
          }

          @Override
          public void onError(ServerHandler.Errors error) {
              body...
          }
});
```

```
let onSuccess: ([PointOfInterest]) -> Void = {points in
        body...
}


let onError: (ServerHandler.ErrorType) -> Void = {error in
        body...
}

try! ServerHandler.instance.getAvaliableTags(
        onSuccess: onSuccess,
         onError: onError)
```

Puede observarse cómo, mientras en *Java* nos vemos forzados a definir una interfaz, e implementarla para pasar un objeto de dicha clase como parámetro, para que a posteriori, se llame a los métodos de dicho objeto. Sin embargo, en *Swift* basta con crear una función, que se acomode al argumento solicitado por el prototipo de la llamada, para que dicha función sea utilizada por la función que la recibe como argumento, simplificando el trabajo a realizar.

5.1.4. Concurrencia

Uno de los objetivos de este desarrollo es el mantener una experiencia de usuario fluida, para ello el objetivo son lograr los 60 FPS en el dibujado de la interfaz de usuario.

Para lograrlo, será necesario el uso de hilos de ejecución para repartir la carga de trabajo, sin embargo, ambos sistemas tienen una restricción común, el hilo principal será el único que pueda manejar las vistas, así como la interacción por parte del usuario [17, 6], con esta restricción en mente, se propone el siguiente modelo:

1. El hilo principal se encargará del dibujado de la interfaz de usuario y de la interacción por parte del usuario.
2. Hilo que mostrará la vista de la cámara trasera en pantalla.
3. Hilo encargado de dibujar los elementos en pantalla, dependiendo de la plataforma, este tendrá un diseño u otro para cada plataforma, debido a la restricción del hilo encargado de la interfaz de usuario.
4. Hilo manipulador de los cambios en el sensor de orientación, realizando los cálculos y el tratamiento necesario de los datos para obtener la dirección actual del dispositivo.
5. Hilo que enviará una señal Bluetooth LE a las antenas receptoras.
6. Un último hilo para ejecutar las peticiones al servidor y actualizar los datos de posicionamiento.

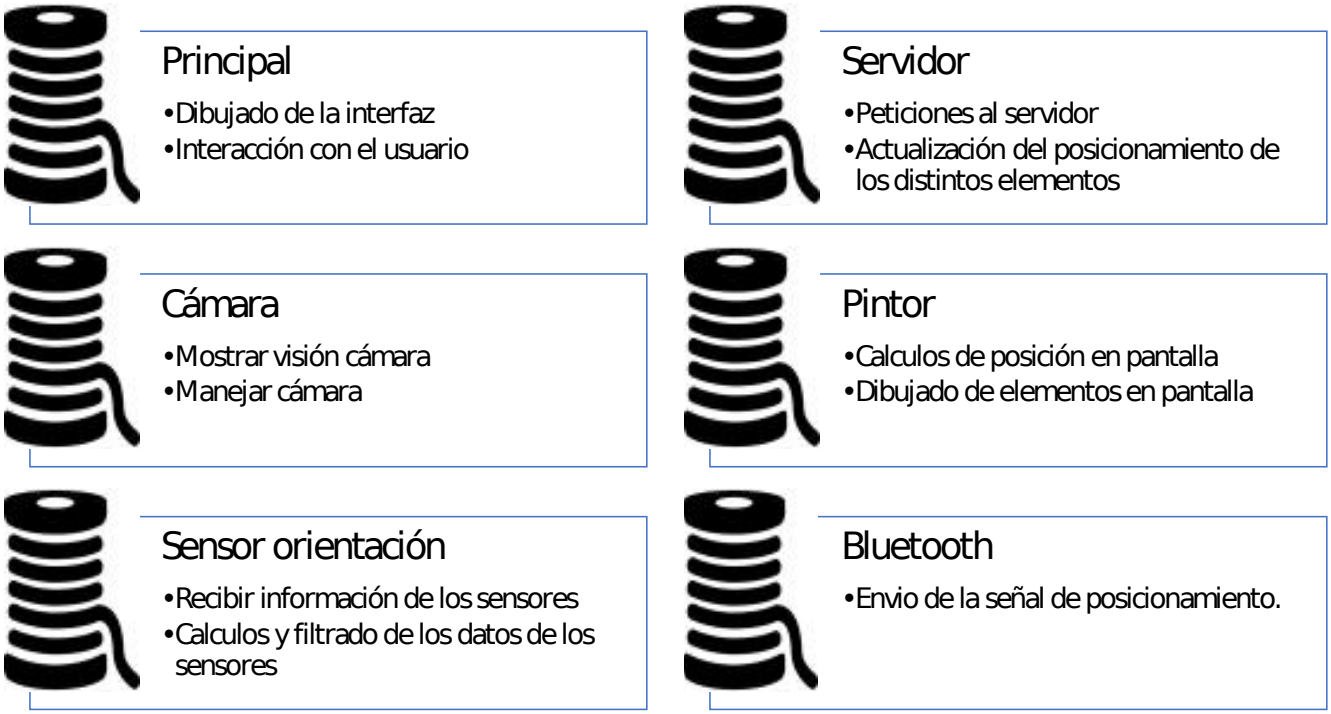


Figura 5.8: Concurrencia diseñada para Skywalker.

Para facilitar la tarea de creación de hilos, ambas plataformas proponen frameworks y métodos especiales para la generación de peticiones asíncronas, con la posibilidad de repartir trabajo entre el hilo principal y estos hilos extras. Aún así, también se proporcionan los clásicos hilos y semáforos, aunque se recomienda el uso de los frameworks de alto nivel proporcionados en cada plataforma, por su facilidad de uso.

Sin embargo, se ha decidido el uso de los threads de bajo nivel para las operaciones de dibujado y de peticiones de actualización al servidor en la versión *Android*, el motivo es simple, estas abstracciones están pensadas para un uso esporádico, y no para un uso continuo de dichos hilos, sin embargo, proporcionan una gran ventaja respecto de los hilos tradicionales, y es que ambas construcciones poseen planificadores que se encargan del uso eficiente de hilos. Para el resto de operaciones, se utilizarán estos frameworks de paralelismo, por su facilidad de uso, y porque la mayoría de constructores de otros frameworks utilizados toman como parámetro algún tipo de hilo o cola de operaciones.

Como ejemplo en *Android*, consideraremos el framework de la cámara, en concreto la clase *CameraCaptureSession* [19]. Puede observarse como el método

```
void setRepeatingRequest(CaptureRequest request,
                         CameraCaptureSession.CaptureCallback listener,
                         Handler handler)
```

toma como parámetro un elemento de tipo *Handler*, si miramos la documentación de dicho método podemos leer lo siguiente:

> Handler: the handler on which the listener should be invoked, or null to use the current thread's looper.

Si profundizamos más, y miramos la documentación de la clase *Handler* [23], observamos:

> *There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread*

than your own.

Por lo que podemos asociar una cola de eventos en un hilo externo para el manejo de los eventos de la cámara, evitando de este modo el bloqueo del hilo principal y consiguiendo una de las partes del modelo propuesto para el reparto del trabajo.

Este ejemplo ilustra perfectamente el diseño a seguir para el resto de elementos de *Android*, ya que el uso de *Handlers* esta extendido por todo el sistema.

Por otra parte, iOS proporciona *GCD* [3], un framework de alto nivel para facilitar la concurrencia y aprovechar el hardware disponible de la mejor forma posible. De forma breve, *GCD* permite el envío de operaciones, al estilo de los *Handlers* de *Android*, entre distintos hilos, aunque de una manera mucho más sencilla y directa, en parte gracias al lenguaje en sí.

En el siguiente ejemplo puede verse como enviar una petición asíncrona a una cola externa, para que esta envíe a su vez una petición al hilo principal de la aplicación.

```
queue.async {
        print("Hilo externo")
        DispatchQueue.main.sync {
                print("Hilo principal")
        }
}
```

GDC proporciona distintos elementos para el manejo de la asincronía, entre los que destacan:

1. Dispatch Queues: Colas de peticiones a ejecutar.
2. Dispatch Sources: Para el manejo de eventos del SO.

Además, se permite la personalización de estas unidades de ejecución independientes, permitiendo la ejecución en formato *FIFO* o la ejecución de las solicitudes forma completamente concurrente. Otra ventaja es que *GCD* asegura la iniciación de la ejecución de las peticiones en el orden de llegada, evitando condiciones de carrera por este motivo. Por último, a estas colas de ejecución se les puede asociar un *Quality of Service*, cambiando así el planificador que las controla, y distinguiendo entre los siguientes tipos:

1. *User-interactive*
2. *User-initiated*
3. *Utility*
4. *Background*

La gran ventaja de *GCD* en el aspecto del rendimiento es que es el propio sistema el encargado de creación de los hilos, este sistema se encarga de monitorizar la carga de trabajo del sistema en el momento de realización de una petición, dependiendo de este factor, se cogerá un hilo de un *Thread pool* que el sistema posea, o se realizará la operación en otro hilo que ya se encuentre en ejecución, siendo completamente transparente al programador.

Se utilizará *GCD* en toda la aplicación, ya que se han realizado pruebas de rendimiento, comprobando que este sistema cumple los requisitos de tiempo necesarios para un dibujado en tiempo real. Como en

el caso de *Android*, las *DispatchQueues* están aplicadas en todas las llamadas del sistema, permitiendo el uso de éstas para manejar los distintos eventos, como puede verse en el siguiente ejemplo, extraído del subsistema [2]:

```
func startDeviceMotionUpdates(to queue: OperationQueue,
                withHandler handler: @escaping CMDeviceMotionHandler)
```

Nótese el parámetro *queue*, de tipo *OperationQueue*, con la siguiente descripción:

> *An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.*

Es decir, la cola de operaciones donde queremos que se ejecuten los eventos del sensor. Este ejemplo puede extenderse a todo el framework de *iOS*.

5.1.5. Sensor de orientación

El objetivo es conseguir un vector normal al dispositivo, donde X apunte al norte magnético, Z apunte al suelo, siendo Y ortogonal a ambos y apuntando al este magnético, todo ello representado en coordenadas del mapa proporcionado por XtremeLoc, lo cual no será sencillo, debido a los siguientes aspectos:

Interferencias magnéticas Los sensores de estos dispositivos se ven afectados por interferencias electromagnéticas [14, 4], por lo que cabe la posibilidad de que se vean afectados por dichas interferencias.

Calibración del sensor Ambos Frameworks proporcionan callbacks con actualizaciones de la precisión del sensor, una vez se activen esas llamadas, habrá que informar al usuario para que calibre el dispositivo mediante movimiento.

Sensibilidad Los valores deberían ser filtrados para evitar cambios en los valores producidos por pequeñas fluctuaciones, con este fin, se ha diseñado una solución con un filtro de paso bajo [8]; un filtro vía software que permite el paso de señales con una frecuencia menor que una frecuencia de corte, y atenúa las señales mayores a dicha frecuencia de corte. El siguiente código java proporciona dicho filtro:

```
float[] lowPass( float[] input, float[] output ) {
    if ( output == null ) return input;
    for ( int i=0; i<input.length; i++ ) {
        output[i] = output[i] + ALPHA * (input[i] - output[i]);
    }
    return output;
}
```

Una vez implementado, habrá que probar con dispositivos reales el valor *ALPHA* ideal.

Ambas plataformas cuentan con un tipo especial de sensor, que proporciona la orientación espacial del dispositivo en tres representaciones distintas:

Ángulos eulerianos Tomando como referencia la representación de la figura 5.9 tenemos:

1. α: Ángulo entre el eje x y el eje N.

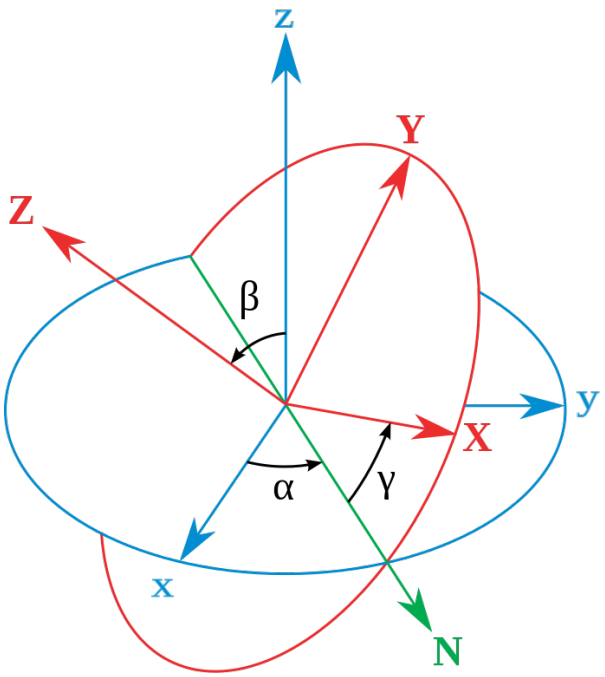


Figura 5.9: Representación de ángulos eulerianos.

2. β: Ángulo entre el eje z y el eje Z.

3. γ: Ángulo entre el eje N y el eje X.

Matrices de rotación

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Cuaterniones

$$\mathbb{H} = \{a + bi + cj + dk : a, b, c, d \in \mathbb{R}\} \subset \mathbb{C}^2 \subset \mathbb{R}^4$$

De ellas, la más intuitiva son los ángulos eulerianos, sin embargo, se podría producir el bloqueo del cardán [26], perdiendo así un grado de libertad y produciendo medidas no deseadas e incorrectas. Para solucionar esto, tendremos que usar cualquiera de las otras opciones.

La principal ventaja de los cuaterniones respecto de las matrices de rotación, es que su computación es más eficiente, a la vez que ocupan menos espacio en memoria, sin embargo, y debido a que solo habrá un representación activa a la vez, podemos lidiar con estas desventajas, y es que la programación de las funciones necesarias para los cálculos son más sencillas en las matrices. Además, las matrices de rotación poseen la siguiente propiedad:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

Donde R representa una rotación cuyos ángulos tait-bryan son α, β y γ sobre los ejes z, y, x respectivamente [38]. Por lo que rotar una matriz de rotación, sobre un eje arbitrario es tan sencillo como realizar el producto

$$R' = RR_e$$

Donde R_e es la matriz de rotación sobre un eje arbitrario.

Una vez elegida la representación adecuada, el siguiente objetivo será conseguir el vector normal al dispositivo, aquí se presenta otro problema, y es que el marco de referencia de los dispositivos no está preparado para trabajar con el dispositivo en vertical, sino en horizontal, como si estuviera apoyado sobre una mesa, aquí entra en juego la propiedad de composición de las matrices de rotación; queremos rotar dichas matrices para cambiar su marco de referencia, por lo que se realizará una multiplicación de la matriz dada por el sensor con una matriz de rotación sobre el eje correspondiente a cada plataforma con una rotación de 90º, obteniendo así la matriz final de rotación.

Por suerte, ambos frameworks proporcionan utilidades para lograr este fin, por su parte, *Android* tiene una llamada propia del sensor que produce este efecto. Por su parte, *iOS* tiene clases matemáticas especiales, altamente optimizadas tanto en tiempo como consumo energético, que realizan la rotación de una matriz sobre el eje solicitado, con el ángulo deseado.

En último lugar, se extrae el vector normal al dispositivo de la matriz de rotación final, aprovechando, una vez más, la multiplicación de matrices.

$$V_{normal} = M_f V_r$$

Siendo V_r

1. Android: $V_r = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$
2. iOS: $V_r = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

Esta diferenciación se debe a que *Android* considera que es el eje Y del marco de referencia del sensor el que apunta al norte magnético, e *iOS* considera, por su parte, que es el eje X del marco de referencia del sensor el que debe apuntar al norte magnético.

Para obtener el vector representado en coordenadas de XtremeLoc bastará con aplicar al vector de orientación obtenido el desfase existente entre el norte magnético real y norte magnético representado en coordenadas de XtremeLoc, el cual se obtiene mediante la API REST del sistema.

Sensor de orientación óptimo

Lo expuesto anteriormente proporciona el vector deseado, pero, ¿es realmente necesario tanto trabajo para conseguirlo?. La respuesta es no, pudiendo conseguir un diseño más eficiente en términos computacionales, además de aclarar dicho diseño e implementación. Hay dos puntos claves fácilmente mejorables, la conversión del sistema de coordenadas y la obtención del vector de orientación a través de la matriz de rotación.

Respecto de la conversión de coordenadas, rotar la matriz de coordenadas es una operación completamente innecesaria si utilizamos el vector $V_r = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ para ambas versiones, ya que como se ha dicho anteriormente, ambos Frameworks tienen como referencia un estado horizontal. Basándonos en la forma de trabajar de las matrices de rotación, deducimos que llevar un punto del estado original al

nuevo estado descrito por dicha matriz no necesita realmente de ninguna transformación para la matriz, si somos capaces de obtener el correspondiente punto a transformar descrito en términos de la referencia utilizada por la matriz.

Segundo y último, la obtención del vector de orientación anterior conlleva una multiplicación realmente innecesaria, ya que el factor derecho de la multiplicación es, en realidad, la extracción de una de las columnas de la matriz, pudiendo así, extraer los datos directamente, no solo evitando operaciones innecesarias, sino mejorando el uso de la caché, ya que la matriz a utilizar es en realidad un vector.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ d \\ g \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} b \\ e \\ h \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} c \\ f \\ i \end{pmatrix}$$

Para comprobar la mejora que esté diseño mejorado supone, se han implementado ambos diseños en la versión Android, obteniendo la media aritmética en la diferencia en tiempo de ejecución tras 200 muestras:

Versión	Tiempo (ns).
Básica	305
Optimizada	122

Como puede observarse, el nuevo diseño logra ser 2,5 veces más rápido.

5.1.6. Dibujado de elementos en pantalla

Para realizar el dibujado en pantalla de los elementos necesitamos conocer:

- El vector de orientación del dispositivo, adaptado a las coordenadas utilizadas por XtremeLoc.
- El campo de visión de la cámara.
- La posición de cada elemento a dibujar.
- La posición del propio dispositivo.

El primer elemento ya está disponible gracias al sensor de orientación diseñado al igual que el ángulo de visión de la cámara. Por último, las posiciones también son conocidas a la API de XtremeLoc. Lo único que nos queda es componer la información para mostrarla en pantalla.

Por cada punto a mostrar, será necesario realizar un vector desde las coordenadas del dispositivo hacia las coordenadas del punto. Gracias al vector de orientación, podemos obtener el ángulo entre la orientación del usuario y el punto a mostrar. Con estos elementos, lo único que debemos hacer es comprobar si dicho

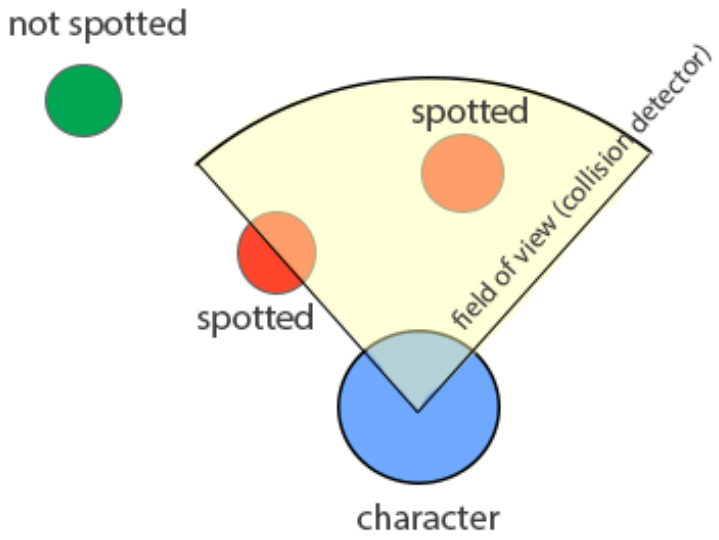


Figura 5.10: Detección de elementos.

ángulo es menor al ángulo de visión horizontal de la cámara, si es así, podemos afirmar que el punto entra dentro de la visión de la cámara, y por tanto deberá ser dibujado como elemento en el campo de visión, en caso contrario, el dibujado será para un elemento fuera del campo de visión.

Los mismos elementos utilizados para esta lógica serán utilizados para conocer la posición en pantalla correspondiente.

Para más detalles, mirar la correspondiente sección a cada plataforma.

5.1.7. Transmisión de señal Bluetooth

Ambas plataformas, permiten el envío de señales Bluetooth de Baja Energía de hasta 31 bytes. Por lo que se generará un elemento de trabajo, cuyo único trabajo consistirá en enviar una señal compatible con el estándar iBeacon [35], durmiendo hasta el siguiente envío.

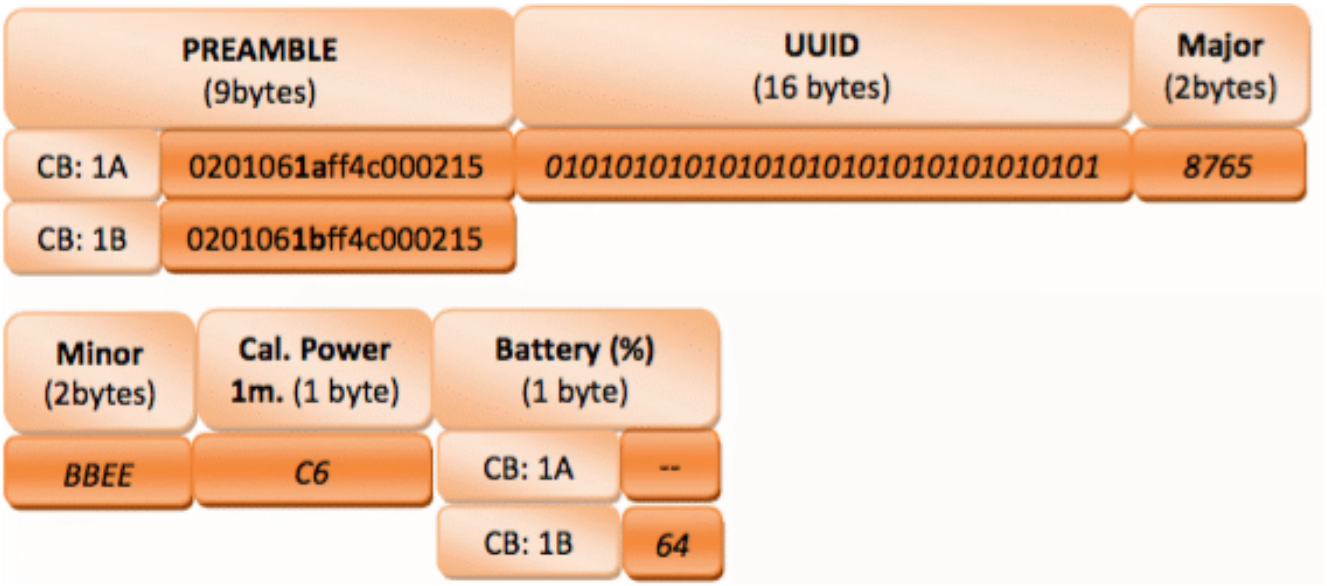


Figura 5.11: Formato del paquete iBeacon.

5.1.8. Selección de elementos a mostrar

Para el diseño de las vistas del filtrado de elementos se necesita aplicar el patrón de diseño *Object Pool* en ambas versiones, debido al énfasis en rendimiento que ambas plataformas tienen.

Una vista de tipo tabla crea de antemano las vistas que necesita mostrar, a medida que el usuario va navegando por dicha tabla, el framework se encarga de modificar las vistas, ejecutando llamadas que el programador ha de implementar para conseguir el comportamiento deseado, evitando así la creación y destrucción innecesaria de objetos, reutilizando aquellas vistas cuyo contenido ya no será visible para mostrar diseños que si serán visibles.

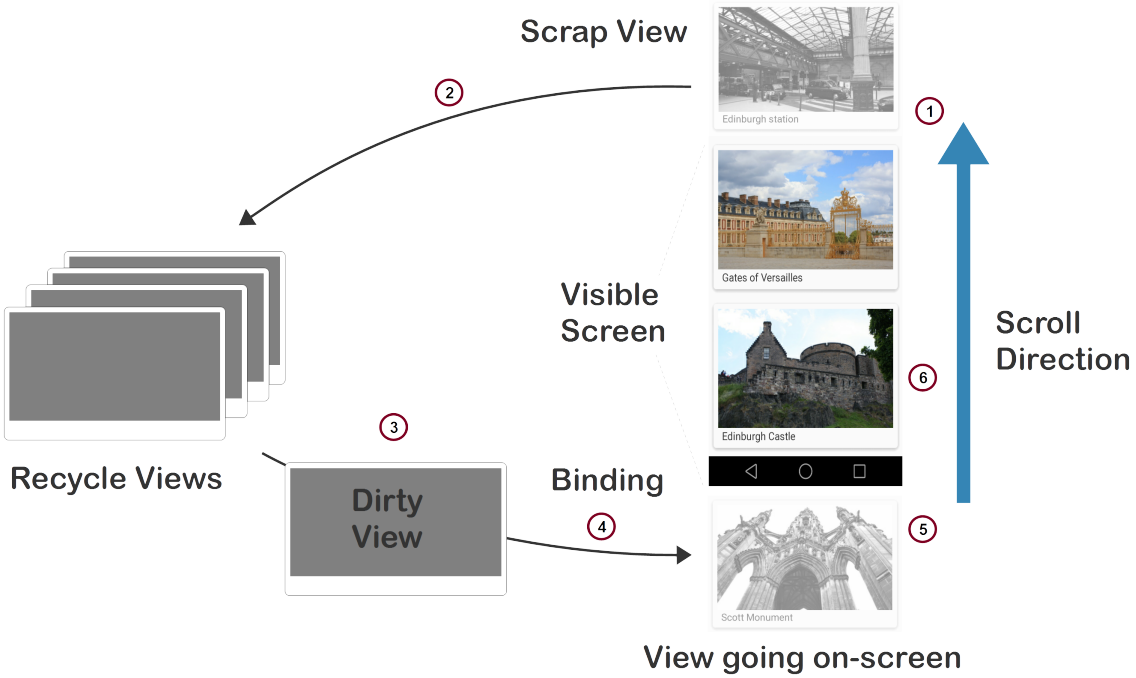


Figura 5.12: Object pool en vistas de tipo tabla.

iOS no permite crear tablas sin este tipo de patrón, sin embargo, *Android* si que lo permite, pero para realizar una versión robusta, se utilizará el patrón en todas las versiones.

5.1.9. Peticiones al servidor

El servidor de *XtremeLoc* responde a peticiones de tipo *REST*, que realizaremos desde un hilo extra, como se había indicado en la Subsección 5.1.4. La documentación correspondiente a dichas peticiones se encuentra en el Apéndice A.

5.2. Diseño Android

5.2.1. Campo de visión de la cámara

Para conseguir el campo de visión de la cámara trasera de un dispositivo *Android* se presentan varias opciones:

1. Confiar en una biblioteca externa o base de datos que contengan la información del dispositivo.
2. Crear un sistema de calibración a medida, para obtener el campo de visión del dispositivo.
3. Utilizar la API de la cámara, en conjunto con trigonometría.

La primera opción no es muy fiable, ya que podríamos caer en datos inexistentes o incorrectos. La segunda opción sería ideal, pero seguramente sería, en si mismo, un proyecto completamente diferente, además de obligar al usuario a calibrar su dispositivo. La última opción confía en que los datos del sensor de la cámara sean los correctos, sin embargo, es una opción universal y funcionaría sin interacción del usuario, por lo que se ha implementado está solución.

Dado que, se van a utilizar los APIs distintas de la cámara, dada la necesidad de soportar versiones antiguas, la obtención del campo de visión tendrá dos variantes:

En la versión 1 de la API, se tienen llamadas que directamente dan los valores del campo de visión, tanto vertical como horizontal, por lo que en esta API no necesitamos trigonometría.

Sin embargo, la versión 2 de la API prescinde de estos métodos, a cambio de otorgar las características del sensor, gracias a las cuales, podremos realizar unos sencillos cálculos para obtener los ángulos deseados.

$$\alpha = 2 \arctan \frac{L}{2f_c}$$

donde L es la dimensión del objetivo, y f_c la longitud focal de la lente [37]. Por suerte, la información del sensor dada por la API nos proporciona estos datos, divididos en categoría horizontal y vertical.

Dado que no se va a permitir hacer zoom sobre la imagen, y el foco va a estar gestionado por la API, no necesitamos cálculos adicionales.

5.2.2. Dependencias entre capas

Las dependencias entre capas, una vez elegidas las clases es el siguiente:

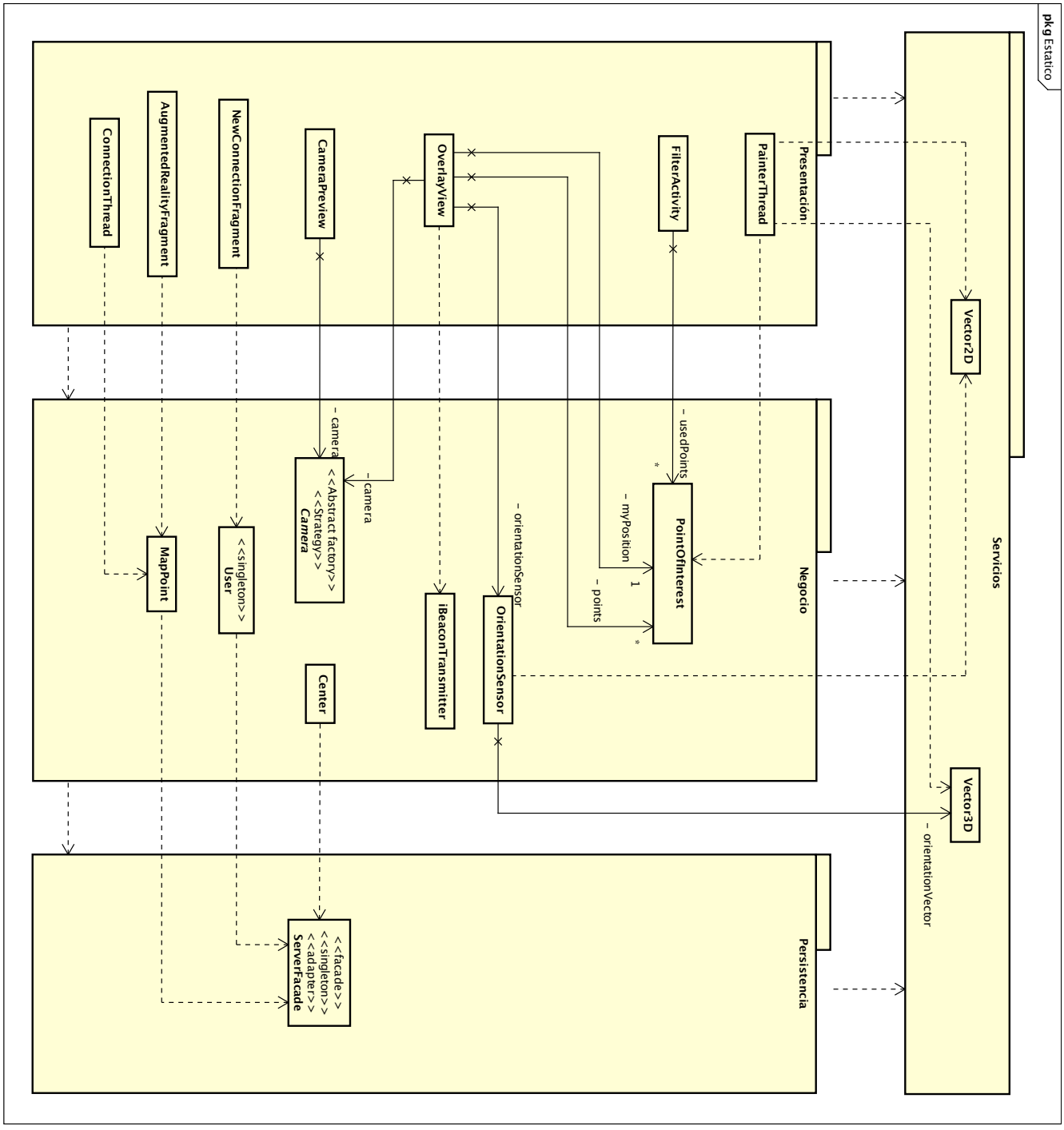


Figura 5.13: Diagrama de dependencias entre capas.

5.2.3. Diagrama de clases

Comenzamos por las clases correspondientes a la arquitectura elegida, separadas en diagramas correspondientes a las distintas capas.

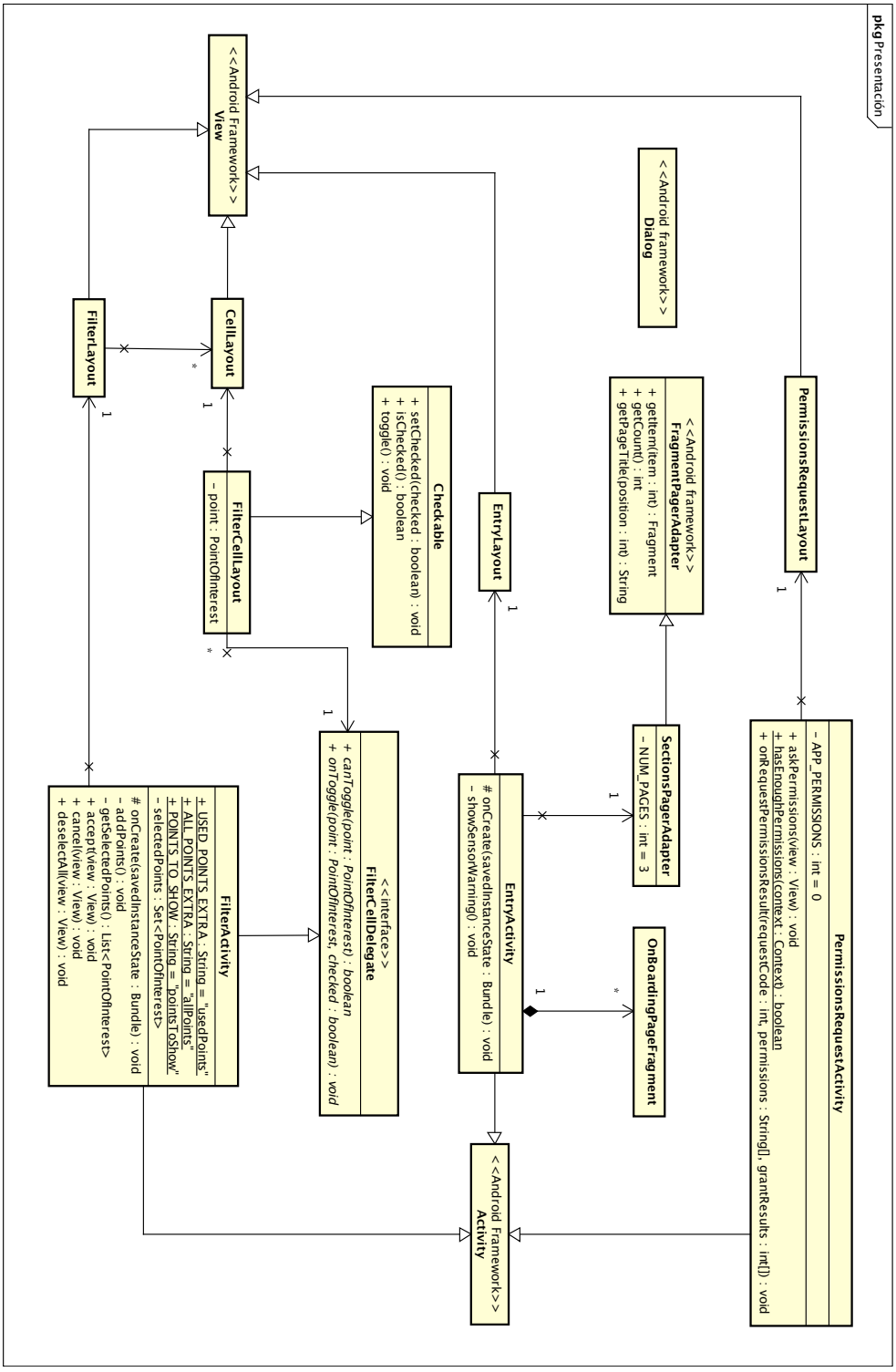


Figura 5.14: 1^{er} Diagrama de clases para la capa de presentación.

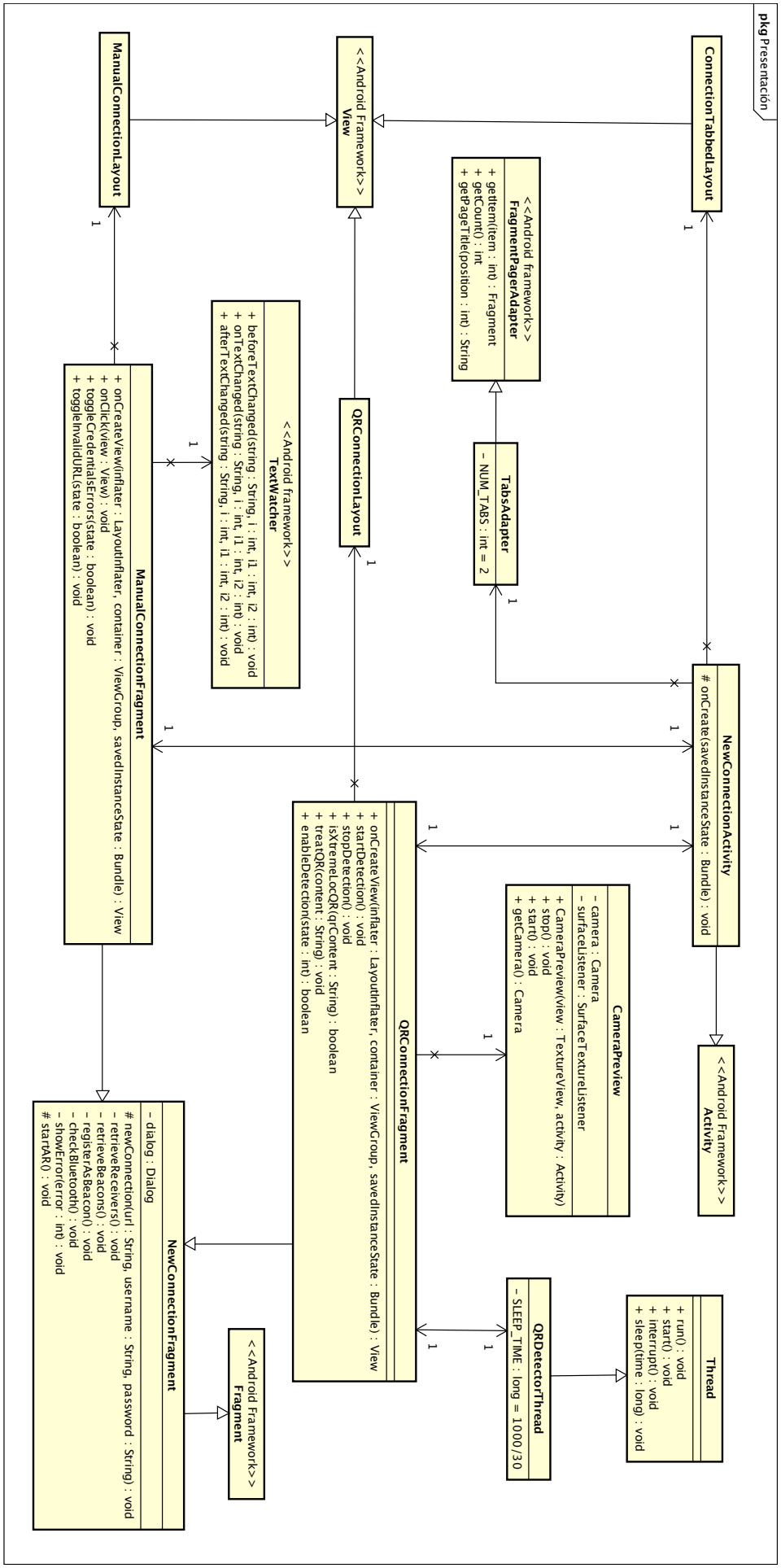


Figura 5.15: 2º Diagrama de clases para la capa de presentación.

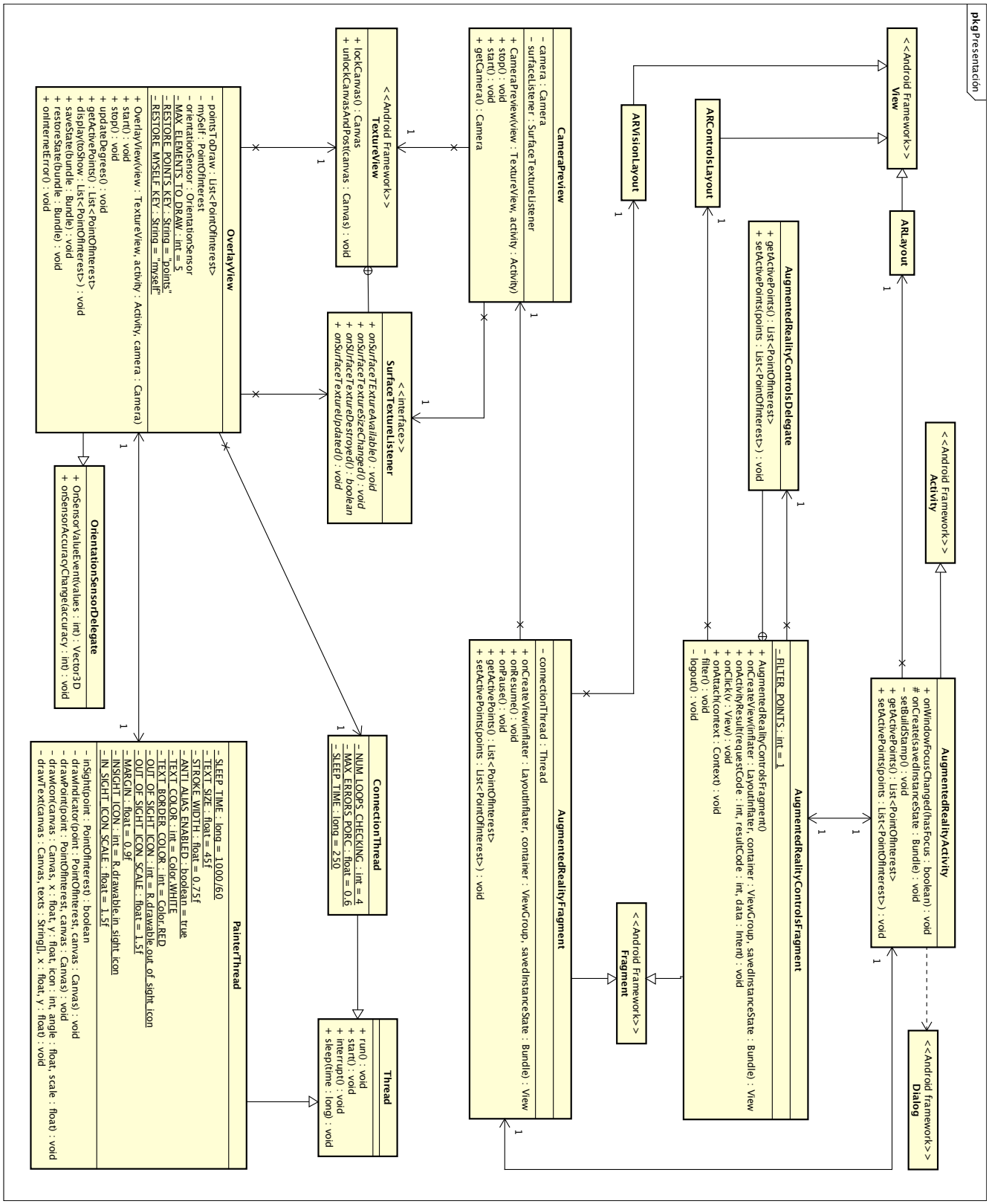


Figura 5.16: 3^{er} Diagrama de clases para la capa de presentación.

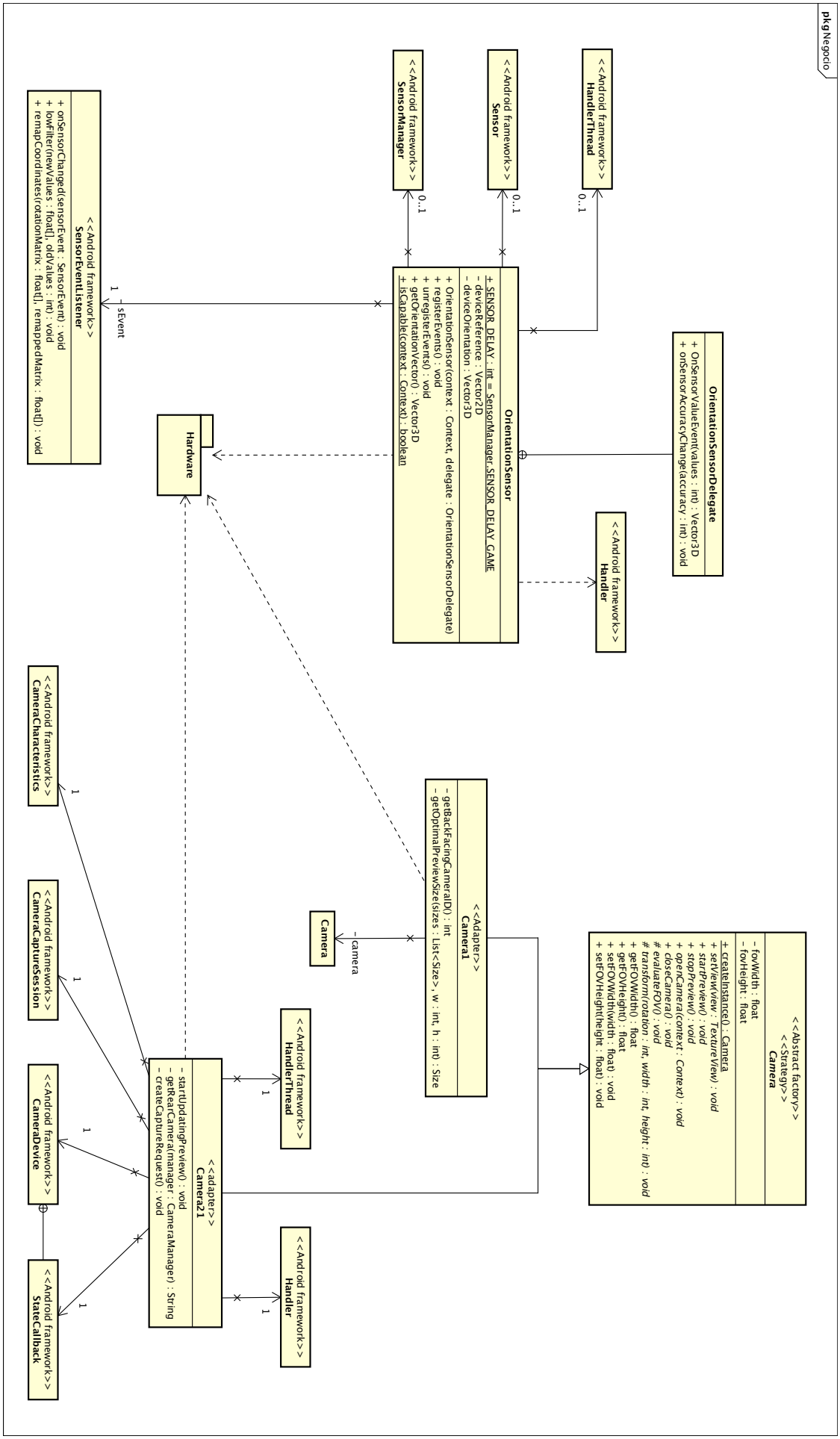


Figura 5.17: 1^{er} Diagrama de clases para la capa de negocio.

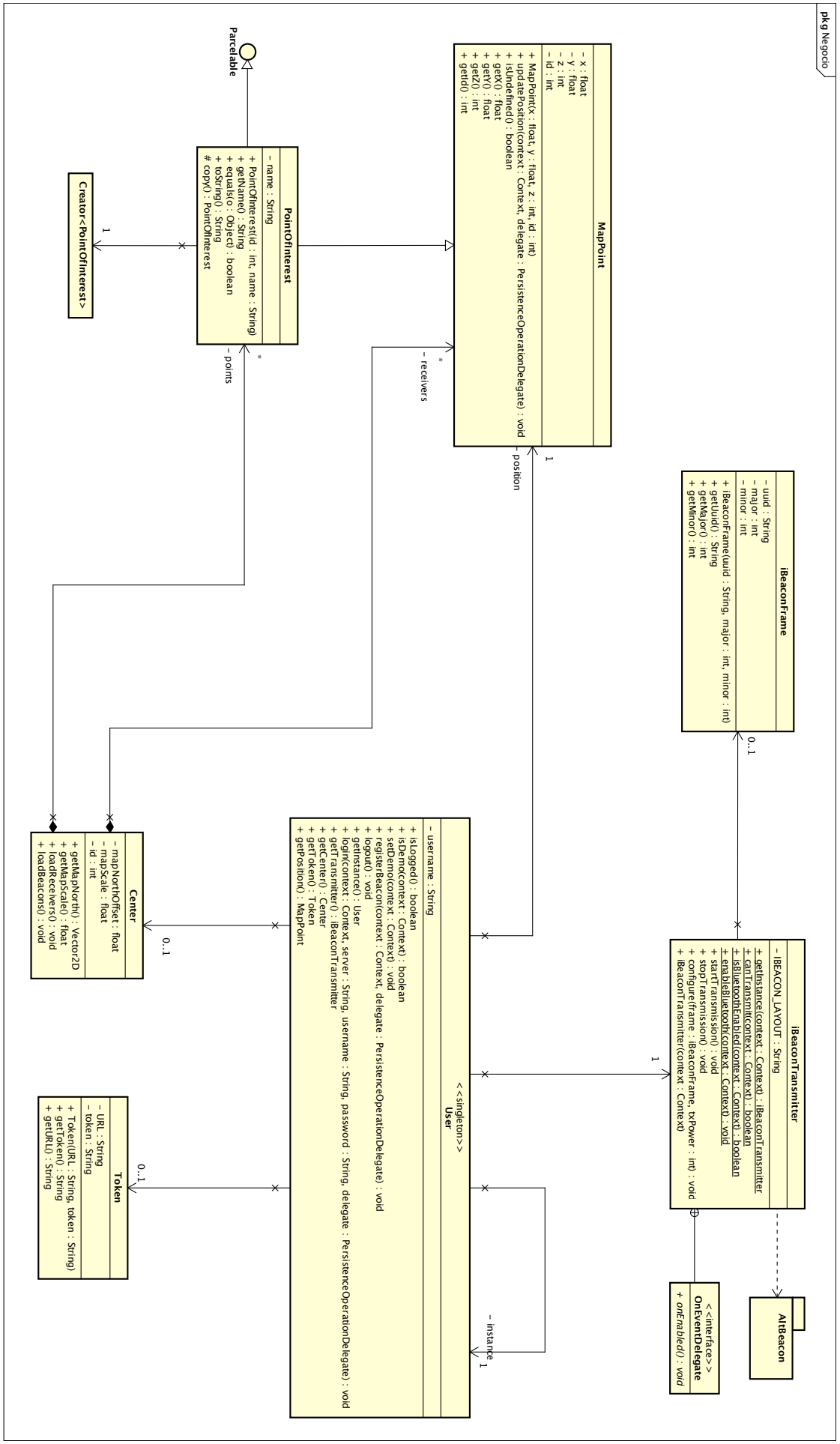


Figura 5.18: 2º Diagrama de clases para la capa de negocio.

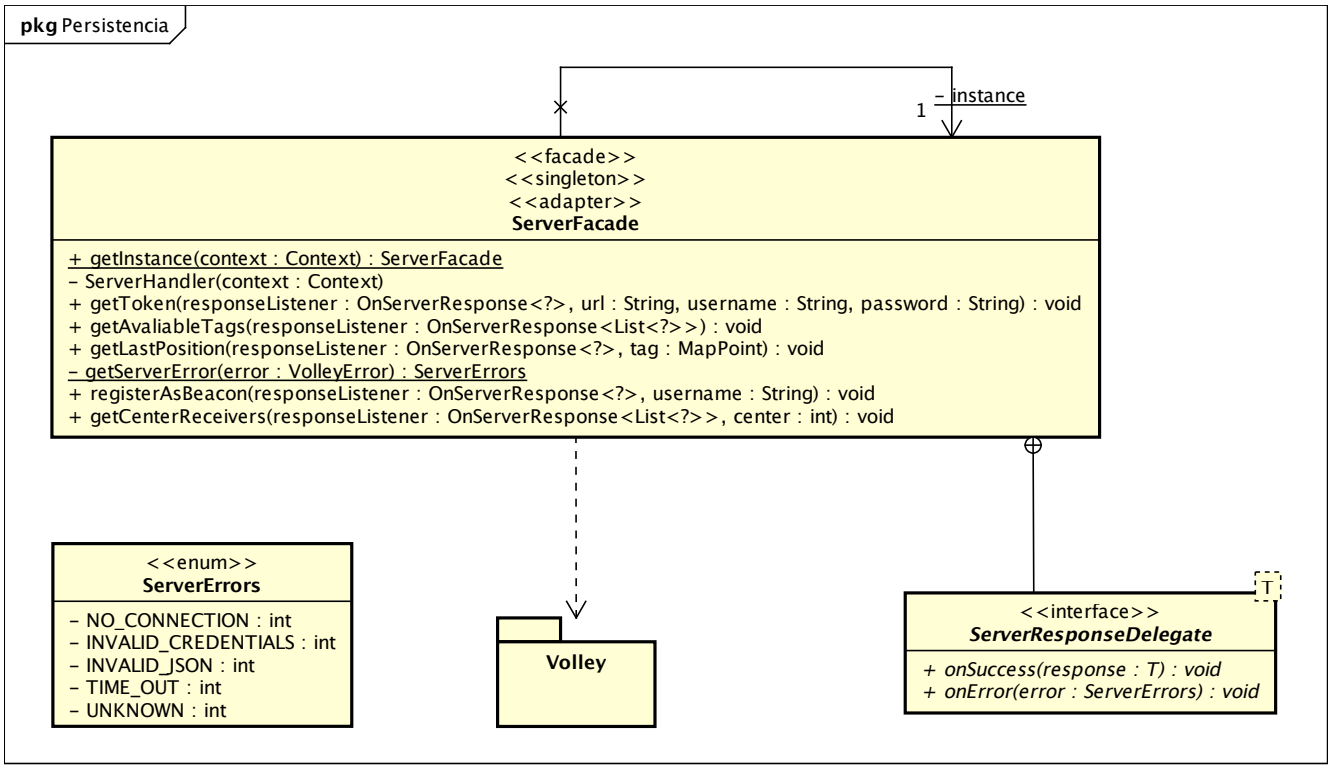


Figura 5.19: Diagrama de clases para la capa de persistencia.

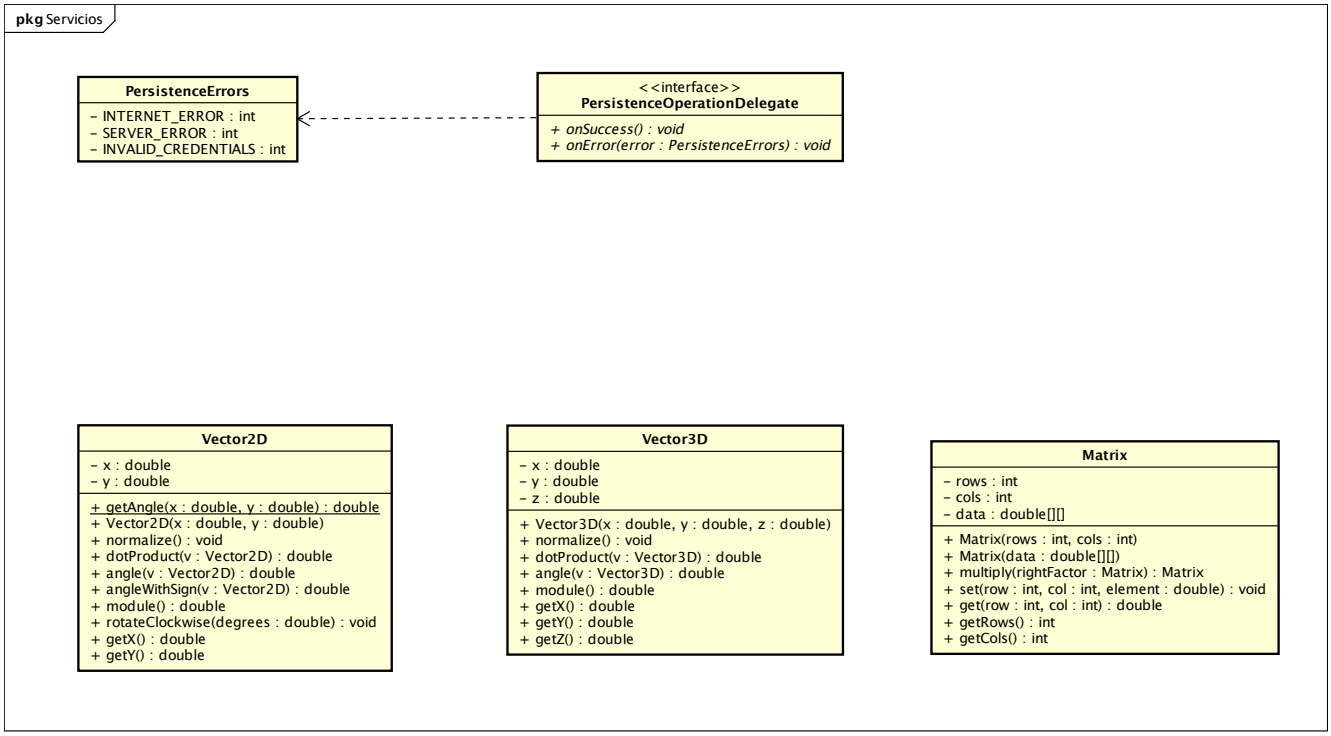


Figura 5.20: Diagrama de clases para la capa común de servicios.

5.2.4. Diagramas de secuencia

Los siguientes diagramas de secuencia se centran en el trabajo generado por los distintos hilos de ejecución. No se incluyen todos los diagramas de secuencia relativos al ciclo de vida completo del sistema, ya que muchas llamadas son simplemente de carga de vistas, generación de *listeners* [20] para eventos del usuario y demás llamadas del Framework *Android*.

Para el dibujado en pantalla, *Android* permite que, un hilo que no sea el principal, dibuje en una vista de tipo *TextureView* [16], de está forma, la gestión de esta tarea será fácilmente implementable.

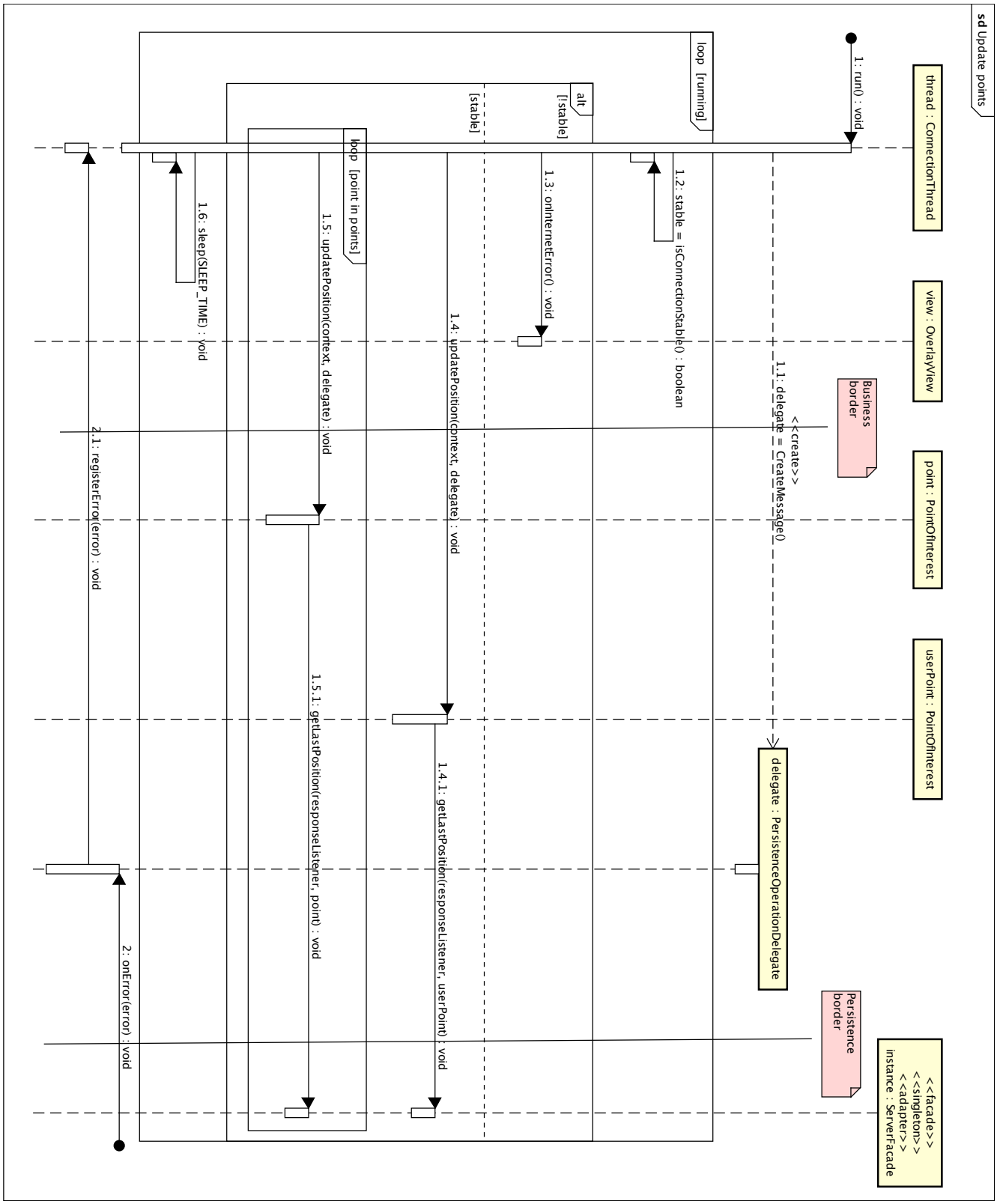


Figura 5.21: Diagrama de secuencia para la petición de la última posición de un beacon.

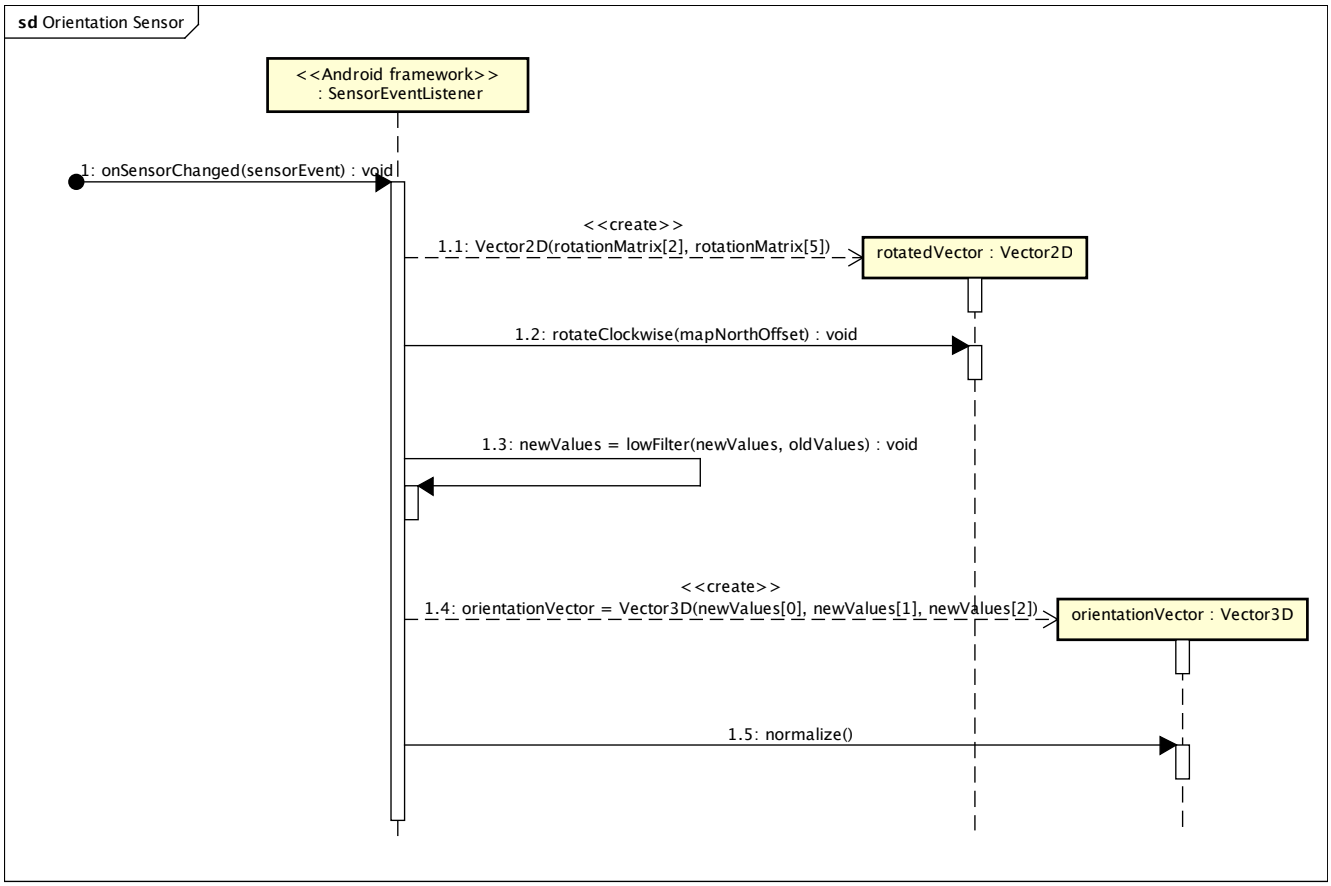


Figura 5.22: Diagrama de secuencia para el hilo que gestiona los cambios en la orientación del dispositivo.

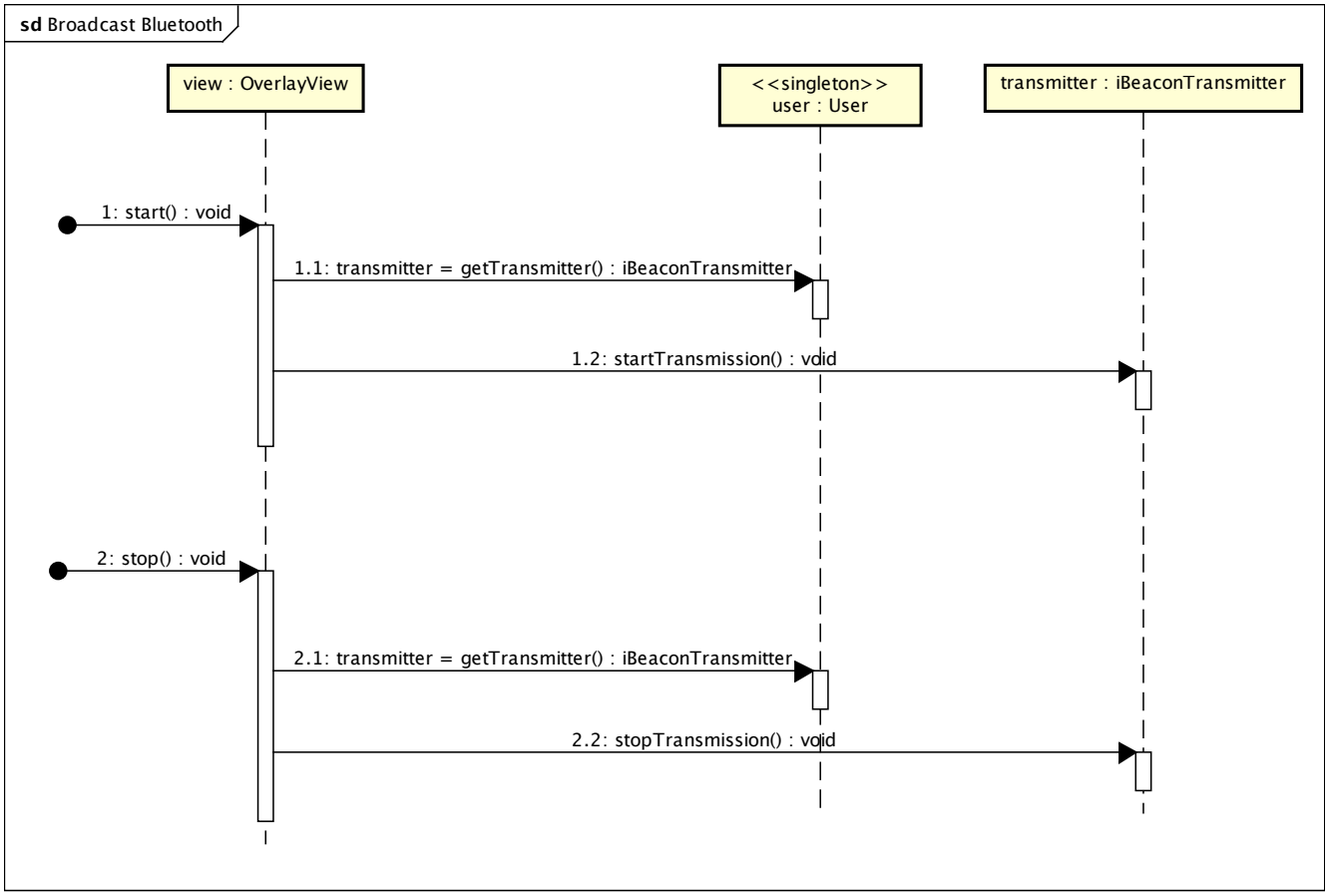


Figura 5.23: Diagrama de secuencia para el manejo de la señal bluetooth a emitir.

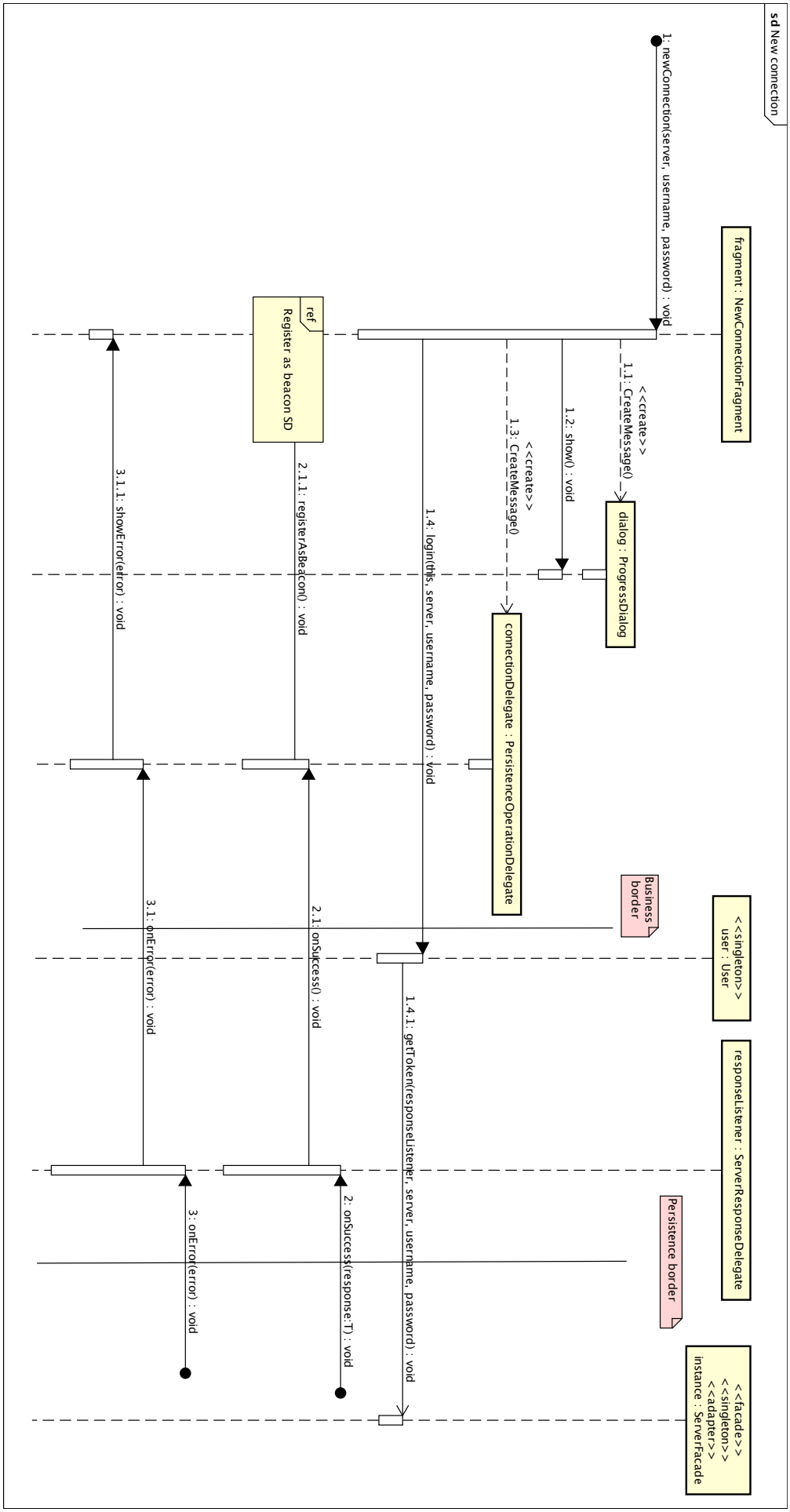


Figura 5.24: Diagrama de secuencia para una nueva conexión con el servidor.

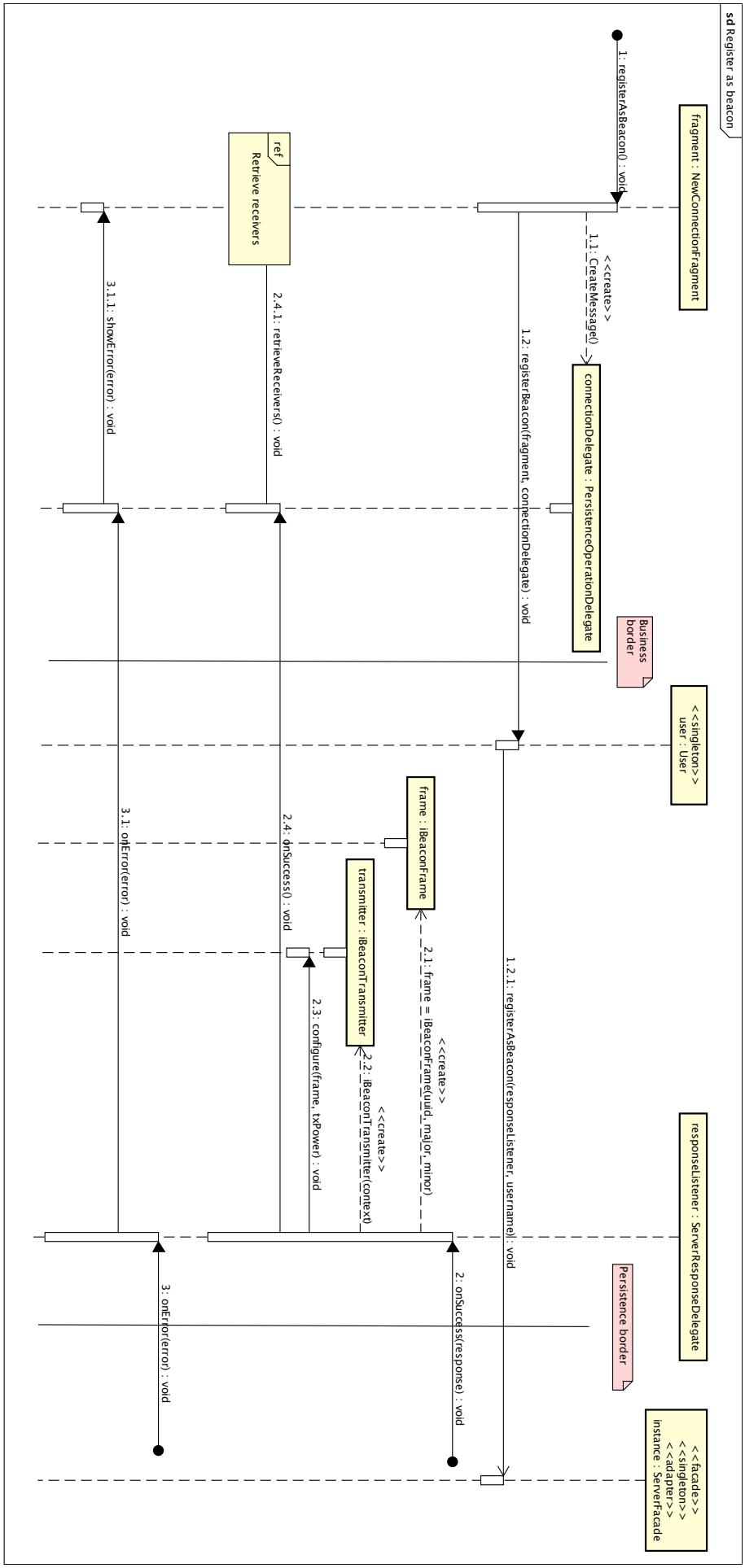


Figura 5.25: Diagrama de secuencia para el registro del dispositivo.

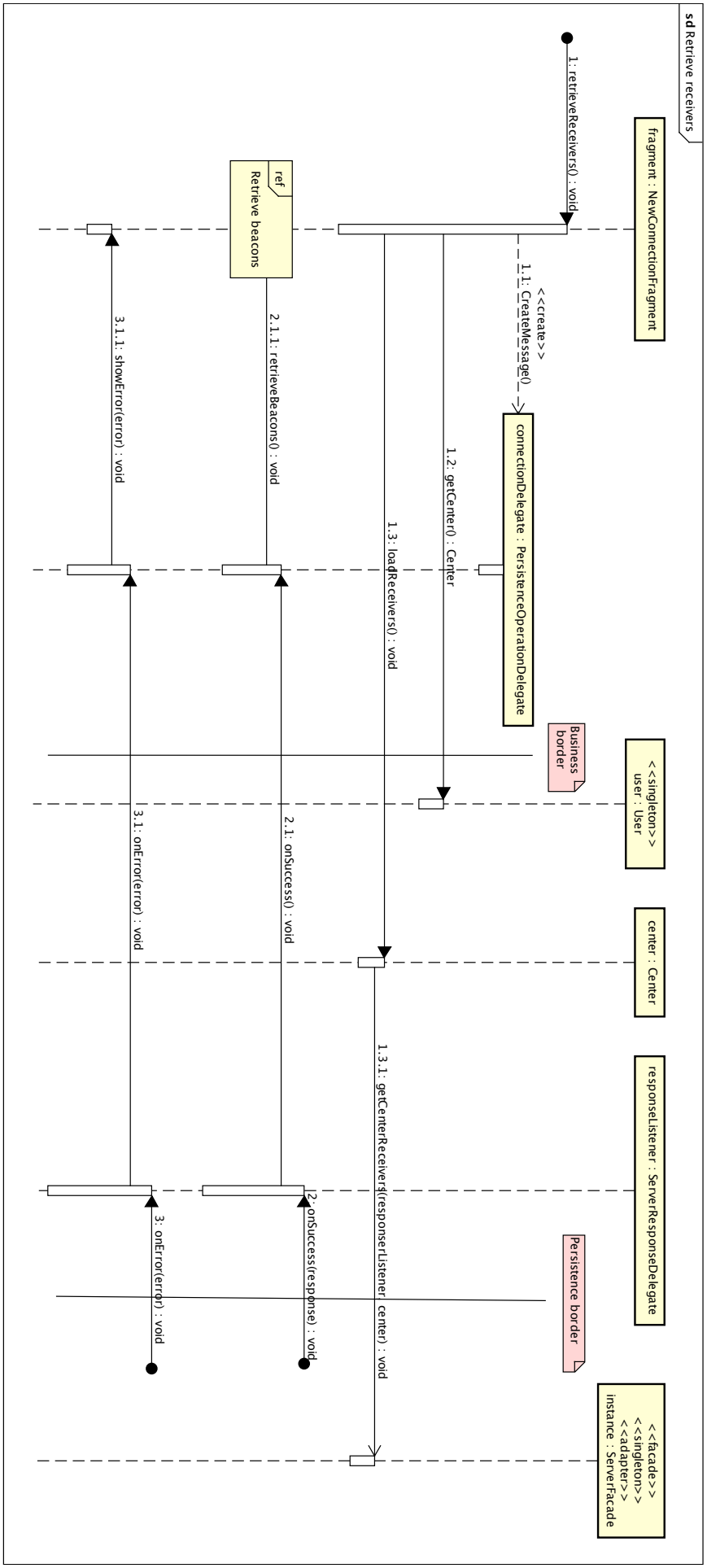


Figura 5.26: Diagrama de secuencia para la obtención de las antenas.

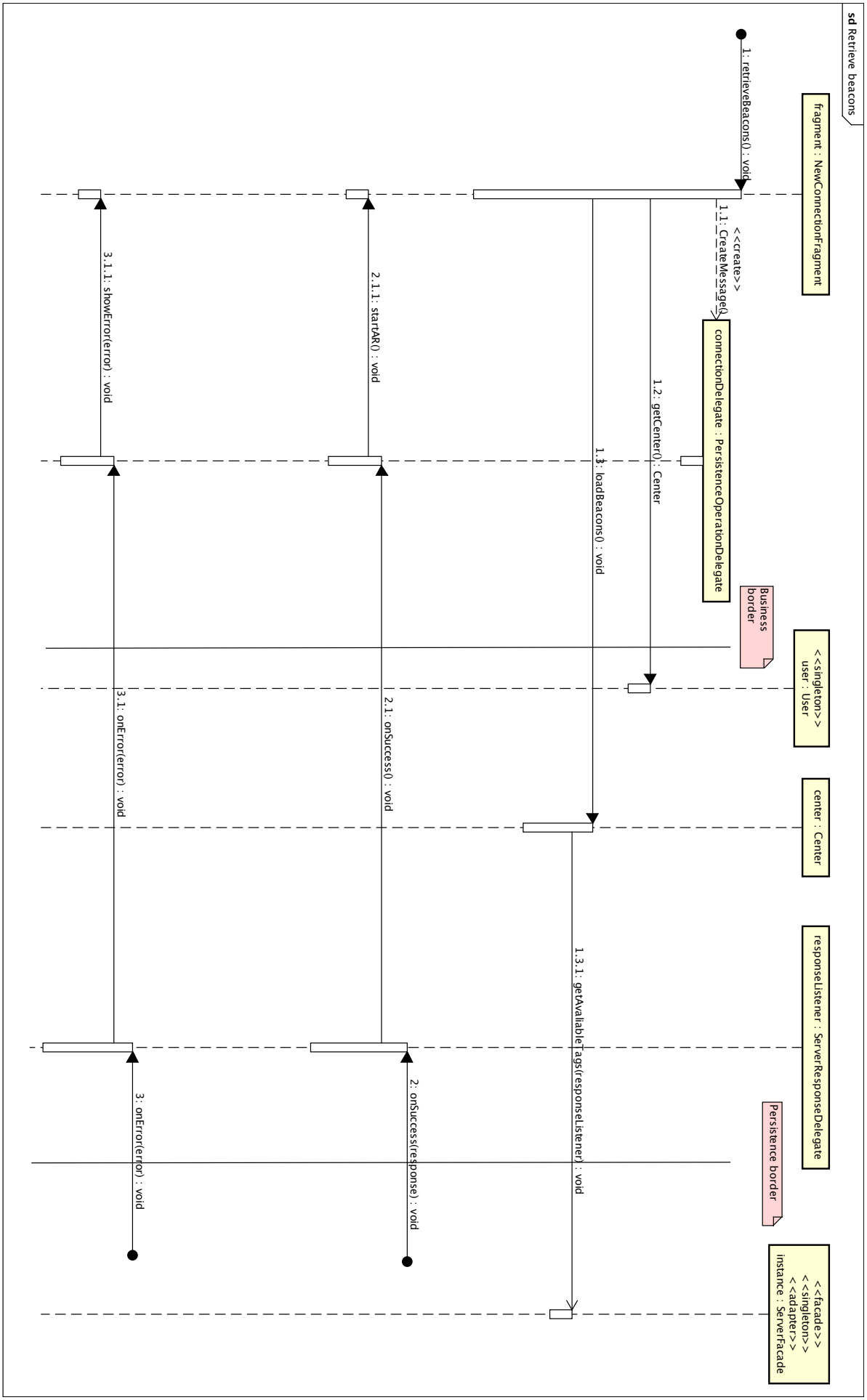


Figura 5.27: Diagrama de secuencia para la obtención de los puntos de interés.

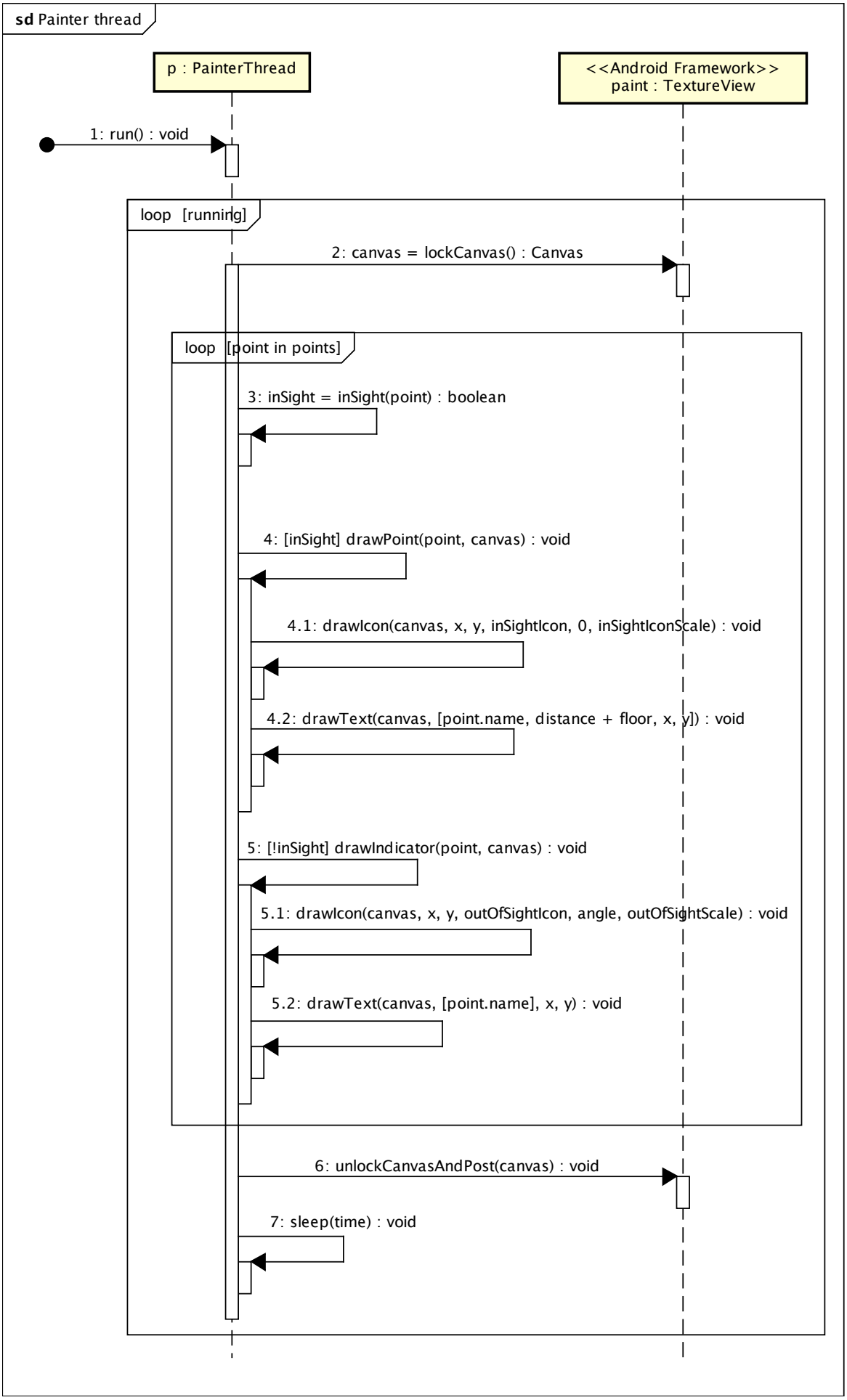


Figura 5.28: Diagrama de secuencia para el hilo que gestiona el dibujado en pantalla.

5.3. Diseño iOs

5.3.1. Campo de visión de la cámara

Aquí se tienen las mismas opciones que en Subsección 5.2.1, y al igual que en la versión para *Android* se ha decidido confiar en la API para obtener el campo de visión. Sin embargo, en *iOS* no hay posibilidad de obtener las características de la cámara, aunque se puede obtener directamente el campo de visión horizontal dada una sesión de captura, pero no hay ninguna forma de obtener el ángulo vertical de forma directa. Por suerte, una simple operación nos permitirá obtener dicho valor:

$$\alpha_{vertical} = 2 \arctan\left(\frac{H}{2} \times \frac{h}{w}\right)$$

Donde H es el ángulo horizontal, y w y h son la anchura y altura respectivamente.

De está forma, sabiendo el ratio de aspecto de la imagen podremos obtener el ángulo de visión vertical.

5.3.2. Dependencias entre capas

Las dependencias entre capas son:

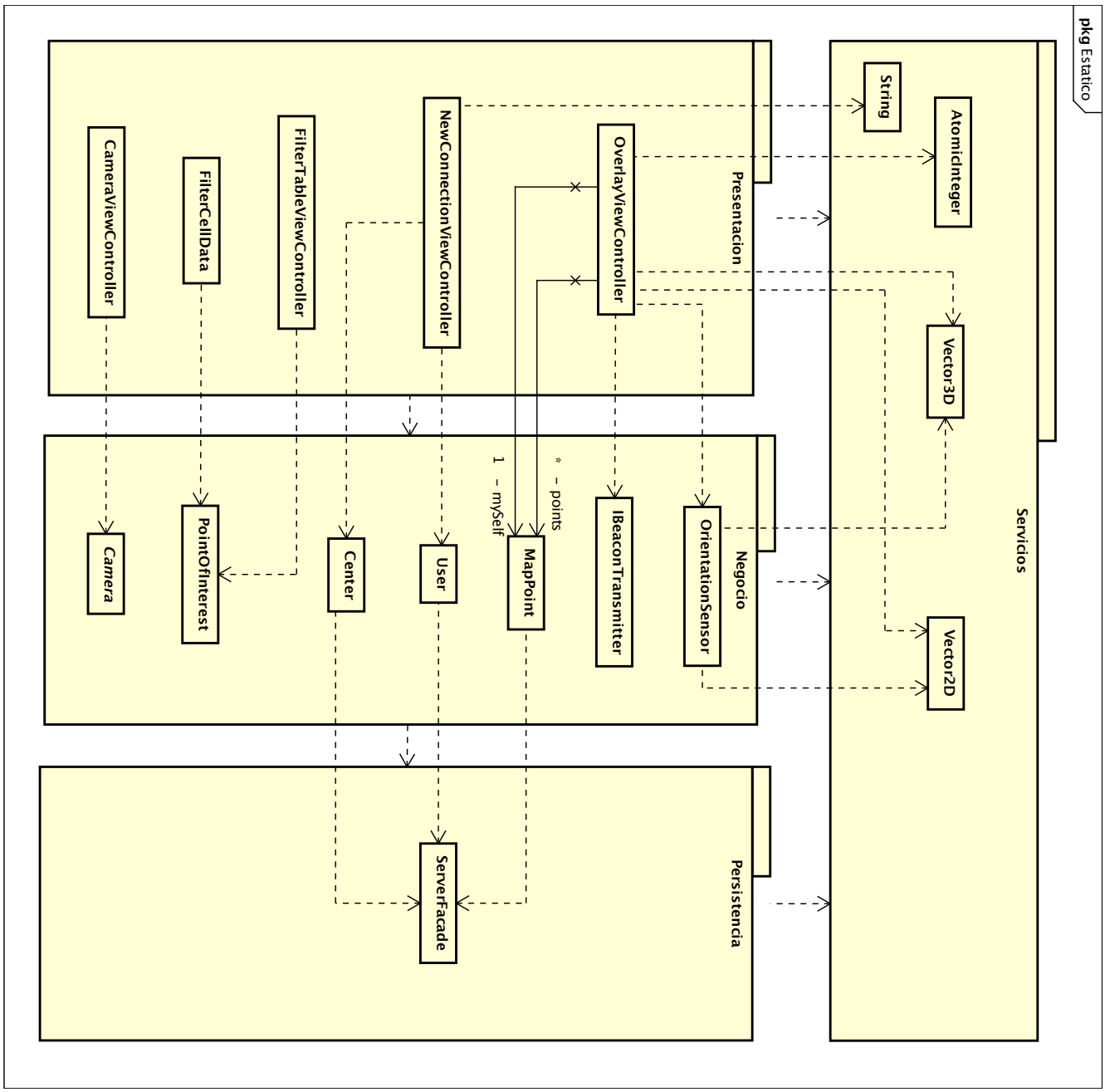


Figura 5.29: Diagrama de dependencias entre capas.

5.3.3. Diagrama de clases

Primero, las clases correspondientes a la arquitectura elegida, separadas en diagramas correspondientes a las distintas capas.

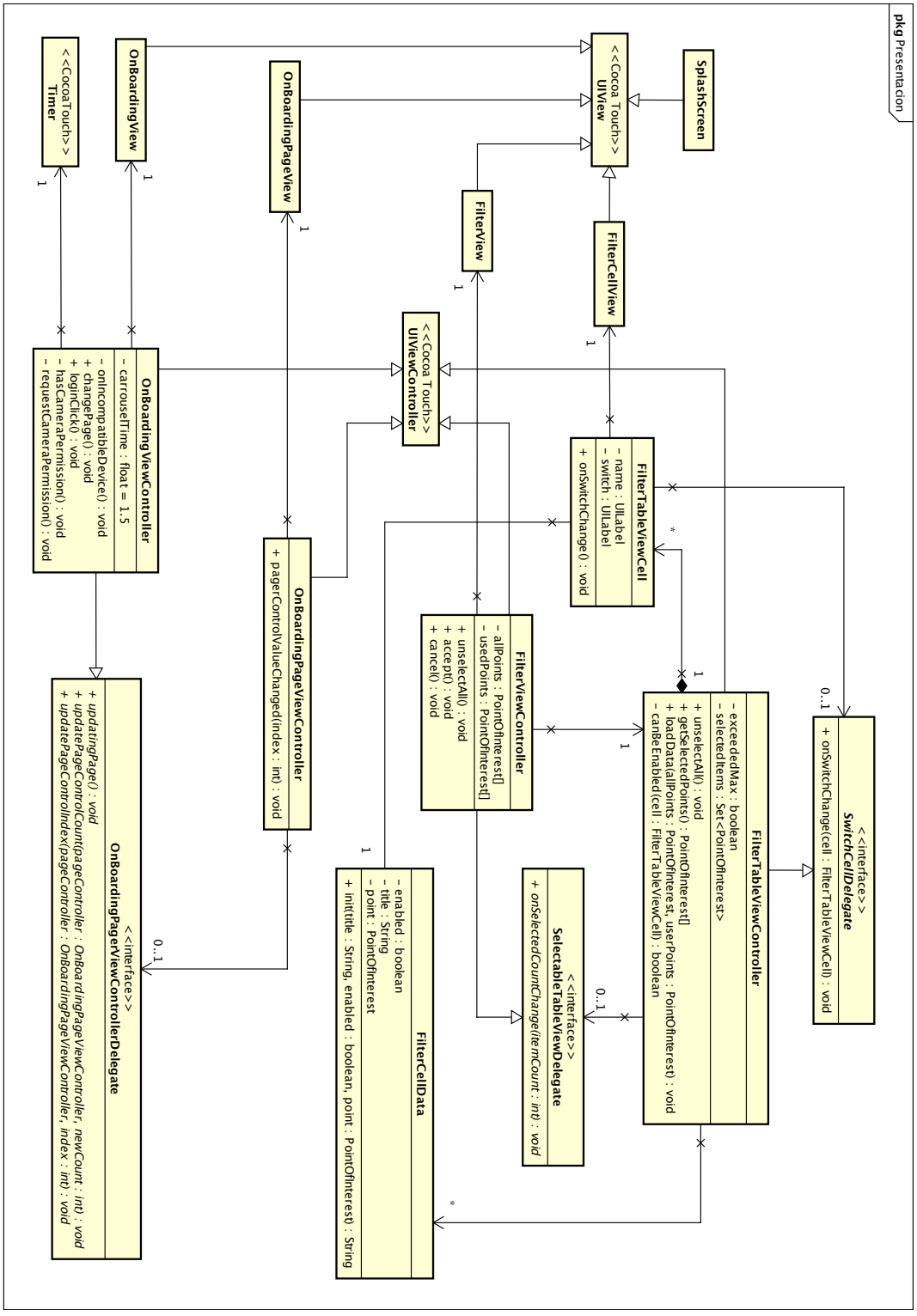


Figura 5.30: 1^{er} Diagrama de clases para la capa de presentación.

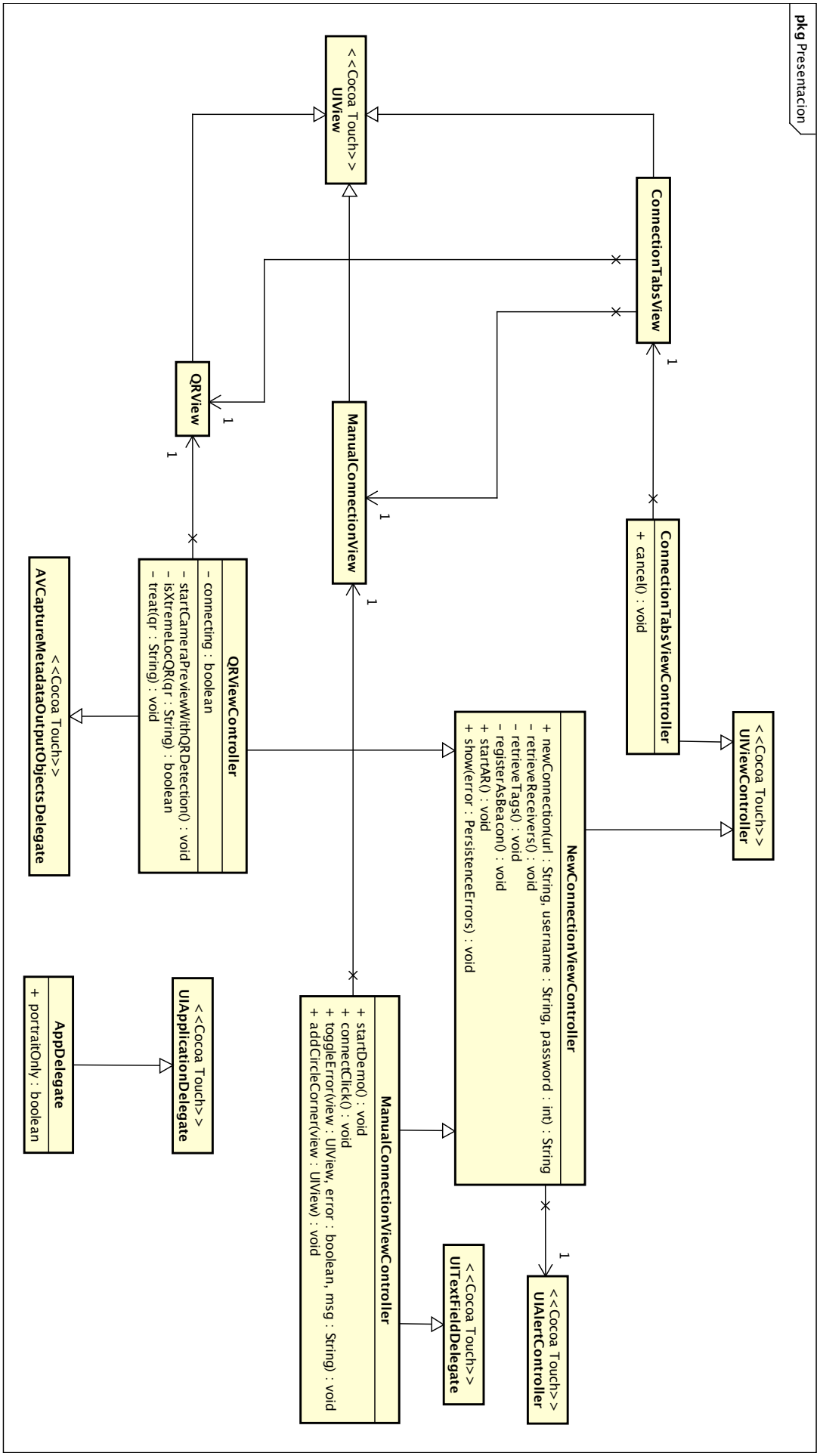


Figura 5.31: 2º Diagrama de clases para la capa de presentación.

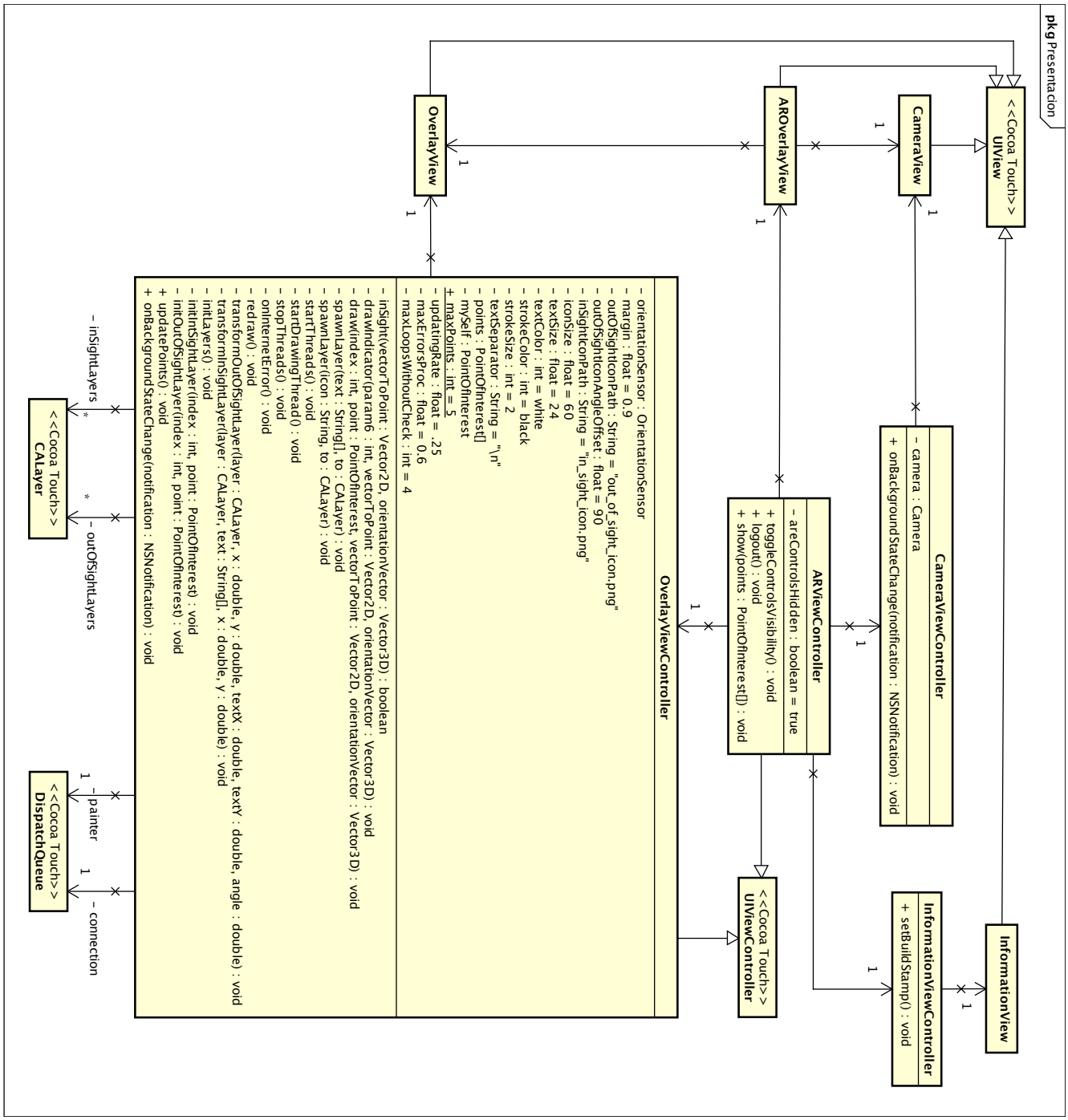


Figura 5.32: 3^{er} Diagrama de clases para la capa de presentación.

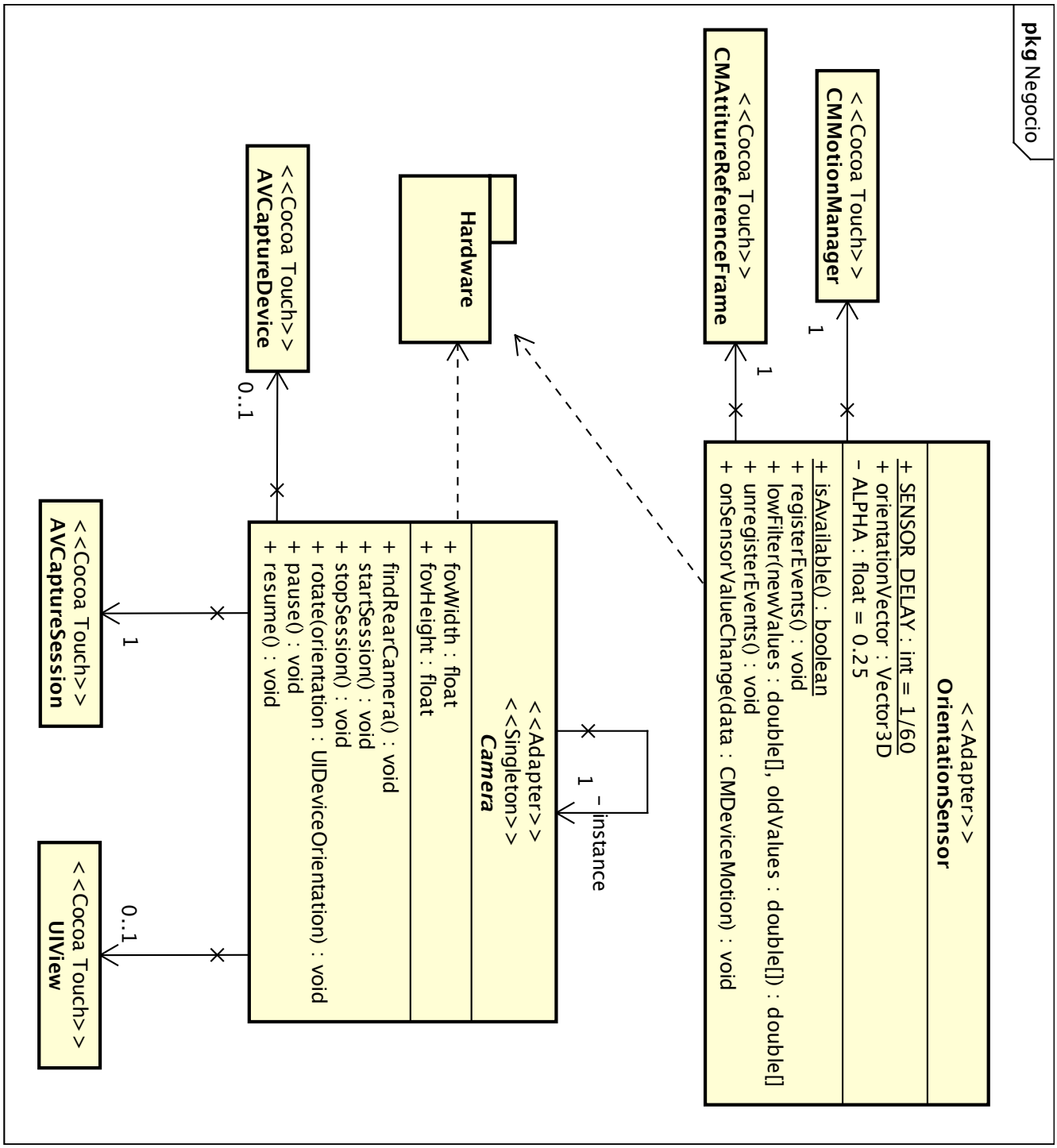


Figura 5.33: 1^{er} Diagrama de clases para la capa de negocio.

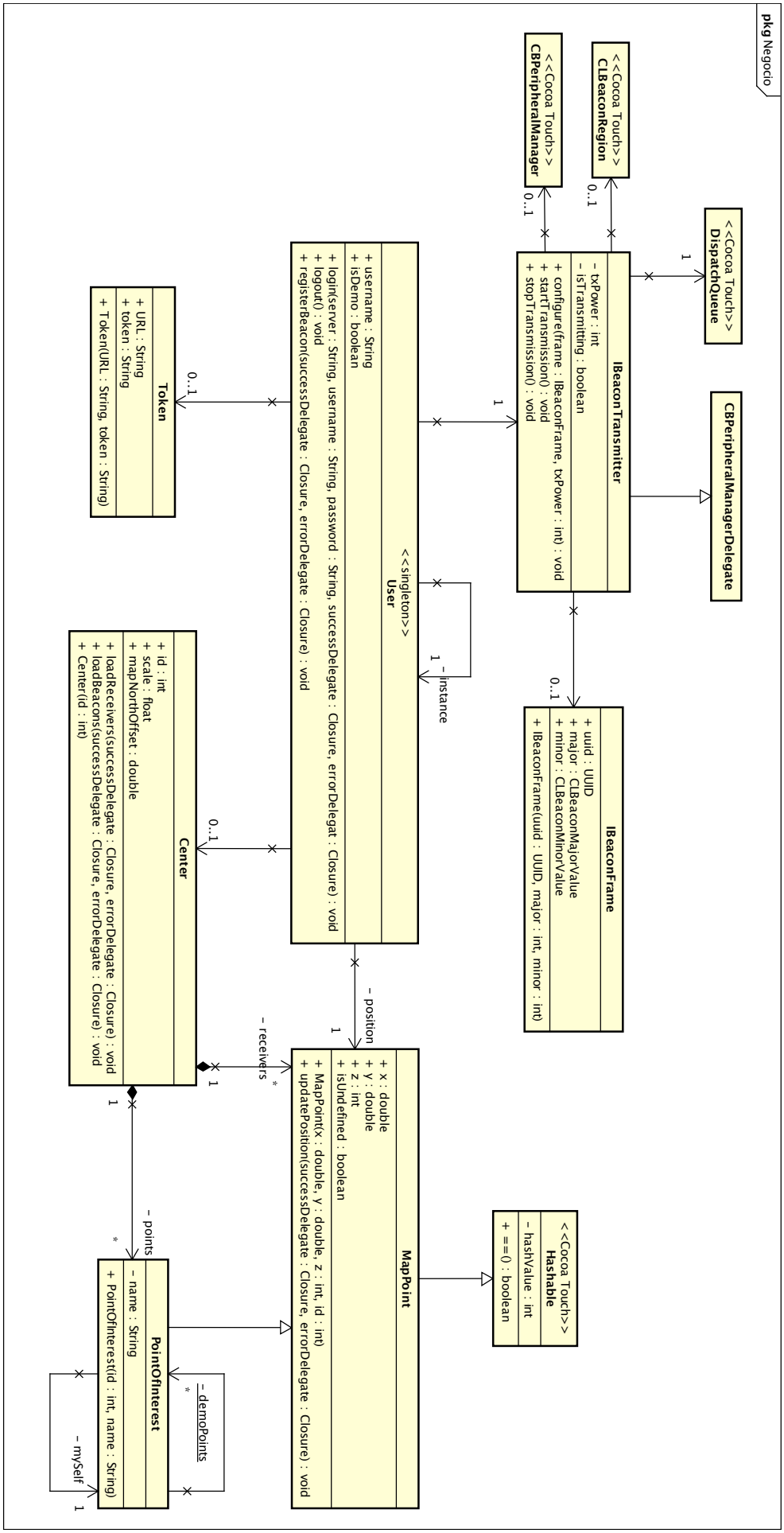


Figura 5.34: 2º Diagrama de clases para la capa de negocio.

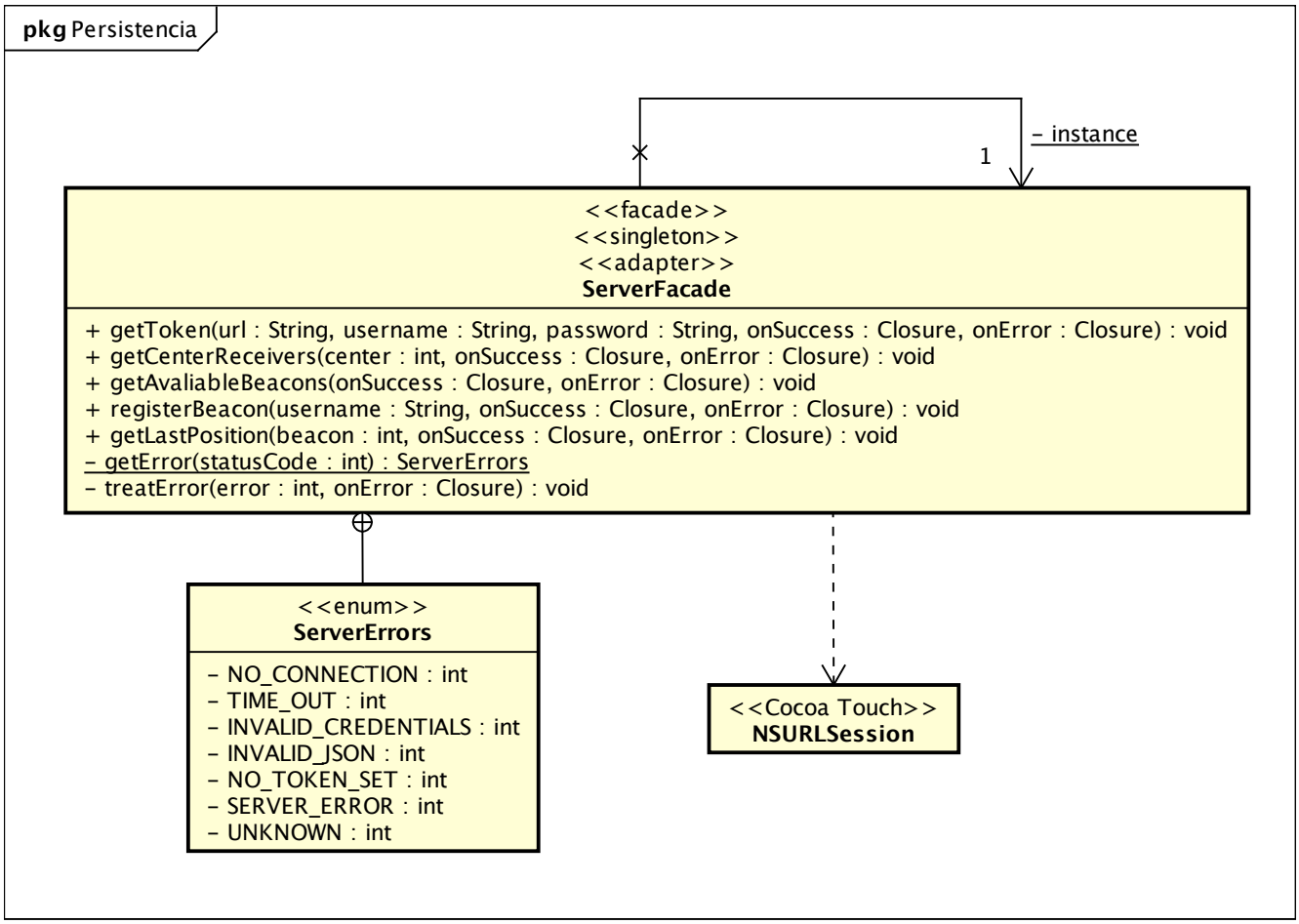


Figura 5.35: Diagrama de clases para la capa de persistencia.

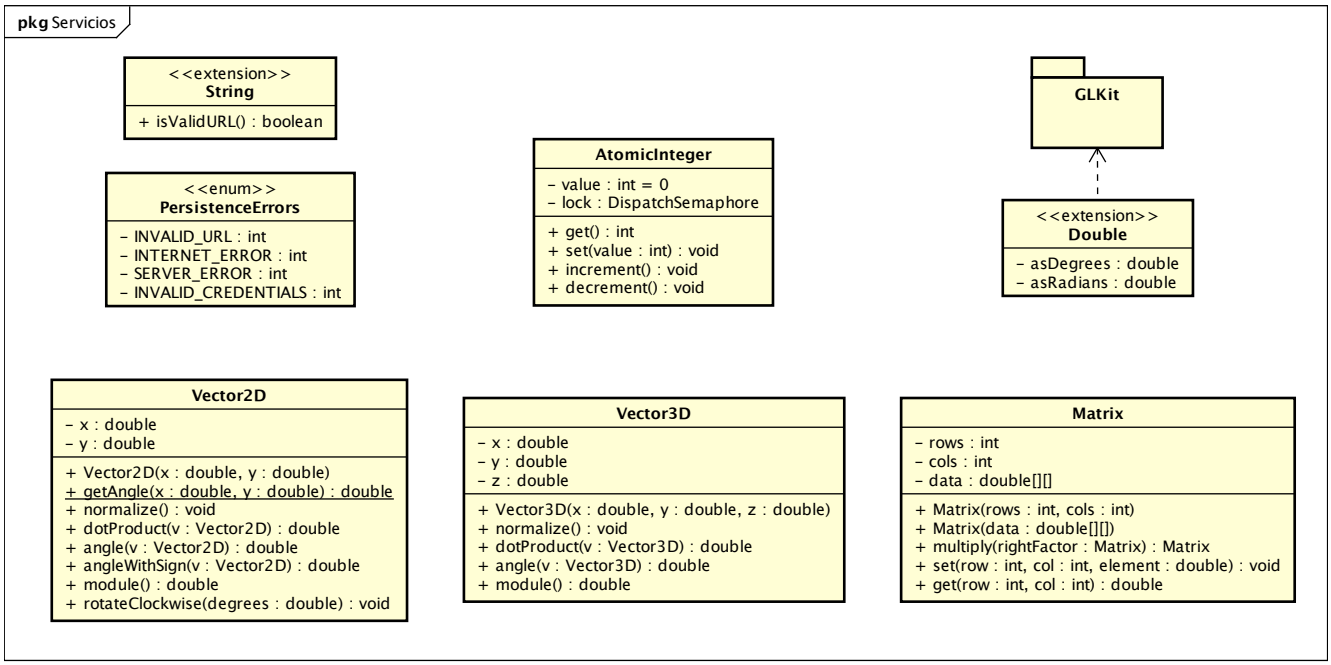


Figura 5.36: Diagrama de clases para la capa común de servicios.

5.3.4. Diagramas de secuencia

Antes de comenzar, hay que tener en cuenta el estilo que se ha seguido, al no haber un estándar UML para la representación de closures, se ha decidido crear una clase que las represente, y crear una nueva llamada a un método de ejecución en las closures creadas, para representar su llamada.

Resaltar también la no inclusión de todos los diagramas de secuencia posibles, mostrando solamente los más importantes, por el mismo motivo que en Subsección 5.2.4, la mayoría de los diagramas se corresponderían con la gestión del ciclo de vida de las vistas, eventos de interacción por parte del usuario y demás llamadas comunes a cualquier aplicación *iOS*.

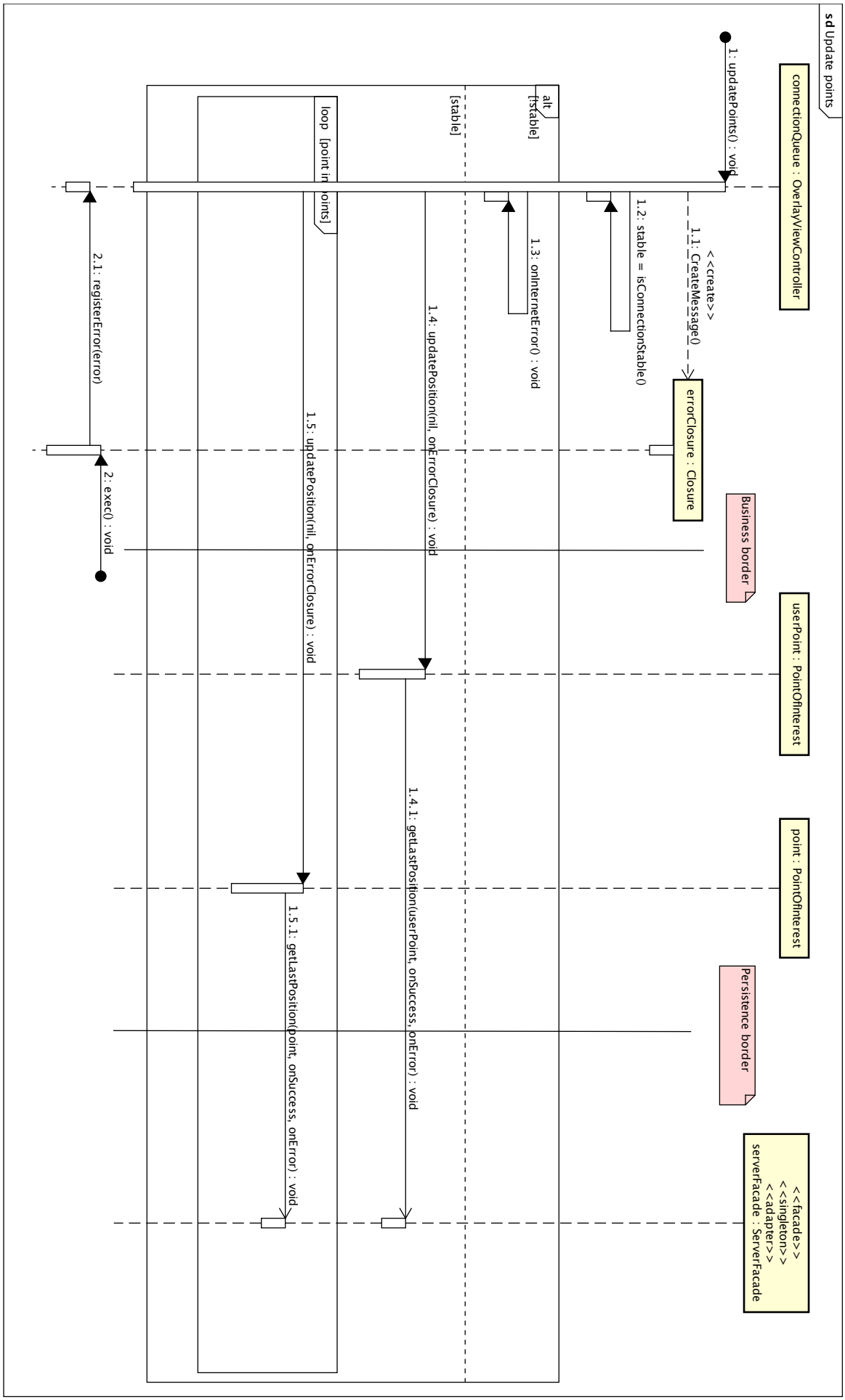


Figura 5.37: Diagrama de secuencia para el hilo que gestiona las peticiones de actualizaciones de los beacons.

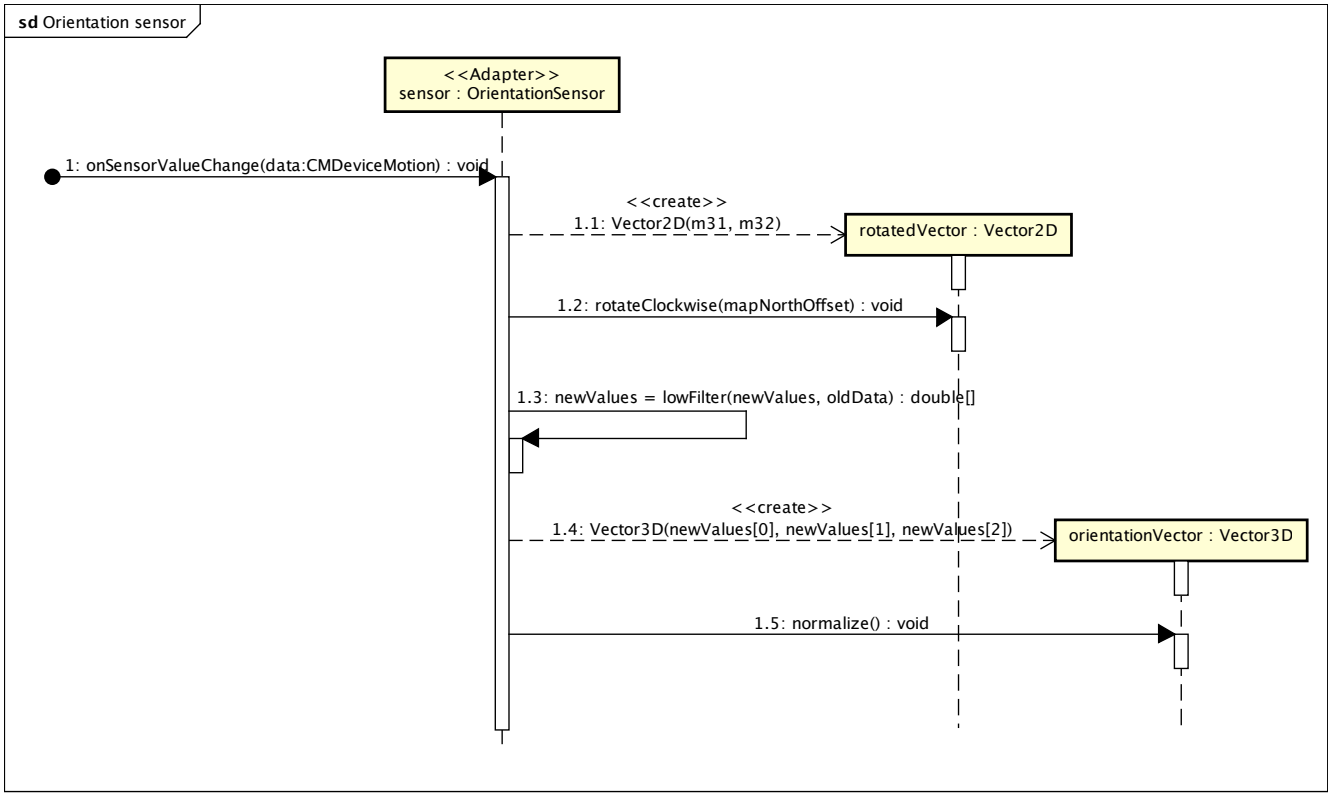


Figura 5.38: Diagrama de secuencia para la gestión de un cambio en la orientación del dispositivo.

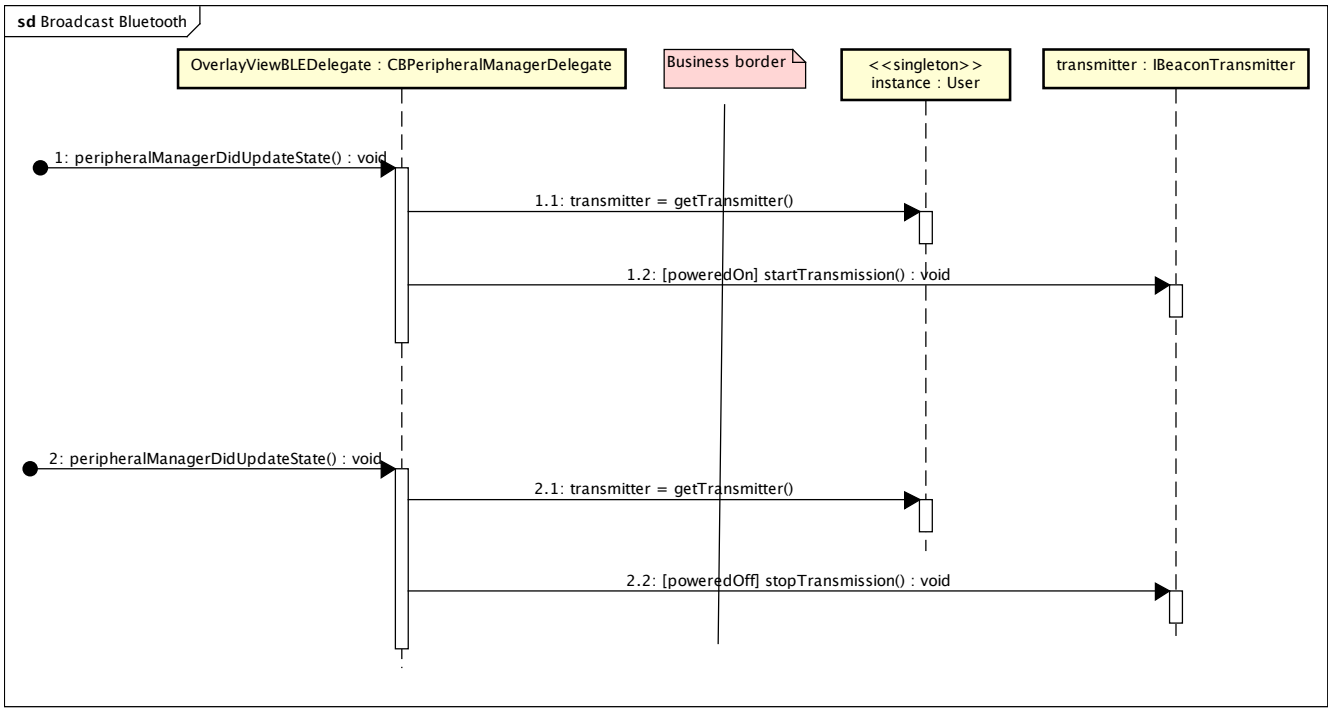


Figura 5.39: Diagrama de secuencia para el manejo de la señal bluetooth a emitir.

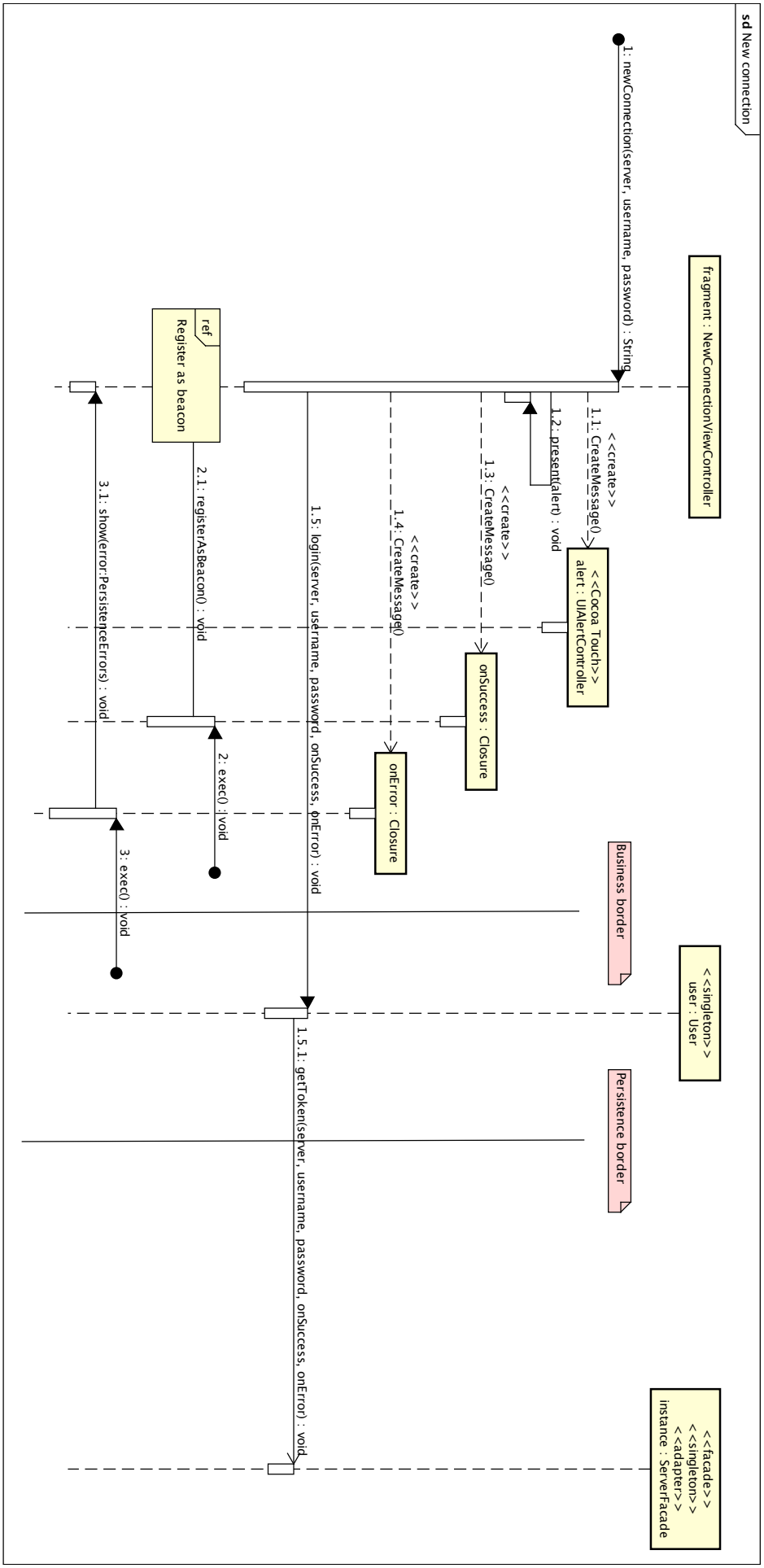


Figura 5.40: Diagrama de secuencia para una nueva conexión con el servidor.

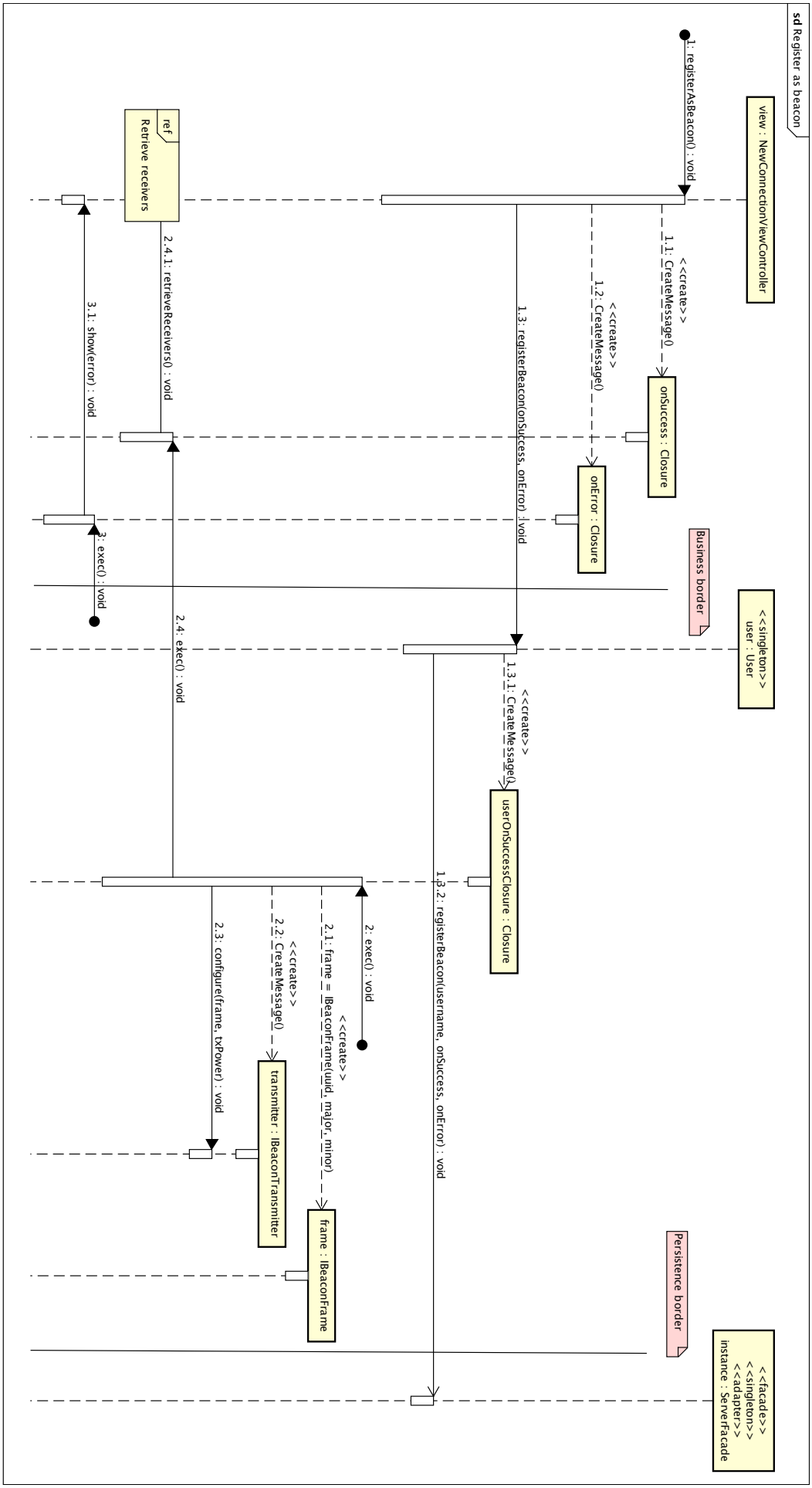


Figura 5.41: Diagrama de secuencia para el registro del dispositivo.

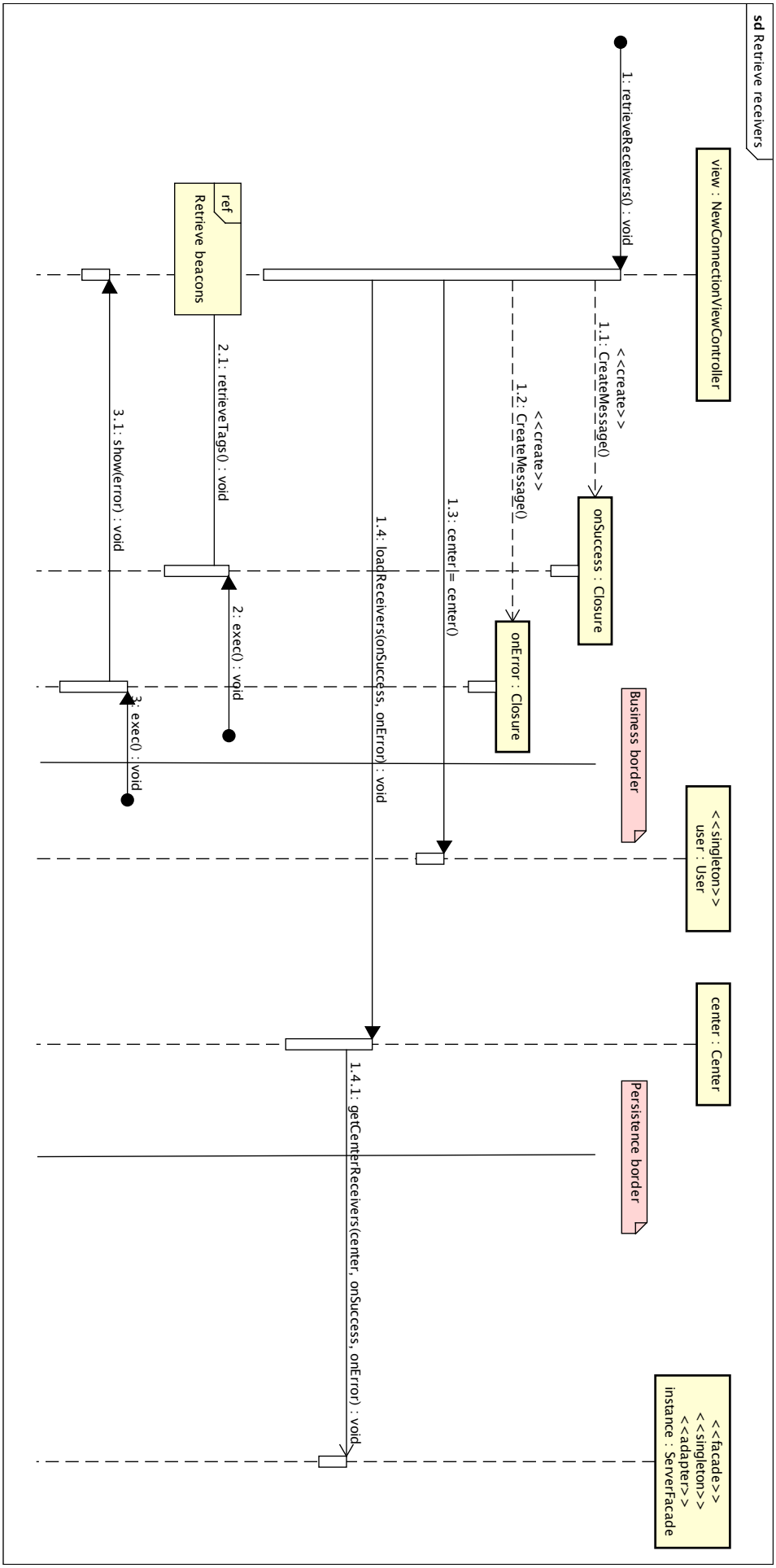


Figura 5.42: Diagrama de secuencia para la obtención de las antenas.

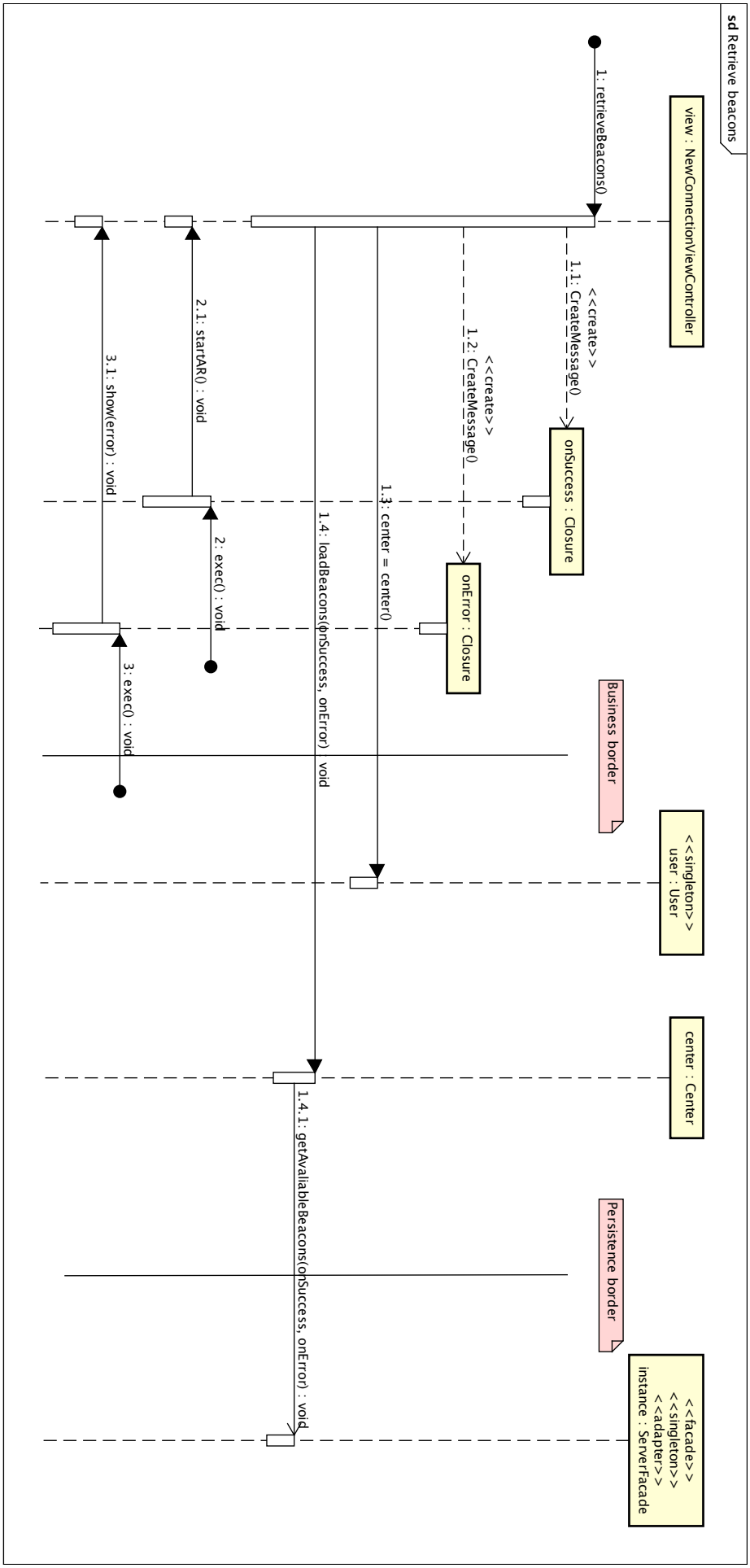


Figura 5.43: Diagrama de secuencia para la obtención de los puntos de interés.

Al contrario que el Framework de *Android*, *iOS* no permite el dibujado en pantalla por parte de otro hilo que no sea el principal, con la única excepción del dibujado de motores gráficos, cuyo tipo de vista a utilizar ha sido estudiado, y rechazada al no disponer de funciones básicas de dibujado de imágenes planas y texto.Por lo que la forma de trabajo cambia respecto de la versión *Android*.

Se crearán las capas de la vista de antemano, con sus respectivos iconos, siguiendo el patrón *Object Pool*. Las dos motivaciones de no destruir y crear nuevas capas con cada nuevo dibujado son, por un lado, la mejora de rendimiento que esto supone, y por otro el parpadeo visual que se produciría en cada ciclo de dibujado.

Una *Dispatch Queue*, que se ejecutará cada cierto tiempo, realizará los cálculos necesarios para conocer que capas modificar, y que nuevas propiedades han de tener estás capas, cada vez que se haya completado el calculo de una capa, se solicitará al hilo principal que modifique aquella correspondiente a los datos generados, colocando esta operación en la cola de operaciones del hilo principal de está forma, los cálculos pesados no bloquearán el hilo de la interfaz de usuario, y tampoco se bloqueará la cola de dibujado, ya que podremos solicitar estos dibujados de forma asíncrona, ya que el formato FIFO nos asegura su correcto orden de ejecución, permitiendo al hilo de dibujado seguir trabajando de forma libre, sin necesidad de sincronismos con la cola de operaciones principal de la App.

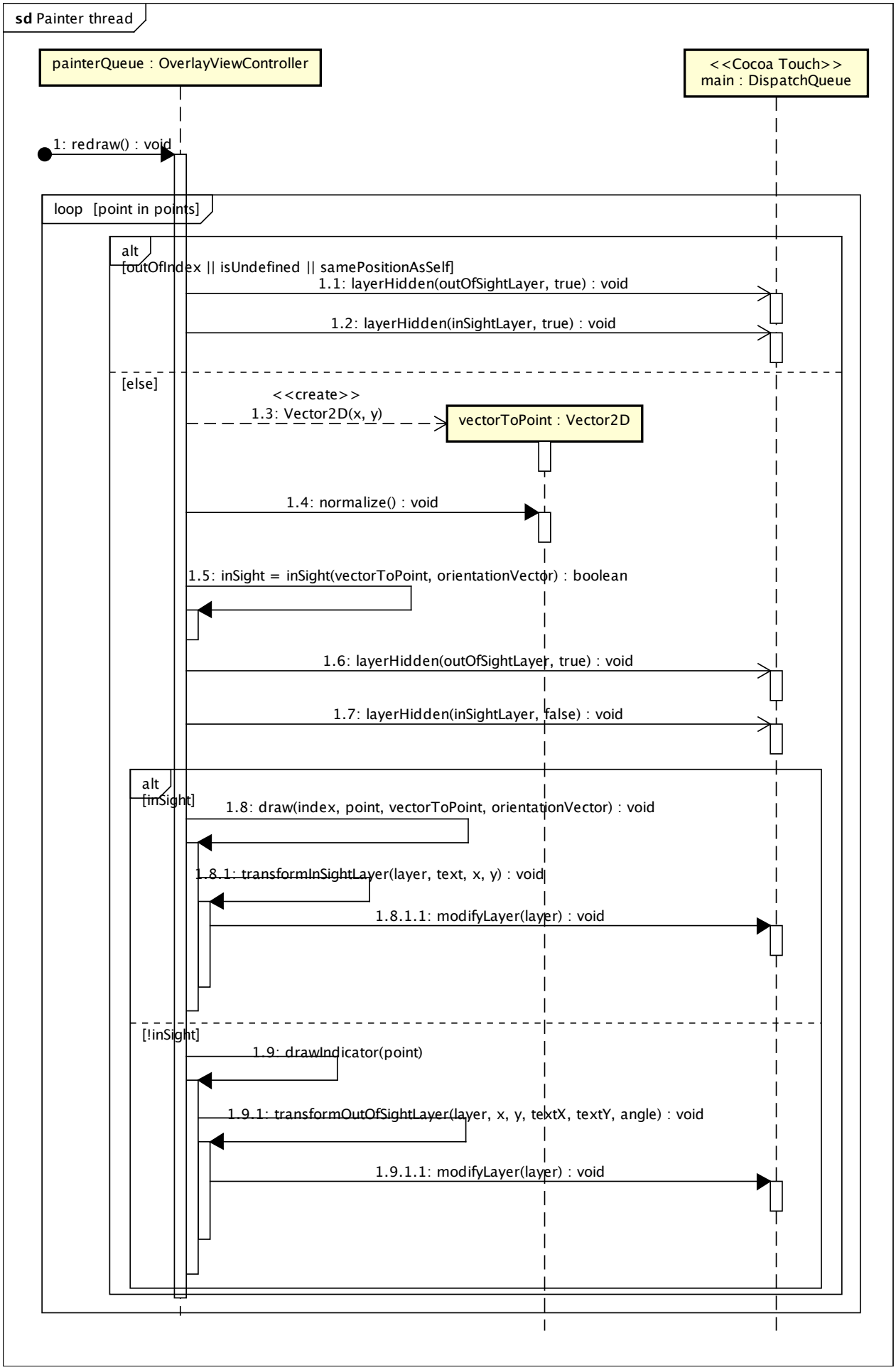


Figura 5.44: Diagrama de secuencia para el hilo que se encarga del dibujado en pantalla.

Capítulo 6

Implementación

Este capítulo aborda cuestiones cómo que entornos de programación se han utilizado, que Frameworks de apoyo han sido usados, clarificaciones sobre el código generado, etc.

6.1. Aspectos relativos al código

A la hora de la escritura del código del sistema, se han seguido las siguientes lineas de diseño.

1. Todos aquellos elementos, que tengan un carácter literal y estático, han sido insertados como constantes, para su fácil modificación posterior, y evitar de esta forma errores en la programación, además de habilitar optimizaciones del compilador.

2. Los hilos que se encargarán del trabajo, serán insertados como clases internas a los controladores de vista, para así mejorar el rendimiento y no tener que realizar llamadas del tipo *getAtributo()*.

3. Cuando se necesitan acceder a atributos, como posiciones u orientaciones, se creará una copia de dicho valor, para evitar condiciones de carrera y datos inconsistentes.

4. No se ha tenido en cuenta el impacto en batería, todo esta centrado en obtener el máximo rendimiento.

5. Se han ejecutado los analizadores de código de ambas plataformas, puliendo así bugs o aspectos inapreciables a simple vista.

6.2. Servicios para el desarrollador

En ambas versiones se ha incluido *Fabric* [36], un framework externo que se encarga de proporcionar ciertos servicios a los desarrolladores.

En este caso, se ha implementado *crashlytics* y *answers*, dos sub-sistemas que permiten la obtención de estadísticas relativas al uso de la aplicación e informes sobre los fallos que se han producido en el código de una aplicación durante su uso, respectivamente. Se podrían implementar más servicios que *Fabric* ofrece en el futuro, aunque estos dos sistemas son los considerados como esenciales, y los cuales han ayudado durante el desarrollo de *Skywalker*.

Valdría la pena considerar el cambio a *Firebase* [21] de *Google*, aunque este sea de pago, por los servicios que este presenta, teniendo en cuenta la adquisición por parte de *Google* de *Fabric*, ya que esta compra podría modificar el por venir de *Fabric*, afectando a *Skywalker*.

6.3. Internalización de la interfaz

Skywalker se encuentra preparado para su traducción a distintos idiomas y culturas, habiendo utilizado medidas relativas al comienzo y fin de los elementos de la interfaz, con lo que aquellas culturas cuya lectura se produzca de derecha a izquierda deberían ver una interfaz concorde a este hecho.

Como lenguas iniciales, se han implementado inglés y castellano, habiendo separado todas aquellos elementos susceptibles de ser traducidos del código hacía los ficheros correspondientes a cada plataforma, favoreciendo así la labor de traducción de la aplicación.

6.4. Android

Para *Android*, *Skywalker* ha sido implementado utilizando Android Studio, con una versión objetivo de *Android* 4.3 o superior. Utilizando como sistema de automatización de la compilación *Gradle*. En este momento, todas las bibliotecas utilizadas están incluidas en su última versión, y durante posteriores modificaciones del código deberían mantenerse actualizadas, salvo causa mayor.

6.4.1. Implementación de la UI

Durante el desarrollo, *Google* introdujo un nuevo tipo de *ViewGroup*, denominado *ConstraintLayout*, el cual permite, de forma similar a *AutoLayout* en *iOS*, colocar los elementos de la interfaz con posiciones relativas a otros elementos. Se ha aprovechado esta nueva posibilidad en aquellas interfaces donde tenía sentido utilizarla, y debería potenciarse su uso allá donde encaje en el futuro.

Asimismo, las bibliotecas de diseño de *Google* han sido utilizadas, con el fin de implementar *Material Design* de manera sencilla y favoreciendo un sencillo mantenimiento. Así, en caso de modificaciones en la especificación de estas guías de diseño, con un simple cambio de biblioteca externa debería bastar para implementar estos cambios en *Skywalker*.

Por último, se ha incluido la librería *Tap Target View* [25], de código abierto, para la implementación de los *Tap Target* considerados en las guías de diseño de *Material Design*, debido a su facilidad de uso.

6.4.2. Eventos en las actividades y fragmentos

Uno de los aspectos fundamentales en cualquiera App para *Android* es el manejo de los eventos de las actividades y fragmentos, ignorarlos podría provocar consumo excesivo de la batería, errores en el estado de la App e incluso su detención completa.

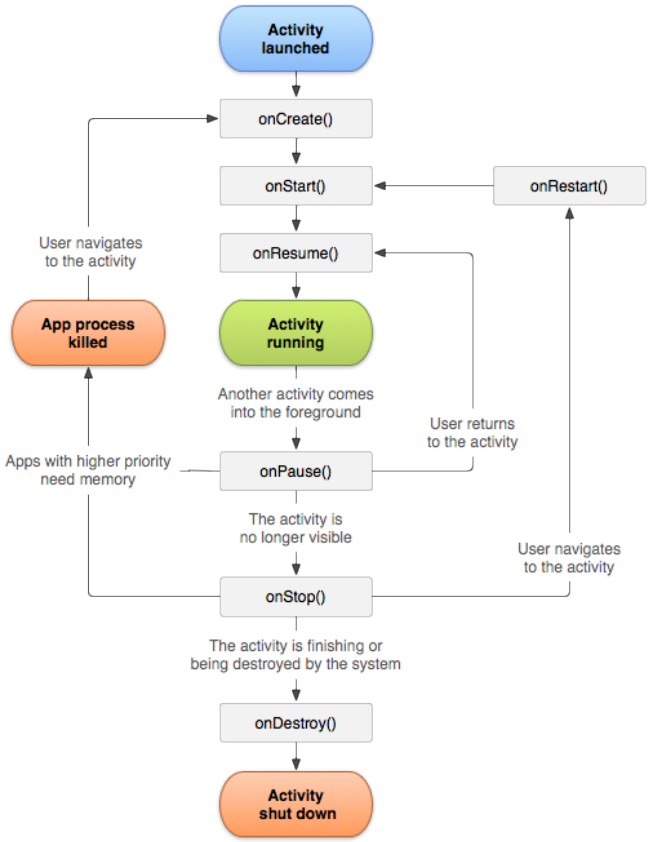


Figura 6.1: Ciclo de vida de una actividad

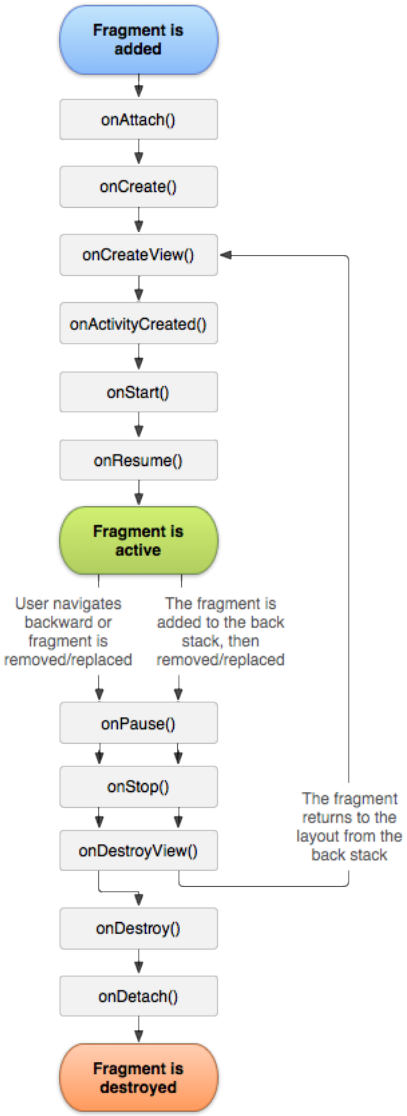


Figura 6.2: Ciclo de vida de un fragmento

Para *Skywalker*, se han aprovechado los eventos de parada para la detención de los sensores, cámara, envío de paquetes Bluetooth, e hilos de dibujado y peticiones; a su vez los eventos de continuación para reanudar los trabajos anteriormente descritos.

Mención especial a la actividad de realidad aumentada, ya que utilizando la implementación por defecto, se producía el siguiente ciclo, tanto en la primera creación como en los cambios en la orientación de la interfaz:

1. Creación la vista.
2. Apertura de la cámara.
3. Inicio de una sesión en la cámara.
4. Cambio en la orientación.
5. Inicio de la destrucción de la vista.
6. Detención de la sesión de la cámara.
7. Cierre de la cámara.
8. Regreso al primer paso del ciclo.

Este ciclo, funcionaba perfectamente, sin embargo producía un retraso en el giro de la interfaz prolongado, llegando a irritar a los usuarios que probaron la aplicación.
Por lo que la implementación final no produce la destrucción de la vista en un cambio de orientación, cambiando el ciclo anterior por el siguiente:

1. Creación la vista.
2. Apertura de la cámara.
3. Inicio de una sesión en la cámara.
4. Cambio en la orientación.
5. Detención de la sesión de la cámara.
6. Configuración de la sesión de la cámara.
7. Inicio de una sesión en la cámara.
8. Regreso al paso 4.

Aunque el número de pasos intermedios no haya cambiado, la no destrucción de la vista, con su posterior creación, ha mejorado el rendimiento notablemente, produciéndose un giro de la vista más rápido, con una parada de la cámara inapreciable por parte del usuario.

6.4.3. Códigos QR

Los servicios de Google Play son necesarios para el escaneo de códigos QR, ya que se ha decidido implementar esta funcionalidad mediante la API Vision [22]. Esta API permite el reconocimiento de distintos elementos, tales como códigos QR, sonrisas en fotografías, texto. Además es facil de integrar con la cámara del dispositivo.

Sin embargo, esto no ha sido posible, debido a que dicha integración necesita del uso del elemento *Surface View* en la interfaz de usuario, cuya implementación se ha realizado con *Texture Views*, ya que esta permite ser modificada libremente, sin producir artefactos extraños, necesario por la ubicación del fragmento de códigos QR en pestañas movibles.

Para poder integrar la cámara de esta forma con la API, se ha utilizado un hilo extra, el cual, con cada ciclo de su ejecución, recoge el bitmap de lo que la cámara ve en dicho momento, crea un marco nuevo, y se lo pasa a la API para su procesamiento.

6.4.4. Peticiones a XtremeLoc

Debido a las recomendaciones de *Google*, se ha utilizado la biblioteca externa *Volley* [24] para las peticiones al servidor, los motivos, por los que se acomoda perfectamente al proyecto, son:

1. Habilita el uso de caché para peticiones vía red.
2. Concurrencia transparente para el programador.
3. Soporte para peticiones JSON.
4. Facilidad de uso.

Importante destacar el uso de cabeceras de peticiones HTTP personalizadas, ya que *XtremeLoc* solicita un Token de autenticación para responder a las peticiones, se ha incluido un campo como el siguiente:

```
Authorization: Bearer 1234Token
```

6.4.5. Señal iBeacon

Aunque *Android* permite el envío de señales Bluetooth de baja energía personalizadas, su construcción no es trivial, ya que no se pueden acceder a las posiciones del array de datos manualmente, sino que hay ciertas llamadas para colocar los datos a partir de cierta posición ya definida, con lo que construir un paquete con el formato correcto llevaría a diseñar un parser personalizado que trabaje con las llamadas permitidas por el framework.

Por suerte, existe una librería externa open-source, llamada *Android Beacon Library* [28], cuya implementación posee su propio parser, el cual admite cualquier formato personalizado, incluso aquellos no estandarizados, además, la librería se comunica con las clases correspondientes del framework para llevar a cabo la transmisión, por lo que su uso es necesario para esta versión, simplificando, en gran medida, las tareas a realizar.

6.4.6. Compatibilidad del dispositivo y eventos de conexiones

Los usuarios esperan que la aplicación funcione correctamente en sus dispositivos, y muchos de ellos no miran los requisitos mínimos a la hora de instalar nuevas Apps, por lo que se ha incluido un mecanismo de detección de compatibilidad al iniciar la App, si el dispositivo sobre el que se ejecuta no fuera compatible, se avisaría al usuario y la aplicación terminaría.

Por otra parte, a la hora de iniciar la visión de realidad aumentada, se ha implementado la activación automática de la conexión Bluetooth, y durante su funcionamiento, se ha incluido un *Broadcast Listener* [18] para atender a eventos de activación y desactivación de dicha conexión. Por suerte, el framework permite la activación de la conexión mediante código, así que el usuario no necesita salir de la aplicación.

La conexión con el servidor no necesita de un *Broadcast Listener*, sino que son los propios errores devueltos por *Volley* los que son utilizados para detectar desconexiones, informar al usuario y volver a la pantalla de inicio de sesión.

6.4.7. Sensor no calibrado

Como ya se dijo en el Capítulo 5, si el sensor de orientación cambia su precisión se producirá una llamada al callback correspondiente, *Android* permite diferenciar 4 niveles de precisión:

1. Desconocido.
2. Precisión baja.
3. Precisión media.
4. Precisión alta.

Dado que *Skywalker* necesita la máxima precisión posible, ante cualquier precisión menor a la alta se mostrará un dialogo al usuario, con instrucciones de como calibrar su dispositivo, una vez el sensor note que ha sido calibrado, y haya llamado al callback con precisión alta, dicho dialogo desaparecerá.

6.5. iOS

En el caso de *iOS*, Skywalker ha sido implementado utilizando XCode, con una versión objetivo de *iOS* 8.0 o superior. En esta versión, no han sido necesarios Frameworks adicionales ya que toda la funcionalidad necesaria ya está integrada con el sistema operativo.

6.5.1. Cambios en la orientación

iOS permite, desde el fichero descriptor de la App, modificar que orientaciones de pantalla se soportan en toda la App, el problema, es que queremos permitir el modo horizontal solamente en la visión de realidad aumentada, siendo el modo vertical el único permitido para el resto de vistas, la solución que se ha implementado consiste en modificar el método correspondiente del delegado de la aplicación para que, devuelva las orientaciones soportadas solicitadas por cada controlador de vista. De este modo, un controlador que quiera permitir cualquier modo de orientación, simplemente deberá acceder al delegado de la App, y modificar un atributo, cuando dicho controlador vaya a desaparecer, utilizará la misma metodología para volver al comportamiento por defecto.

6.5.2. Ciclos de vida de las vistas y de la aplicación

Como las actividades en *Android*, las vistas de *Cocoa Touch* tienen su propio ciclo de vida, el cual se aprovecha para terminar o iniciar los envíos de paquetes bluetooth, la cámara, los sensores y los hilos de

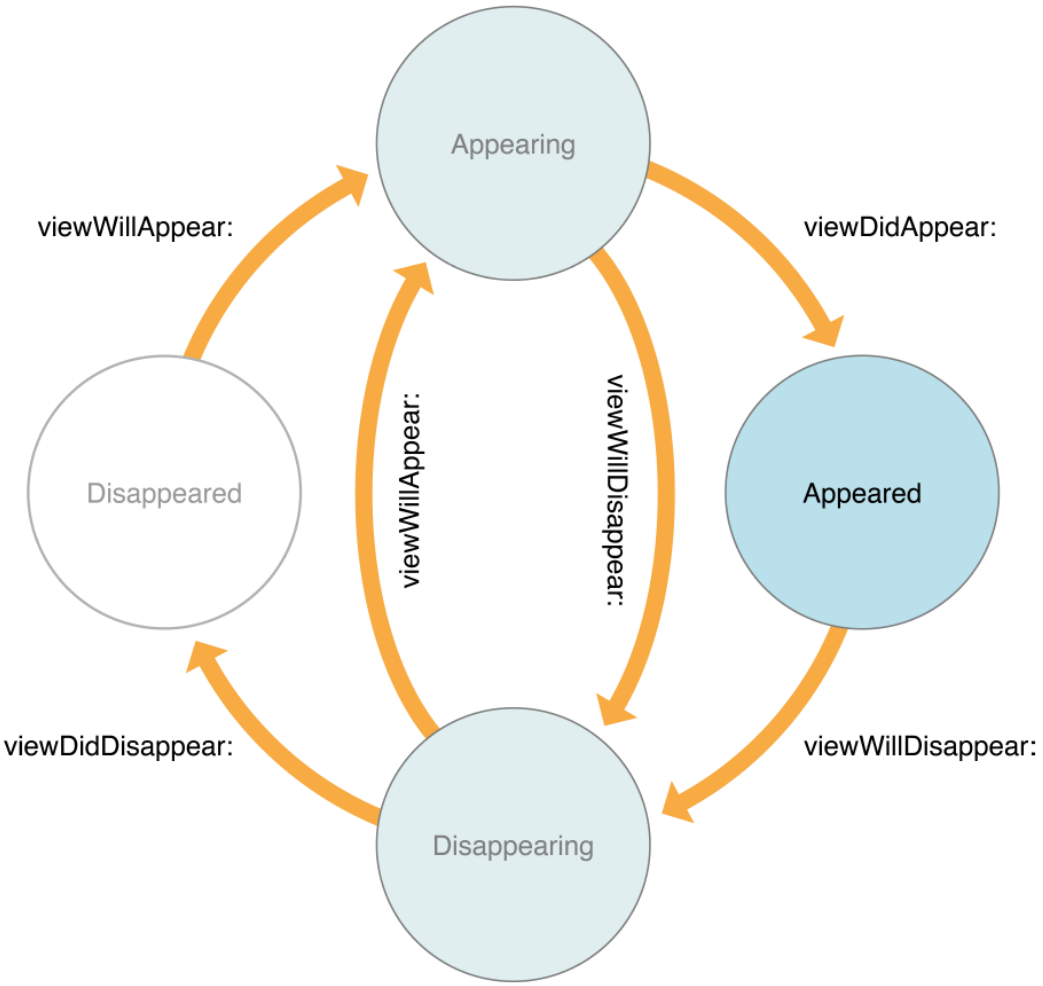


Figura 6.3: Ciclo de vida de una vista

dibujado y de peticiones a *XtremeLoc*.

Es importante destacar que la cámara, debe modificar la capa donde va a depositar su imagen, cada vez que una vista haya cambiado el tamaño de sus hijas, con el callback *viewDidLayoutSubviews*, sino podría producirse un tamaño incorrecto en la vista de la cámara que el usuario percibe.

Además, ciertas vistas se suscriben al centro de notificaciones de *iOS* para recibir eventos de cambio del estado de la aplicación, de esta manera se reanudan conexiones, cámara, sensores y/o Bluetooth cuando la aplicación pasa del segundo plano al primer plano de ejecución. Este añadido es debido a que el ciclo de una vista no es ejecutado cuando el usuario pulsa el botón de inicio, abre el panel de control o entra en los ajustes del sistema, por nombrar algunos eventos que pueden producir este efecto.

6.5.3. Compatibilidad del dispositivo y eventos de conexiones

Al igual que la versión para *Android*, se ha integrado una revisión de la compatibilidad del dispositivo en uso para ver si cumple los requisitos de funcionamiento, sin embargo, *Apple* no implementa ningún mecanismo para salir de una aplicación, salvo que se produzca una excepción, por lo que si no es posible utilizar la App en dicho dispositivo, se informará al usuario y se deshabilitará la opción de iniciar sesión, previniendo el uso de la App.

Por otra parte, los errores en la conexión con el servidor se tratan de igual forma que en la versión para *Android*, los errores proporcionados por *NSURLSession* serán los que se utilizan para cerrar la sesión

actual y volver a la pantalla de inicio de sesión.

Por último, *iOS* no permite la activación de conexiones Bluetooth si no es por parte del usuario, por lo que se ha implementado un delegado del gestor Bluetooth en la vista de realidad aumentada, implementando la llamada correspondiente para tratar los cambios en el estado del modulo Bluetooth, y en el caso de necesitarlo, solicitar al usuario una activación del Bluetooth, llevándole directamente a la pantalla de ajustes Bluetooth.

6.5.4. Sensor no calibrado

Como añadido de la llamada al callback correspondiente, *iOS* permite modificar un atributo del gestor del sensor, para que, si fuera necesaria la calibración del dispositivo, sea el propio sistema operativo quien presentará al usuario una interfaz de calibración, cuya desaparición se produce al finalizar la calibración. Dicho atributo, ha sido activado para gestionar este tipo de evento.

Capítulo 7

Pruebas

Para este desarrollo se han realizado dos tipos de pruebas, Pruebas sintéticas y Pruebas en dispositivos.

7.1. Pruebas sintéticas

Para las pruebas sintéticas, se han realizado pruebas para la capa servicios, sin tener en cuenta la codificación real de las clases que componen esta capa. No se han realizado otro tipo de tests debido a la dificultad de su prueba, ya que una gran parte del código se utiliza para la interfaz de usuario. Además estas clases han sido desarrolladas con metodología TDD, ya que se conocía la funcionalidad requerida antes de su implementación. Es por ello que el único tipo de pruebas realizadas sea de tipo unitario y de caja negra.

Para *Android* se ha utilizado el Framwork de pruebas JUnit, para *iOS* ha sido utilizado el Framework XCTests. Ambos son los recomendados por los fabricantes de cada sistema, y aquellos que ellos mismos utilizan en sus desarrollos. Ambos sistemas han informado de un 100 % de cubrimiento del código en las pruebas realizadas.

7.1.1. Clase Matrix

Descripción	Construcción de una matriz utilizando un array bidimensional de números de tipo double.
Resultado esperado	Cada una de las celdas de la matriz contiene el valor del array utilizado en su construcción.
Resultado obtenido	El valor de cada celda de la matriz contiene el valor de la misma posición del array utilizado en su construcción.
Observaciones	Correcto.

Descripción	Multiplicación de dos matrices.
Resultado esperado	$\begin{pmatrix} 30 & 24 & 18 \\ 89 & 69 & 54 \\ 138 & 114 & 90 \end{pmatrix}$
Resultado obtenido	$\begin{pmatrix} 30 & 24 & 18 \\ 89 & 69 & 54 \\ 138 & 114 & 90 \end{pmatrix}$
Observaciones	Correcto.

Descripción	Multiplicación de dos matrices incompatibles.
Resultado esperado	Excepción al realizar la multiplicación.
Resultado obtenido	Excepción al realizar la multiplicación.
Observaciones	Correcto.

7.1.2. Clase Vector2D

Descripción	Construcción de un vector bidimensional mediante sus componentes X e Y.
Resultado esperado	Las componentes del vector construido tienen su valor correspondiente.
Resultado obtenido	Las componentes del vector construido tienen su valor correspondiente.
Observaciones	Correcto.

Descripción	Normalización de un vector.
Resultado esperado	$X = \frac{1}{\sqrt{2}} \quad Y = \frac{1}{\sqrt{2}}$
Resultado obtenido	$X = \frac{1}{\sqrt{2}} \quad Y = \frac{1}{\sqrt{2}}$
Observaciones	Correcto.

Descripción	Producto escalar de dos vectores.
Resultado esperado	-14
Resultado obtenido	-14
Observaciones	Correcto.

Descripción	Obtención del ángulo interno entre dos vectores para cada una de las pruebas realizadas.
Resultado esperado	Prueba A: 0º Prueba B: 90º Prueba C: 180º Prueba D: 90º
Resultado obtenido	Prueba A: 0º Prueba B: 90º Prueba C: 180º Prueba D: 90º
Observaciones	Correcto.

Descripción	Obtención del ángulo, en el sentido de las agujas del reloj, entre dos vectores para cada una de las pruebas realizadas.
Resultado esperado	Prueba A: 0º Prueba B: 90º Prueba C: -180º Prueba D: -90º
Resultado obtenido	Prueba A: 0º Prueba B: 90º Prueba C: -180º Prueba D: -90º
Observaciones	Correcto.

Descripción	Obtención del modulo de un vector.
Resultado esperado	Prueba A: $\sqrt{2}$ Prueba B: 1 Prueba C: 1
Resultado obtenido	Prueba A: $\sqrt{2}$ Prueba B: 1 Prueba C: 1
Observaciones	Correcto.

Descripción	Rotación de un vector, con un ángulo determinado, siguiendo el sentido de las agujas del reloj.
Resultado esperado	Prueba A: $X = 0 \;\; Y = -1$ Prueba B: $X = -1 \;\; Y = 0$ Prueba C: $X = 0 \;\; Y = 1$ Prueba D: $X = 0 \;\; Y = 1$ Prueba E: $X = -1 \;\; Y = 0$ Prueba F: $X = 0 \;\; Y = -1$ Prueba G: $X = \frac{1}{\sqrt{2}} \;\; Y = \frac{1}{\sqrt{2}}$
Resultado obtenido	Prueba A: $X = 0 \;\; Y = -1$ Prueba B: $X = -1 \;\; Y = 0$ Prueba C: $X = 0 \;\; Y = 1$ Prueba D: $X = 0 \;\; Y = 1$ Prueba E: $X = -1 \;\; Y = 0$ Prueba F: $X = 0 \;\; Y = -1$ Prueba G: $X = \frac{1}{\sqrt{2}} \;\; Y = \frac{1}{\sqrt{2}}$
Observaciones	Correcto.

Descripción	Obtención del ángulo, en el rango [0, 360], correspondiente al vector en dos dimensiones (1, 0), y a un vector externo.
Resultado esperado	Prueba A: 0^o Prueba B: 90^o Prueba C: 180^o Prueba D: 270^o Prueba E: 45^o Prueba F: 135^o Prueba G: 225^o Prueba H: 315^o
Resultado obtenido	Prueba A: 0^o Prueba B: 90^o Prueba C: 180^o Prueba D: 270^o Prueba E: 45^o Prueba F: 135^o Prueba G: 225^o Prueba H: 315^o
Observaciones	Correcto.

7.1.3. Clase Vector3D

Descripción	Construcción de un vector tridimensional mediante sus componentes X, Y y Z.
Resultado esperado	Las componentes del vector construido tienen su valor correspondiente.
Resultado obtenido	Las componentes del vector construido tienen su valor correspondiente.
Observaciones	Correcto.

Descripción	Normalización de un vector.
Resultado esperado	Prueba A: $X = 0{,}802 \;\; Y = 0{,}267 \;\; Z = 0{,}534$ Prueba B: $X = 1 \;\; Y = 0 \;\; Z = 0$ Prueba C: $X = 0 \;\; Y = 1 \;\; Z = 0$ Prueba D: $X = 0 \;\; Y = 0 \;\; Z = 1$
Resultado obtenido	Prueba A: $X = 0{,}802 \;\; Y = 0{,}267 \;\; Z = 0{,}534$ Prueba B: $X = 1 \;\; Y = 0 \;\; Z = 0$ Prueba C: $X = 0 \;\; Y = 1 \;\; Z = 0$ Prueba D: $X = 0 \;\; Y = 0 \;\; Z = 1$
Observaciones	Correcto.

Descripción	Producto escalar de dos vectores.
Resultado esperado	122
Resultado obtenido	122
Observaciones	Correcto.

Descripción	Obtención del ángulo interno entre dos vectores para cada una de las pruebas realizadas.
Resultado esperado	Prueba A: 0^o Prueba B: 90^o Prueba C: 180^o
Resultado obtenido	Prueba A: 0^o Prueba B: 90^o Prueba C: 180^o
Observaciones	Correcto.

Descripción	Obtención del modulo de un vector.
Resultado esperado	Prueba A: $\sqrt{3}$ Prueba B: 1 Prueba C: 1 Prueba D: 1
Resultado obtenido	Prueba A: $\sqrt{3}$ Prueba B: 1 Prueba C: 1 Prueba D: 1
Observaciones	Correcto.

7.2. Pruebas en dispositivos

Las pruebas presentadas a continuación no han sido realizadas utilizando frameworks con dichos fines, sino que se han utilizado los siguientes dispositivos físicos:

- Sony Xperia Z3 Compact
- Samsung Galaxy Tab S2 8”
- iPad Air 2
- iPod Touch 6ª generación

Con el fin de comprobar el correcto funcionamiento del sistema y de la interfaz de usuario desarrollada.

7.2.1. Tratamiento de errores del usuario

Descripción	Pruebas de errores por parte del usuario.
Acción realizada	Se inicia la aplicación en un dispositivo que carece de los sensores necesarios o de la versión mínima del sistema operativo.
Resultado esperado	Al iniciar la aplicación, aparece un dialogo informando al usuario de la incompatibilidad del dispositivo, deshabilitando el botón de inicio de sesión en iOS, o cerrando la aplicación en Android.
Resultado obtenido	Al inicio, aparece una alerta informando al usuario, tras lo cual, iOS presenta el botón de inicio de sesión deshabilitado; Android cierra la aplicación directamente.
Observaciones	Correcto, verifica los requisitos no funcionales 1 y 2.

Descripción	Pruebas de errores por parte del usuario.
Acción realizada	Se introduce una URL incorrecta.
Resultado esperado	Se deshabilita el botón de inicio de sesión al detectar una URL incorrecta, informando al usuario del error, habilitandose en cuanto ésta sea correcta.
Resultado obtenido	Al introducir una URL incorrecta se visualiza el error, además de ver el botón de inicio de sesión deshabilitado.
Observaciones	Correcto.

Descripción	Pruebas de errores por parte del usuario.
Acción realizada	Las credenciales introducidas son incorrectas.
Resultado esperado	Una vez se reciba la respuesta del servidor, desaparecerá la visualización del progreso y aparecerán errores en los campos de usuario y contraseña.
Resultado obtenido	Al recibir la respuesta del servidor, desaparece el dialogo de progreso, mostrando error de credenciales en los campos usuario y contraseña.
Observaciones	Correcto.

Descripción	Pruebas de errores por parte del usuario.
Acción realizada	Se deshabilita el bluetooth durante el funcionamiento de la visión de realidad aumentada.
Resultado esperado	Se pide al usuario que habilite el Bluetooth, desactivando conexiones y el dibujado hasta que se produzca dicho suceso.
Resultado obtenido	El usuario es informado de la necesidad de activar el Bluetooth, no se producen nuevas peticiones de red ni dibujado, una vez habilitado el Bluetooth, se vuelven a activar el dibujado y las peticiones de red.
Observaciones	Correcto.

7.2.2. Funcionalidad

Descripción	Pruebas de funcionalidad.
Acción realizada	Se realiza un inicio de sesión correcto.
Resultado esperado	Se inicia la vista de realidad aumentada, con los datos proporcionados por el servidor introducido.
Resultado obtenido	Aparece la vista de realidad aumentada, con los datos proporcionados por el servidor introducido.
Observaciones	Correcto, verifica el requisito funcional 5.

Descripción	Pruebas de funcionalidad.
Acción realizada	No hay conexión durante el inicio de sesión.
Resultado esperado	Durante el inicio de sesión se informa al usuario de un error de red.
Resultado obtenido	El usuario es informado de un error de red durante el inicio de sesión.
Observaciones	Correcto.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se inicia sesión en un dispositivo Android con el Bluetooth apagado.
Resultado esperado	Se desencadena el proceso de inicio de sesión, activando el Bluetooth.
Resultado obtenido	El proceso de inicio de sesión comienza, activando a su vez el Bluetooth del dispositivo.
Observaciones	Correcto.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se inicia sesión con un servidor destino incorrecto, o con problemas en su funcionamiento.
Resultado esperado	Se desencadena el proceso de inicio de sesión, una vez detectado algun problema con el servidor, se informa al usuario, deteniendo el proceso.
Resultado obtenido	El proceso de inicio de sesión comienza, parando el proceso e informado al usuario al detectar un error en el servidor.
Observaciones	Correcto.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se dirige la cámara hacía un código QR cuando no se está en la pestaña correspondiente.
Resultado esperado	El código no es analizado.
Resultado obtenido	El código es analizado.
Observaciones	Incorrecto. El tratamiento de códigos QR no se deshabilita al cambiar de pestaña, se introduce un callback que desactiva dicho tratamiento. CORREGIDO.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se escanea un código QR con datos correctos.
Resultado esperado	Se desencadena el proceso de inicio de sesión.
Resultado obtenido	El proceso de inicio de sesión comienza.
Observaciones	Correcto, verifica el requisito funcional 5.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se escanea un código QR con datos incorrectos.
Resultado esperado	Se informa al usuario que el código QR es incorrecto.
Resultado obtenido	El usuario es informado de que el código escaneado es incorrecto.
Observaciones	Correcto.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se produce algún error en la conexión con el servidor durante el funcionamiento de la visión de realidad aumentada.
Resultado esperado	Aparece información relativa al estado de la conexión, una vez aceptada dicha información, se regresa a la vista de inicio de sesión.
Resultado obtenido	El usuario es informado del error en la conexión, una vez aceptada, aparece la vista de inicio de sesión.
Observaciones	Correcto.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se inicia la visión de realidad aumentada.
Resultado esperado	Aparece la visión de la cámara trasera.
Resultado obtenido	Se visualiza la visión de la cámara trasera.
Observaciones	Correcto, verifica el requisito funcional 2.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se produce un movimiento del dispositivo, cambiando su orientación y localización relativa al espacio.
Resultado esperado	Los puntos de interés a mostrar aparecen en su posición correspondiente en pantalla.
Resultado obtenido	Al girar el dispositivo los puntos de interés van cambiando su posición, con el ángulo correcto y el tipo adecuado, sin embargo, en ciertas ocasiones, estas posiciones no son las correctas, obteniendo un cierto ángulo de desvío lateral.
Observaciones	Verifica los requisitos funcionales 2, 3 y 6. Sin embargo, se ha detectado una inexactitud en los sensores utilizados para obtener la orientación del dispositivo, así como una pequeña variación en el ángulo de la cámara calculado respecto del valor real y una alta sensibilidad a las interferencias magnéticas. Se ha propuesto una solución a cada uno de estos problemas en la Sección 9.2.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se produce un movimiento del dispositivo, cambiando su orientación de pantalla.
Resultado esperado	Los puntos de interés a mostrar aparecen en su posición correspondiente en pantalla.
Resultado obtenido	Al girar el dispositivo los puntos de interés van cambiando su posición, con el ángulo correcto y el tipo adecuado.
Observaciones	Correcto, verifica los requisitos funcionales 2, 3 y 6.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se dispone de conexión Bluetooth y se ha iniciado sesión en un servidor XtremeLoc.
Resultado esperado	Se envían señales Bluetooth de baja energía, con el formato correspondiente al solicitado por el servidor.
Resultado obtenido	Las antenas reciben la señal Bluetooth en formato iBeacon correspondientes al usuario que inició sesión.
Observaciones	Correcto, verifica el requisito funcional 6, y el requisito no funcional 4.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se dispone de conexión Bluetooth y se ha iniciado sesión en un servidor XtremeLoc.
Resultado esperado	Los puntos de interés actualizan su posición con la información administrada por el servidor remoto.
Resultado obtenido	Se actualizan los puntos de interés con la información recibida en la conexión con el servidor remoto.
Observaciones	Correcto, verifica el requisito funcional 5, y el requisito no funcional 3.

Descripción	Pruebas de funcionalidad.
Acción realizada	Se seleccionan los nuevos elementos a mostrar y se acepta la decisión.
Resultado esperado	Solamente se muestran en la vista de realidad aumentada aquellos elementos seleccionados por el usuario para ser mostrados.
Resultado obtenido	La vista de realidad aumentada solamente muestra aquellos elementos que el usuario seleccionó para ser mostrados.
Observaciones	Correcto, verifica el requisito funcional 1.

7.2.3. Interfaz de usuario

Descripción	Pruebas de interfaz de usuario
Acción realizada	Se han iniciado profilers para cada plataforma, utilizando para ello varios dispositivos físicos reales.
Resultado esperado	60 FPS en el dibujado de la pantalla, en cualquiera de las vistas.
Resultado obtenido	Los profilers indican un dibujado de 60 FPS en todas las vistas de la aplicación.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	Navegación por todas las vistas, con el dispositivo configurado en los idiomas Inglés y castellano.
Resultado esperado	Todos los elementos aparecen en su idioma correspondiente.
Resultado obtenido	La interfaz de usuario aparece en el idioma de configuración del dispositivo.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	En la vista de tutorial se pulsa sobre el botón de inicio de sesión.
Resultado esperado	Aparece la vista de inicio de sesión.
Resultado obtenido	Aparece la vista de inicio de sesión.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	En la vista de realidad aumentada se inicia un cierre de sesión.
Resultado esperado	Al carecer de los permisos necesarios, se solicita al usuario dichos permisos, mostrando el motivo de su solicitud.
Resultado obtenido	Al carecer de los permisos necesarios, se solicita al usuario dichos permisos, mostrando el motivo de su solicitud.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	En la vista de inicio de sesión se cambia entre las distintas pestañas.
Resultado esperado	La vista actual cambia entre el inicio de sesión mediante código QR o mediante formulario, dependiendo de la pestaña seleccionada.
Resultado obtenido	Al seleccionar una pestaña se muestra el contenido de los distintos modos de inicio de sesión.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	En la vista de realidad aumentada se inicia un cierre de sesión.
Resultado esperado	La vista de realidad aumentada es destruida, la conexión actual se interrumpe y aparece la vista de inicio de sesión.
Resultado obtenido	Aparece la vista de inicio de sesión, destruyendo la correspondiente a la realidad aumentada, la conexión con el servidor se detiene.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	En la vista de realidad aumentada se pulsa el botón de filtrado de elementos.
Resultado esperado	Aparece la vista de filtrado de elementos.
Resultado obtenido	Aparece la vista de filtrado de elementos.
Observaciones	Correcto.

Descripción	Pruebas de interfaz de usuario
Acción realizada	Se selecciona el máximo de elementos disponibles para ser mostrados, a continuación se reduce el número de elementos seleccionados.
Resultado esperado	El resto de elementos ignoran las pulsaciones por parte del usuario, apareciendo deshabilitados en la versión iOS, una vez el número de elementos seleccionados se encuentra por debajo del límite, todos los elementos vuelven a estar disponibles para ser seleccionados.
Resultado obtenido	El resto de elementos ignoran las pulsaciones, en iOS aparecen deshabilitados aquellos elementos no seleccionados. Al disminuir el número de elementos seleccionados, el resto de elementos vuelven a responder a la interacción por parte del usuario.
Observaciones	Correcto.

Capítulo 8

Manual de usuario

Este es el manual de usuario, donde se indican, los procedimientos a seguir para la instalación y correcto uso de la aplicación.

8.1. Instalación

8.1.1. Android

Como cualquier otra aplicación en este sistema, *Skywalker* se puede instalar desde la *Play Store* de *Google*, siendo este el método recomendado, ya que la tienda de aplicaciones asegura la compatibilidad del dispositivo destino.

Sin embargo, se ofrece la posibilidad de instalación mediante la *APK*. Si se desea utilizar este método, habrá que permitir la instalación de aplicaciones desde orígenes desconocidos, esta opción está situada en las opciones de seguridad del sistema.

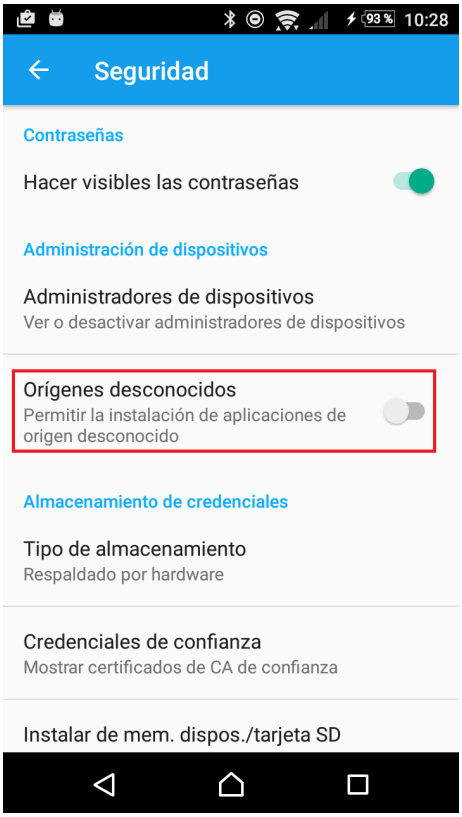


Figura 8.1: Opción de instalación desde orígenes desconocidos.

8.1.2. iOS

La única posibilidad de instalación en sistemas móviles *Apple* pasa por la utilización de la *App Store*, por lo que su instalación es la misma que la de cualquier otra aplicación.

8.2. Desinstalación

En cualquiera de los dos sistemas, la desinstalación se puede realizar desde Ajustes →Aplicaciones, o desde el cajón de aplicaciones, manteniendo pulsada la aplicación propiamente dicha, y arrastrándola a la opción de desinstalación correspondiente a cada capa de personalización.

8.3. Navegación por la interfaz

8.3.1. Inicio de la aplicación

Figura 8.2: Primera vista de la interfaz de usuario Android

Figura 8.3: Primera vista de la interfaz de usuario iOS

En el inicio de la aplicación, se mostrará la figura anterior, en la cual podremos dejar la ejecución automática de la presentación, o podremos navegar entre los distintos paneles mediante un gesto lateral, a izquierda o derecha. La siguiente tabla ilustra el uso de cada elemento para la anterior vista.

Elemento	Función
1. Botón	Navega hacía la vista de inicio de sesión.
2. Indicadores	Indica el número de pagina actual, además de su posición relativa al resto de páginas.

8.3.2. Otorgando permisos

En el caso de que la aplicación no cuente con los permisos necesarios para su correcto funcionamiento aparecerá la siguiente pantalla, la cual no desaparecerá hasta que se hayan otorgado todos los permisos solicitados.

Figura 8.4: Solicitud de permisos Android

Figura 8.5: Solicitud iOS

Elemento	Función
1. Botón	Inicia el proceso de solicitud de los permisos necesarios, una vez concedidos aparecerá la pantalla de inicio de sesión.

8.3.3. Iniciando sesión

Una vez en la pantalla de inicio de sesión, aparecerá la siguiente vista.

Figura 8.6: Vista de inicio de sesión manual Android

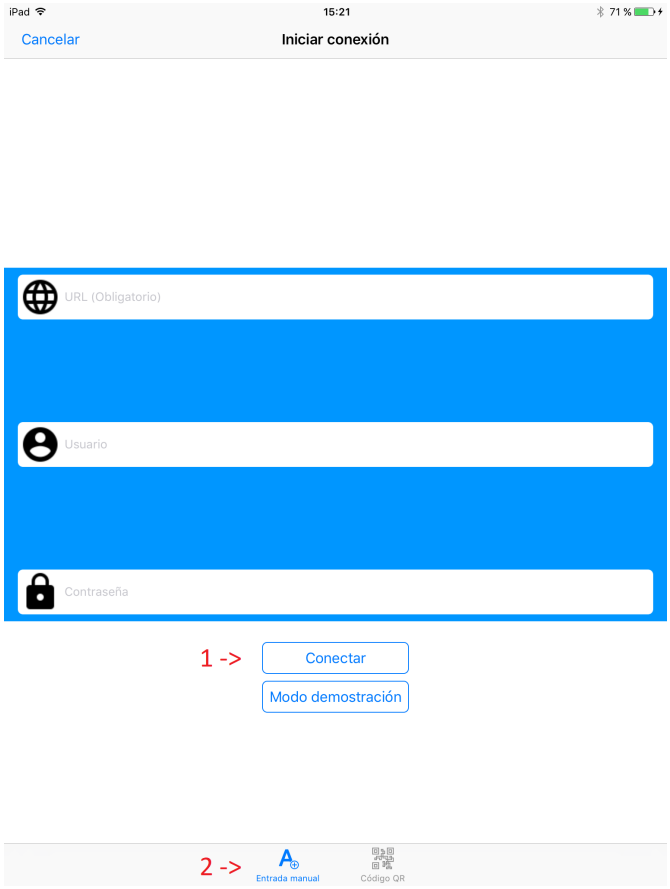


Figura 8.7: Vista de inicio de sesión manual iOS

Una vez introducidos los datos deseados del formulario, pulsando sobre ACEPTAR se iniciará la sesión, con un indicador de progreso. Es importante tener conexión a internet y haber habilitado el Bluetooth en el dispositivo para evitar la aparición de errores a posteriori.

Elemento	Función
1. Botón	Arranca el inicio de sesión.
2. Barra navegación	Indica la pestaña actual, clicando sobre ellas, podremos cambiar entre los distintos modos de inicio de sesión.

Por otra parte, si deseamos iniciar sesión mediante un código QR, aparecerá lo siguiente.

Figura 8.8: Vista de inicio de sesión con código QR Android

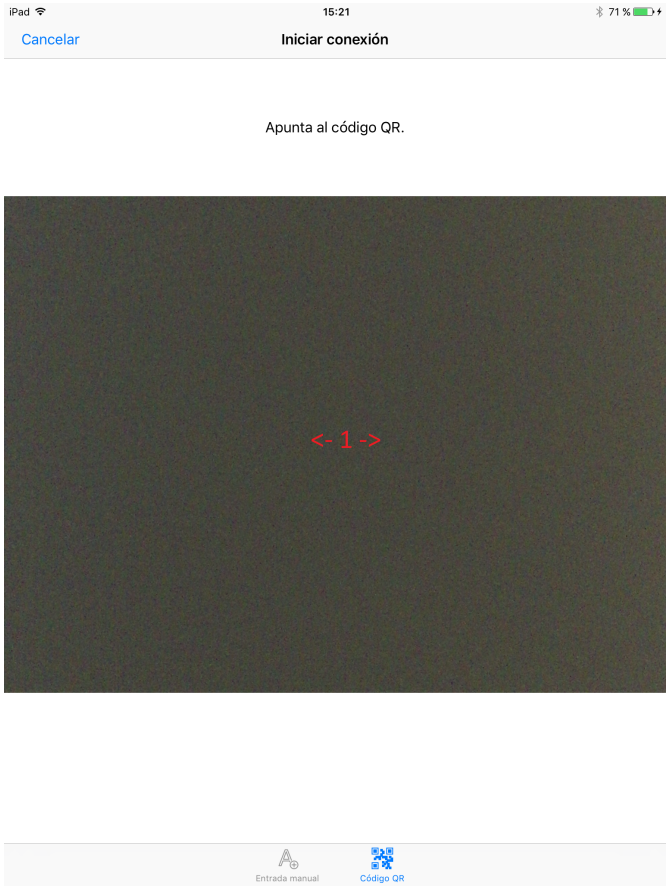


Figura 8.9: Vista de inicio de sesión con código QR iOS

Elemento	Función
1. Vista cámara	Visión de la cámara trasera, centrando en ella un código QR se desencadenará el proceso de inicio de sesión.

8.3.4. Vista de realidad aumentada

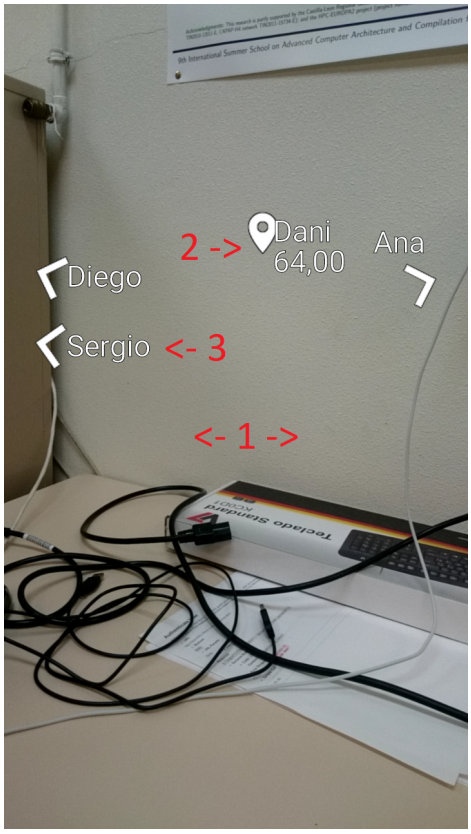


Figura 8.10: Vista de realidad aumentada Android

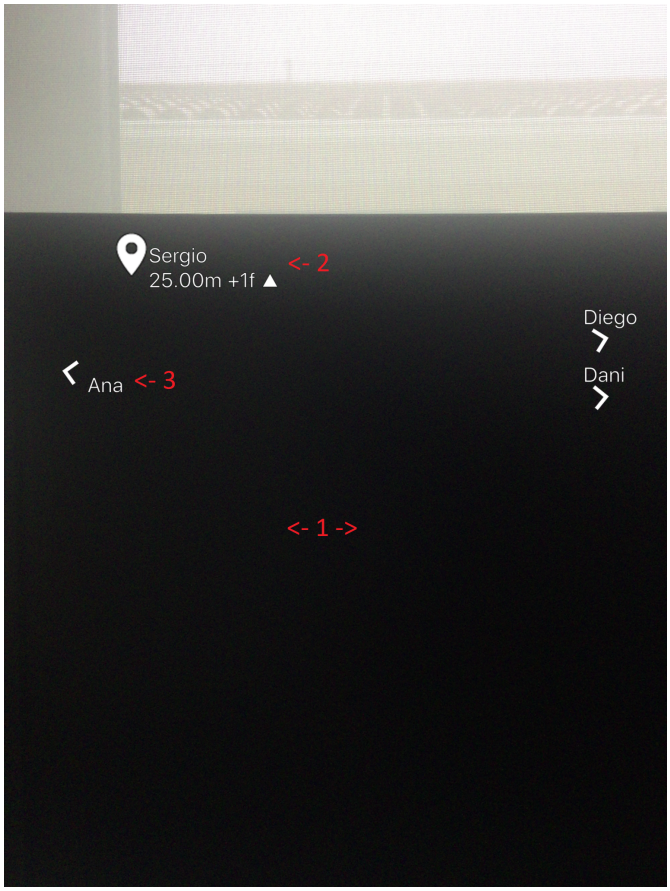


Figura 8.11: Vista de realidad aumentada iOS

Elemento	Función
1. Vista cámara	Visión de la cámara trasera.
2. Elemento	Indicador de elemento que está en el campo de visión actual, indicando distancia hasta el objeto, nombre del objeto, y diferencia de pisos.
3. Elemento	Indicador de elemento fuera del campo de visión actual, indicando hacía donde dirigir el dispositivo para que dicho objeto entre en el campo de visión, además de indicar de que objeto se trata.

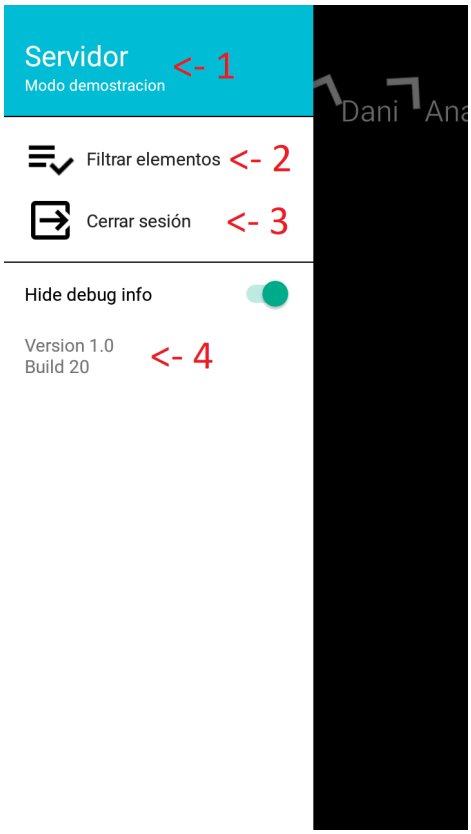


Figura 8.12: Controles para la realidad aumentada Android

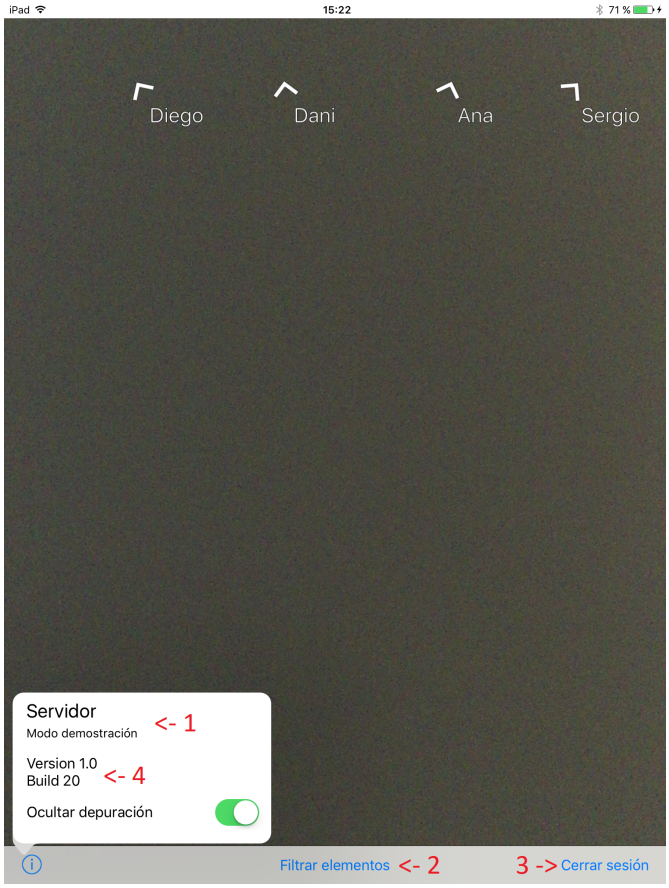


Figura 8.13: Controles para la realidad aumentada iOS

Elemento	Función
1. Indicador	Muestra el servidor actual.
2. Botón	Inicia la selección de elementos, apareciendo la vista correspondiente.
3. Botón	Cierra la sesión, regresando a la vista de inicio de sesión.
4. Indicador	Muestra la versión actual de Skywalker.

8.3.5. Selección de elementos a mostrar

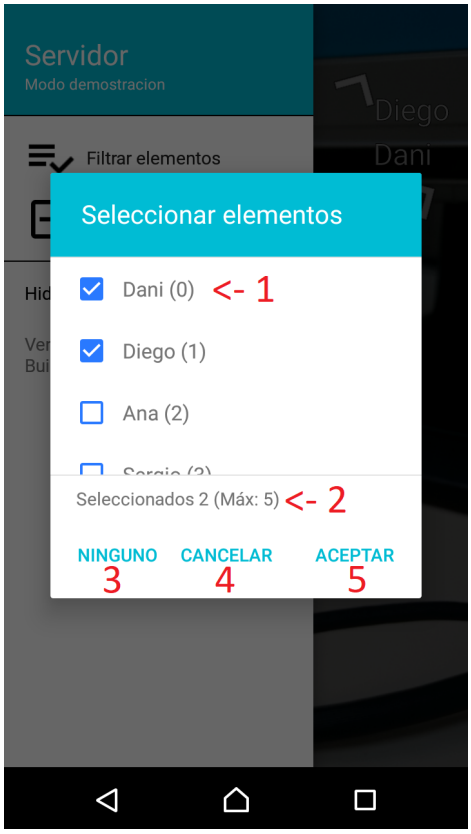


Figura 8.14: Vista de filtrado Android

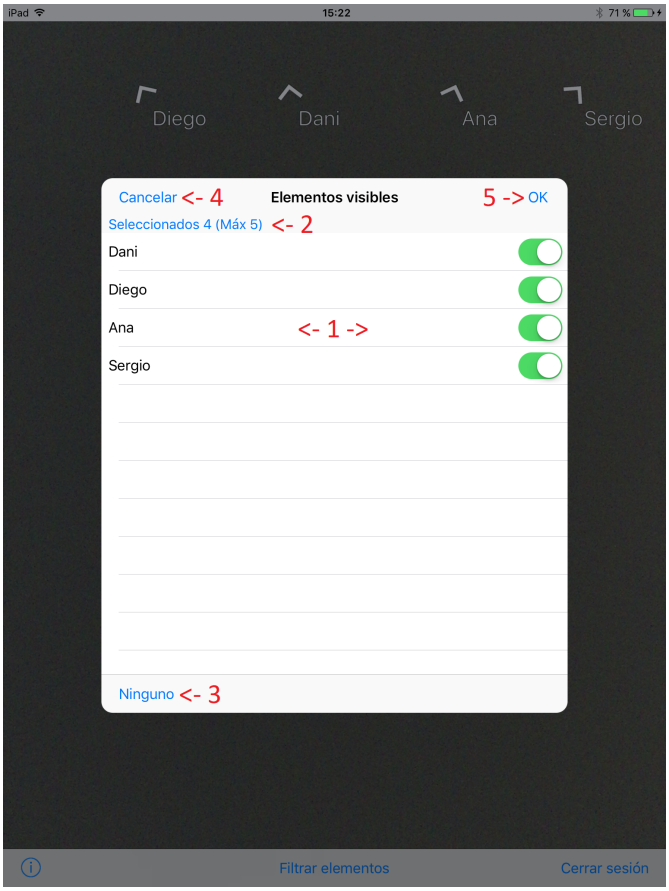


Figura 8.15: Vista de filtrado iOS

Elemento	Función
1. Elementos	Elementos que pueden ser visibles, con la caja seleccionada será un elemento visible, si no lo está sera invisible.
2. Indicador	Estado actual de la selección, indicando el máximo de elementos posibles, y el número actual de seleccionados.
3. Botón	Elimina la selección sobre cualquier elemento ya seleccionado.
4. Botón	Cancela la selección, regresando a la de realidad aumentada, donde se mostrarán aquellos elementos que ya estaban siendo mostrados.
5. Botón	Confirma la selección, regresando a la de realidad aumentada, donde aparecerán aquellos elementos seleccionados.

8.4. Tratamiento de errores

Es posible que durante el funcionamiento de la aplicación aparezcan ventanas informando de algún error, tal y como como aparecen en las figuras 8.16 y 8.17, si esto sucede, simplemente seguir las instrucciones indicadas por dicha ventana, o si no hubiera instrucciones indicadas, contactar con un administrador indicándole el error aparecido.

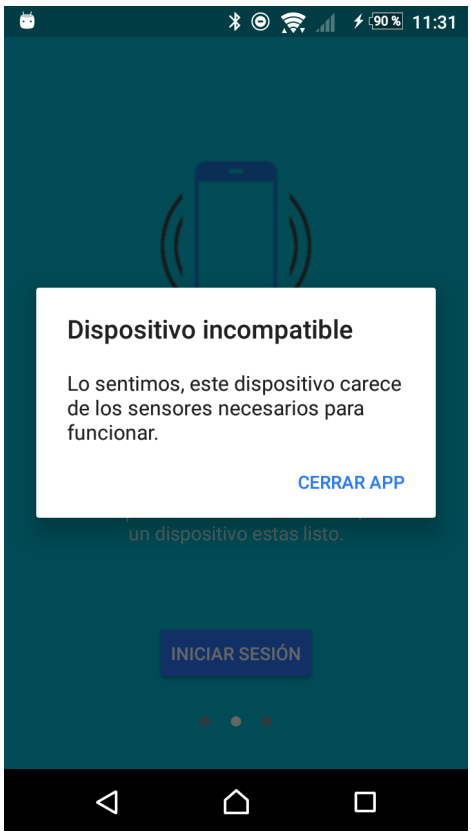


Figura 8.16: Aparición de error Android

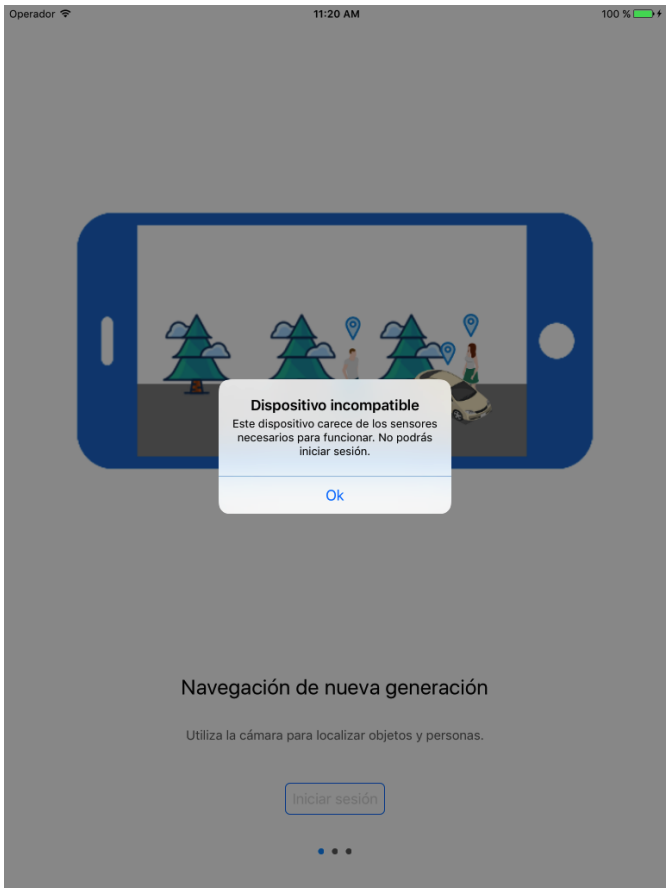


Figura 8.17: Aparición de error iOS

Capítulo 9

Conclusiones y Trabajo Futuro

9.1. Objetivos logrados

Con este desarrollo, se han conseguido los siguientes objetivos:

1. Se han desarrollado dos versiones del sistema de localización de personas a través de realidad aumentada, una para cada plataforma, adaptadas a las posibilidades y peculiaridades de cada una de ellas. Para ello se han utilizado los entornos de desarrollo, lenguajes y Frameworks pertenecientes a cada una.

2. Se ha adaptado el diseño e implementación para la utilización de bibliotecas de soporte en Android, tanto de Google como de terceros, facilitando su mantenimiento y acelerando el desarrollo.

3. Se han utilizado patrones específicos para abstraer y aislar posibles cambios futuros en las APIs.

4. Se ha implementado una solución para la obtención de la orientación espacial del dispositivo en ambas versiones, la cual requiere menos de la mitad del tiempo de ejecución que la versión básica de referencia.

5. Se han minimizado los posibles errores del usuario, teniendo en cuenta cada posible escenario de fallo por parte del potencial usuario del sistema, utilizando para ello una interfaz de usuario agradable, nativa a cada plataforma y sencilla.

6. Se ha conseguido una interfaz de usuario con la velocidad de dibujado deseada, 60 FPS, elaborando para ello un diseño con gran énfasis en la utilización correcta de la concurrencia, evitando así ralentizaciones en la interfaz de usuario y aprovechando el hardware cada vez más potente del que los dispositivos móviles disponen.

9.2. Trabajo futuro

El trabajo aquí presentado puede tener bastante continuidad por parte de trabajos futuros:

1. Utilización de un método de desarrollo común, para facilitar el mantenimiento del código, por ejemplo usando Xamarin.

2. Ampliación a otros sistemas, tales como Google Glass, Microsoft Hololens, o demás dispositivos que vayan surgiendo.

3. Mantenimiento del sistema para seguir funcionando en las nuevas versiones de los sistemas operativos ya soportados, ya que, durante el desarrollo de *Skywalker*, se lanzó la versión preliminar de Android O [13], que incluye numerosos cambios y restricciones, y aunque esta versión no ha afectado en ningún aspecto al trabajo aquí presentado, si que podrían necesitarse cambios sustanciales en futuras versiones del sistema operativo, además, dentro de muy poco tiempo se celebrará la WWDC 2017 [9] de *Apple*, donde se presentará una nueva versión de *iOS*, la cual incluirá, seguramente, bastantes cambios a nivel de API.

4. Ampliación del sistema para la inclusión de distintas localizaciones indoor, e incluso outdoor, para así, utilizando el mismo mecanismo de visualización, saber que algún punto de interés se encuentra en otro edificio o localización, y mostrar su posición tal y como ya se ha implementado.

5. Desarrollar un mecanismo para calibrar la orientación espacial del dispositivo, cambiando el tipo de sensor a uno que únicamente utilice el giroscopio, de esta forma se evitarían las interferencias magnéticas, las descalibraciones del sensor y el margen de error del sensor ya implementado; para solucionar este problema, también podría implementarse otro mecanismo que utilice todos los sensores disponibles, combinando sus resultados, del mismo modo que ya hacen los sensores de tipo fusión de los frameworks utilizados, pero personalizado para *Skywalker*.

6. Implementar un mecanismo para detectar el ángulo de visión correcto para cada dispositivo, almacenando esta información en el sistema XtremeLoc para futuras consultas.

A medida que la tecnología avance y aparezcan nuevos mecanismos y sistemas, se podrían implementar nuevas funcionalidades en nuevos sistemas, con lo que esta lista podría crecer, con puntos imprevisibles con la tecnología disponible hoy en día.

Bibliografía

[1] Robbie Abed. Hybrid vs native mobile apps – the answer is clear. https://ymedialabs.com/hybrid-vs-native-mobile-apps-the-answer-is-clear/, sep 2016.

[2] Apple. Core motion documentation. https://developer.apple.com/reference/coremotion.

[3] Apple. Guía de programación con concurrencia. https://developer.apple.com/library/content/documentation/General/Conceptual/ConcurrencyProgrammingGuide/Introduction/Introduction.html.

[4] Apple. ios motion basics. https://developer.apple.com/library/content/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html.

[5] Apple. ios tabs. https://developer.apple.com/ios/human-interface-guidelines/ui-bars/tab-bars/.

[6] Apple. ios ui thread. https://developer.apple.com/reference/uikit.

[7] Apple. ios view containers. https://developer.apple.com/library/content/featuredarticles/ViewControllerPGforiPhoneOS/ImplementingaContainerViewController.html.

[8] Apple. Low pass filter. https://en.wikipedia.org/wiki/Low-pass_filter.

[9] Apple. Wwdc 2017. https://developer.apple.com/wwdc/.

[10] Janna Badalian. Html5 vs native: The debate is over. https://www.mobilesmith.com/html5-vs-native-debate-is-over/, may 2015.

[11] Google. Android dialogs. https://material.io/guidelines/components/dialogs.html#dialogs-alerts.

[12] Google. Android fragments. https://developer.android.com/guide/components/fragments.html.

[13] Google. Android oreo preview. https://developer.android.com/preview/index.html.

[14] Google. Android sensor events. https://developer.android.com/reference/android/hardware/SensorEvent.html.

[15] Google. Android tabs. https://material.io/guidelines/components/tabs.html#.

[16] Google. Android: Texture view. https://developer.android.com/reference/android/view/TextureView.html.

[17] Google. Android ui thread. https://developer.android.com/guide/components/processes-and-threads.html.

[18] Google. Broadcast listeners. https://developer.android.com/guide/components/broadcasts.html.

[19] Google. Cameracapturesession documentation. https://developer.android.com/reference/android/hardware/camera2/CameraCaptureSession.html.

[20] Google. Eventos de entrada en android. https://developer.android.com/guide/topics/ui/ui-events.html.

[21] Google. Firebase. https://firebase.google.com/?hl=es-419.

[22] Google. Google vision api. https://developers.google.com/vision/.

[23] Google. Handler documentation. https://developer.android.com/reference/android/os/Handler.html.

[24] Google. Volley framework. https://developer.android.com/training/volley/index.html.

[25] KeepSafe. Tap target view library from keepsafe. https://github.com/KeepSafe/TapTargetView.

[26] R Mukundan. Quaternions: From classical mechanics to computer graphics, and beyond. In *Proceedings of the 7th Asian Technology conference in Mathematics*, pages 97–105, 2002.

[27] Netmarketshare. Mobile market share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1, 2017.

[28] Radius networks. Volley framework. http://altbeacon.github.io/android-beacon-library/.

[29] Phandroid. Google glass: Navigation. https://www.youtube.com/watch?v=IZdkIVS53Uw, may 2013.

[30] Phonegap. phonegap. http://phonegap.com.

[31] Guillermo Rodriguez. Diferencias entre realidad aumentada y realidad virtual. http://www.vix.com/es/btg/tech/13396/diferencias-entre-realidad-aumentada-y-realidad-virtual.

[32] Sencha. sencha. https://www.sencha.com.

[33] Robert Sheldon. Understanding html5 mobile application development. http://searchmobilecomputing.techtarget.com/feature/Understanding-HTML5-mobile-application-development, jan 2016.

[34] Sara Sundqvist. The advantages of native apps. http://twotoasters.com/ideas/2013/why-native-apps-are-better-than-mobile-web/, jun 2013.

[35] Accent systems. ibeacon frame. https://accent-systems.com/support/developers/ibks-frames-protocol/.

[36] Twitter. Fabric. https://fabric.io/.

[37] Wikipedia. Camera angle of view. https://en.wikipedia.org/wiki/Angle_of_view.

[38] Wikipedia. Matrices de rotación. https://en.wikipedia.org/wiki/Rotation_matrix.

[39] Wikipedia. Cartografía. https://es.wikipedia.org/wiki/Cartograf%C3%ADa, may 2002.

[40] Wikipedia. Realidad mixta. https://es.wikipedia.org/wiki/Realidad_mixtal, dec 2008.

[41] Xamarin. xamarin. https://www.xamarin.com.

Apéndice A

XtremeLoc API REST

A.1. Authenticate User

Authenticates a user and returns token which grants access to the API for "a while" (to be decided). Token authentication is done reading the Authorization header in each HTTP request (e.g. Authorization: Bearer <token>).

URL	http://xtremeloc.rdnest.com/api/authentication	
Method	POST	
URL Params	Required	None
	Optional	None
Data Params	Required	{ "login": "<user_login>", "password": "<user_password>"}
	Optional	None
Success Response	200	<authentication_token>

A.2. Get the list of authorized centers

Returns json with the lists of centers that are visible for the authenticated user.

URL	http://xtremeloc.rdnest.com/api/centers	
Method	GET	
URL Params	Required	None
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	[{<center_#0>}, {<center_#1>}, ..., {<center_#n>}]

A.3. Get information about a specific center

Returns json of a specified center.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id	
Method	GET	
URL Params	Required	center_id = [integer]
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	{“id”:<center_id>, “address” : “<address>”, “city”:“<city>”, “province” : “<province>”, “country” : “<country>”, “lat” : <latitude>, “lon” : <logitude>, “name” : “<center_name>”, ”northX” : <x_value_of_reference_north_vector>, ”northY” : <y_value_of_reference_north_vector>, ”pcode” : ” <postal_code> ”}

A.4. Get the list of RD Hubs in a specific center

Returns json with the list of RD Hubs of a specified center.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/rdhubs	
Method	GET	
URL Params	Required	center_id = [integer]
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	[{rd_hub_#0}, {rd_hub_#1}, ..., {rd_hub_#n}]

A.5. Get the information of a specific RD Hub

Returns json with information of an specific RD Hub in a specified center.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/rdhubs/:rdhub_id	
Method	GET	
URL Params	Required	center_id = [integer], rdhub_id = [integer]
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	{ ”center” : <center_id>, ”id” : <rd_hub_id>, ”x” : <x_position_between_0_and_1>, ”y” : <y_position_between_0_and_1>,”z” : <floor_number> }

A.6. Get the list of Beacons in a specific center

Returns json with the list of Beacons of a specified center.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/beacons	
Method	GET	
URL Params	Required	center_id = [integer]
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	[{beacon_#0}, {beacon_#1}, ..., {beacon_#n}]

A.7. Get the information of a specific Beacon

Returns json with information of an specific Beacon in a specified center. If a Unix Timestamp is supplied, then the information of the beacon at that precise time is returned, otherwise this method will return the last information available. The Unix Timestamp param is milliseconds since epoch (UTC).

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/ beacons/:beacon_id?time=:unix_timestamp	
Method	GET	
URL Params	Required	center_id = [integer], beacon_id = [integer]
	Optional	unix_timestamp = [integer]
Data Params	Required	None
	Optional	None
Success Response	200	{”center” : <center_id>, ”id” : <beacon_id>, ”name” : ” <name_of_the_owner> ” }

A.8. Get the list of signals of a specific Beacon

Returns json a list of signals of a specific Beacon. This list by default includes the last “few” (to be decided) signals for each RD Hub in range. Optional params may be used to select a specific time lapse for past signals.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/ beacons/:beacon_id/signals?since=:since&until=:until	
Method	GET	
URL Params	Required	center_id = [integer], beacon_id = [integer]
	Optional	since = [long], until = [long]
Data Params	Required	None
	Optional	None
Success Response	200	[{signal_#1}, {signal_#2}, ..., {signal_#n}]

A.9. Get the information of a specific signal of a specific Beacon-RD Hub pair

Returns json of a specific signal of a specific Beacon-RD Hub pair.

URL	http://xtremeloc.rdnest.com/api/centers/:center_id/ beacons/:beacon_id/signals/:rdhub_id/:time	
Method	GET	
URL Params	Required	center_id = [integer], beacon_id = [integer], rdhub_id = [integer], time = [long]
	Optional	None
Data Params	Required	None
	Optional	None
Success Response	200	{"center" : <center_id>, "beacon" : <beacon_id>, "rdhub" : <rdhub_id>, "time" : <timestamp>, "strength" : <signal_strength>}