2017

FIUBAAR

Realidad Aumentada en ANDROID

Documentación del proyecto de Trabajo Profesional de Ingeniería en Informática FIUBAAR.

Esteban Guillardoy (P.81597) eguillardoy@gmail.com
& Alejandro E. Aguilar (P.73935) alezek@gmail.com
Facultad de Ingeniería de la Universidad de Buenos Aires
04/05/2017

TABLA DE CONTENIDO

Introducción ... 6
Objetivos ... 6
Motivación ... 6
Metodología ... 7
Especificación de Requerimientos ... 7
Administración del Proyecto ... 8
Calendario ... 8
Replanificación N°1 - Al inicio del Sprint N°3, 20/Octubre/2014 ... 8
Replanificación N°2 - Al inicio del Sprint N°4, 26/Enero/2015. ... 8
Replanificación N°3 - Al inicio del Sprint N°5, 26/Septiembre/2016. ... 8
Sprint N°1 (137HH)- Inicio: 04/08/2014 Finalización: 07/09/2014 ... 8
Sprint N°2 (144HH) - Inicio: 08/09/2014 Finalización: 19/10/2014 ... 9
Sprint N°3 (156HH) - Inicio: 15/12/2014 Finalización: 25/01/2015 ... 9
Sprint N°4 (152HH) - Inicio: 03/08/2015 Finalización: 13/09/2015 ... 10
Sprint N°5 (163HH) - Inicio: 26/09/2016 Finalización: 06/11/2016 ... 10
Sprint N°6 (151HH) - Inicio: 07/11/2016 Finalización: 18/12/2016 ... 10
Sprint N°7 (154HH) - Inicio: 02/01/2017 Finalización: 26/03/2017 ... 11
Burndown Charts por Sprint ... 12
Arquitectura de la Aplicación ... 16
Arquitectura de Android ... 16
El Patrón Model-View-Presenter ... 17
Performance ... 17
Evolución de la Arquitectura ... 19
Arquitectura 1 - Inicial ... 19
Arquitectura 2 ... 20
Arquitectura 3 ... 21
Arquitectura 4 - Final ... 23
Diagrama de paquetes ... 25
Diagramas de Clases ... 35
Clase mainApplication ... 35
Clase mainActivity ... 36
Clase mainController ... 37
Clase objectAR ... 38
Clase marker ... 39
Clase markerDetector ... 40
Clase nativeHandTrackingDetector ... 41
Clase handDebugView ... 42

Clase vuforiaFragment 43
Clase vuforiaRenderer 44
Clases del paquete Engine3D: Base3DFragment y modelFragment 45
Clases del paquete rest: RestClient y download3DModelAsyncTask 46
Diagramas de Secuencia 47
Secuencia: Escaneo de un marcador 47
Secuencia: Detección de Mano 48
Diagrama de Despliegue 49
Diseño de la Interfaz de Usuario 50
Modelo de datos 51
Pruebas durante el proyecto 54
Sprint N°1 54
US1.1 Instalación de FIUBAAR Cliente 54
US1.2 Ejecución de FIUBAAR Cliente 55
US1.3 FIUBAAR Cliente básica hace uso de stack de Fragments 55
US1.4 OpenCV Integrado 56
US1.5 Engine3D integrado 56
Sprint N°2 57
US2.1 Reconocimiento perpendicular de marcador 57
US2.2 Reconocimiento perpendicular de marcador rotado 57
US2.3 Tracking de marcador 58
US2.4 Reconocimiento de marcador multiangular 58
Sprint N°3 59
US3.1 FIUBAAR Servidor Básico con Registro de Usuarios 59
US3.2 FIUBAAR Servidor Básico con Respuesta 60
US3.3 FIUBAAR Cliente lee QR 60
US3.4 FIUBAAR Cliente edita y guarda configuración 61
US3.5 FIUBAAR Cliente se comunica con FIUBAAR Servidor 61
Sprint N°4 62
US4.1 FIUBAAR Servidor permite crear proyectos 62
US4.2 FIUBAAR Servidor permite eliminar proyectos 62
US4.3 FIUBAAR Cliente ubica un modelo 3D sobre el marcador 63
US4.4 FIUBAAR Cliente lee QR y recibe modelo del proyecto 63
Sprint N°5 64
US5.1 FIUBAAR Cliente reconoce una mano en la escena 64
US5.2 FIUBAAR Cliente puede seguir a la mano en la escena 64
US5.3 FIUBAAR Cliente cuenta los dedos desplegados 65
Sprint N°6 66
US6.1 FIUBAAR Cliente agranda modelo 3d con contacto 66

US6.2 FIUBAAR Cliente gira modelo 3d con contacto 66
US6.3 FIUBAAR Cliente interactúa con el modelo 3D sin contacto 67
Sprint N°7 68
US7.1 FIUBAAR Cliente evalúa estabilidad en reconocimiento 68
US7.2 FIUBAAR Cliente genera eventos "touch" 68
US7.3 FIUBAAR Cliente reconoce gesto de zoom 69
US7.4 FIUBAAR Cliente reconoce gesto de rotado 70
Investigaciones durante el desarrollo 71
Investigación de Markers 71
Estado del arte 71
Requerimientos FIUBAAR 71
Elección del marker núcleo 72
Tracking del marker 72
Marker seleccionado 73
Documentación adicional útil 73
Investigación Detección de Markers 75
Implementaciones analizadas 75
Documentación útil 76
Testing de implementaciones 76
Implementación seleccionada 80
Investigación Detección de código QR 82
Implementaciones encontradas 82
Documentación útil 83
Testing de implementaciones 83
Implementación seleccionada 84
Investigación Engine 3D Android 84
Implementaciones encontradas 84
Documentación útil 85
Implementación seleccionada 85
Investigación Clientes REST Android 85
Implementaciones encontradas 85
Documentación útil 86
Testing de implementaciones 87
Implementación seleccionada 87
Investigación Hand & Finger Tracking 88
Implementaciones encontradas 88
Testing de implementaciones 89
Implementación seleccionada 89
Investigación sobre detección de gestos 90

Ideas de Implementación 93
Documentación útil 95
Conclusiones del Proyecto 97
Apéndice I - Armado del Entorno de Desarrollo 98
Herramientas requeridas 98
Configuración 98
Definición de variables de entorno 99
Dispositivos Android 100
Herramientas adicionales 100
Notas Adicionales 101
Buenas Prácticas 102
Comentarios 102
Desarrollo en Android 102
Repositorio público 103
Repositorio privado 103

INTRODUCCIÓN

OBJETIVOS

El objetivo principal del trabajo descrito en el presente documento, fue el de desarrollar un sistema de realidad aumentada para dispositivos móviles. Este sistema es el resultado de dos aplicaciones individuales que funcionan de manera integrada. Una de ellas es referenciada en el presente documento indistintamente como “Aplicación cliente” o “FIUBAAR Cliente” y es la que se ejecuta en el dispositivo móvil con sistema operativo Android. La segunda, se denomina “Aplicación servidor” o “FIUBAAR Servidor” y su ejecución tiene lugar en una PC con sistema operativo Windows (7 o superior) o Linux.

La aplicación cliente captura video y despliega superpuesto a éste, una animación u objeto 3D en una región determinada por un marcador (*marker*). La animación a mostrar es seleccionada a partir de un código embebido en el marker y se descarga -a demanda- del servidor en el momento en que dicho código es reconocido. Así mismo, la aplicación permite interactuar con la animación a través de gestos realizados con las manos. Éstos son interpretados mediante el análisis de las imágenes capturadas por el dispositivo.

Generalizando, han sido también objetivos del proyecto el que sus autores profundicen sus conocimientos sobre el diseño y desarrollo de soluciones para dispositivos móviles dentro de un entorno Android, con problemas complejos que involucran desde la utilización de implementaciones de algoritmos de visión por computadora hasta el manejo de motores gráficos.

El proyecto cuenta con su página oficial en https://fiubaar.github.io/ , siendo ésta la forma recomendada de acceder a las distintas aplicaciones y a los repositorios públicos del proyecto.

MOTIVACIÓN

A partir de los intereses personales de los miembros del equipo en temas tales como los sistemas gráficos y las tecnologías de procesamiento de imágenes, encontramos en la realidad aumentada el ámbito en el cual dichas disciplinas se integran de forma de armoniosa con gran potencial. Sumado a lo anterior, el gran progreso en el desarrollo de las plataformas móviles nos acerca rápidamente a un amplio público.

Así mismo, encontramos la posibilidad de incorporar elementos de diferenciación respecto a otras implementaciones halladas al analizar el estado del arte, tales como lograr cargas dinámicas de objetos 3D que no se encuentren vinculadas estáticamente al dispositivo, así como también lograr la interacción mediante gestos, acercando la realidad aumentada un paso en la dirección de la realidad virtual.

Lo expuesto nos llevó a considerar que un proyecto con estas características reúne condiciones que motivan el desarrollo de un Trabajo Profesional.

METODOLOGÍA

Para el presente proyecto se utilizó la metodología de desarrollo ágil SCRUM, planteando iteraciones ("sprints") de una duración inicial de cuatro semanas. Cada una de las iteraciones contó con las siguientes etapas: Investigación, Diseño, Desarrollo, Testing y Documentación.

ESPECIFICACIÓN DE REQUERIMIENTOS

El proyecto buscó cumplir con un conjunto de requerimientos funcionales, de los cuales se destacan los siguientes:

- Una aplicación servidor capaz de proveer modelos 3D asociados a los proyectos o campañas vinculados a cada marcador.
- La aplicación cliente debe reconocer un marcador y realizar seguimiento del mismo a partir del video capturado por el dispositivo.
- La aplicación cliente debe ser capaz de capaz de conectarse al servidor asociado y descargar un archivo con el objeto 3D a representar.
- La aplicación cliente debe ser capaz de reconocer una mano e identificar sus dedos, así como también realizar un seguimiento de ellos.
- La aplicación debe ser capaz de realizar gestos realizados con la mano tales como pinchado y giro.

Para mayor nivel de detalle sobre cada uno de estos requerimientos se sugiere consultar la propuesta de trabajo profesional presentada.

ADMINISTRACIÓN DEL PROYECTO

CALENDARIO

REPLANIFICACIÓN N°1 - AL INICIO DEL SPRINT N°3, 20/OCTUBRE/2014.

Se re planifica de acuerdo con la performance mostrada por el equipo hasta este punto del proyecto. La replanificación da lugar a los siguientes cambios:

- Los sprints cambian su duración de las cuatro semanas estimadas inicialmente a seis semanas. Esto se debe a que se verifica la necesidad de incrementar las horas de desarrollo y testing, y a que el tiempo semanal disponible para aplicar al proyecto resulta menor al estimado inicialmente.
- Se incrementa la cantidad de horas de desarrollo en un 25% promedio.
- Se incrementa la cantidad de horas utilizadas en testing en un 100% promedio.
- Se detalla que parte del desarrollo en el sprint 3 contiene perfeccionamiento de la lectura del marker.
- Como consecuencia de lo anterior, la HH totales pasan de 866 a 1057.
- Debido a complicaciones de los integrantes del equipo, el inicio del sprint 3 se posterga. Se da inicio al mismo en fecha 15/Diciembre/2014.

REPLANIFICACIÓN N°2 - AL INICIO DEL SPRINT N°4, 26/ENERO/2015.

Nuevamente, por complicaciones personales de los integrantes del equipo se posterga el inicio del sprint 4 -cuyo inicio debería tener lugar el 26/Enero/2015- hasta el 3/Agosto/2015. No se toma ninguna acción adicional.

REPLANIFICACIÓN N°3 - AL INICIO DEL SPRINT N°5, 26/SEPTIEMBRE/2016.

Nuevamente, el trabajo sufre una postergación, lo que da lugar a esta nueva replanificación, la cual establece el inicio del sprint N°5 en fecha 26/Septiembre/2016. Se contemplan dos semanas de receso entre el 19/Diciembre y 1/Enero. No se toma ninguna acción adicional.

SPRINT N°1 (137HH)- INICIO: 04/08/2014 FINALIZACIÓN: 07/09/2014

- Documentación: Documento de set up del ambiente de desarrollo. [8 HH][AA]

- Investigación: Integración de OpenCV en Aplicaciones Android. [32 HH][EG]
- Investigación: Diseño y definición del marker con justificación de su elección. [18HH][AA]
- Diseño: Diseño de aplicación básica. [18 HH][EG]
- Documentación: Documento de diseño aplicación básica con UML. [6 HH][AA]
- Desarrollo: Armado del ambiente de desarrollo. Aplicación cliente (v0.1) básica funcional y estable para el uso de la cámara con integración con OpenCV. [50 HH][AA+EG]
- Testing: Aplicación cliente en su versión actual. [4 HH][AA]

Estado del Sprint N°1: Cumplido [T. Estimado: 4 semanas, T. Real: 5 semanas]

SPRINT N°2 (144HH) - INICIO: 08/09/2014 FINALIZACIÓN: 19/10/2014

- Documentación: actualización de documentos de diseño. [18HH][EG]
- Desarrollo: Aplicación cliente (v0.2) con reconocimiento y tracking del marker, superando la dificultad de mantener la efectividad de dichas acciones aun cuando se produzcan rotaciones y/o ligeros cambios en ángulo de la cámara respecto al marker. [100HH][AA+EG]
- Investigación: Diseño y definición de un código de identificación adecuado, con detección y corrección de errores. [18HH][AA]
- Testing: Aplicación cliente en su versión actual. [8HH][AA]

Estado del Sprint N°2: Cumplido [T. Estimado: 4 semanas, T. Real: 6 semanas]

SPRINT N°3 (156HH) - INICIO: 15/12/2014 FINALIZACIÓN: 25/01/2015

- Documentación: actualización de documentos de diseño. [4HH][EG]
- Desarrollo: Aplicación servidor (v0.1). [45HH][EG]
- Desarrollo: Aplicación cliente (v0.3) con lectura del código de identificación de cliente y comunicación básica con aplicación servidor. Refinamiento de la lectura del marker (Código QR o equivalente) [75HH][AA]
- Investigación: OpenGL y motores 3D. [20HH][EG]
- Documentación: actualización de documentos de set-up de ambiente (por el motor 3D). [4HH][EG]
- Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°3: Cumplido [T. Estimado: 6 semanas, T. Real: 6 semanas]

SPRINT N°4 (152HH) - INICIO: 03/08/2015 FINALIZACIÓN: 13/09/2015

- Documentación: Actualización de documentos de diseño. [4HH][AA]
- Investigación: Algoritmos e implementaciones para seguimiento de manos y dedos (*hand & finger tracking*) usando OpenCV. [40HH][AA]
- Desarrollo: Aplicación servidor (v0.2) Comunicación completa. El cliente lee el código y lo transmite al servidor, éste identifica al cliente y selecciona la información a representar en la aplicación cliente, y por último envía datos de overlay y animación a la aplicación cliente. [50HH][EG]
- Desarrollo: Aplicación cliente (v0.4) Comunicación completa con servidor y animación básica con motor 3D elegido. Nota: la animación está fuera del alcance de la aplicación, solo se pretende mostrar que la selecciona y descarga correctamente del servidor, se la ubica adecuadamente en la escena y que la interacción con el usuario -en futuros sprints- funciona adecuadamente. [50HH][AA+EG]
- Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°4: Cumplido [T. Estimado: 6 semanas, T. Real: 6 semanas]

SPRINT N°5 (163HH) - INICIO: 26/09/2016 FINALIZACIÓN: 06/11/2016

- Desarrollo: Aplicación servidor v1.0. Versión final, solo un refinamiento de la versión del sprint anterior con bugs corregidos. [24HH][AA]
- Desarrollo: Aplicación cliente (v.0.5). Versión con seguimiento de manos y dedos inicial. [125HH][AA+EG]
- Documentación: Actualización de documentos. [6HH][EG]
- Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°5: Cumplido [T. Estimado: 6 semanas, T. Real: 6 semanas]

SPRINT N°6 (151HH) - INICIO: 07/11/2016 FINALIZACIÓN: 18/12/2016

- Desarrollo: Aplicación cliente (v.0.6). Versión con seguimiento de manos y dedos, aún básico pero integrado ya en las funciones de interacción con la animación. [125HH][AA+EG]
- Documentación: Actualización de documentos. [4HH][AA]
- Documentación: Creación del manual de usuario. [14HH][EG]
- Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA]

Estado del Sprint N°6: Cumplido [T. Estimado: 6 semanas, T. Real: 6 semanas]

SPRINT N°7 (154HH) - INICIO: 02/01/2017 FINALIZACIÓN: 26/03/2017

- Desarrollo: Aplicación cliente (v.1.0). Versión con seguimiento de manos y dedos, totalmente integrado. [100HH][AA+EG]
- Testing: Aplicaciones cliente y servidor en su versión actual (final). [36HH][AA+EG]
- Documentación: Se crean videos de ejemplo de uso. [12HH][AA]
- Demo: Se hace disponible un demo en Google Play. [6HH][EG]

Estado del Sprint N°7: Cumplido [T. Estimado: 6 semanas, T. Real: 12 semanas]

Cantidad Total de Horas Hombre: 1057

A continuación, se detallarán los cuadros de tareas realizadas horas hombre por semana en cada sprint, relacionando la idea inicial de proyecto y lo efectivamente conseguido.

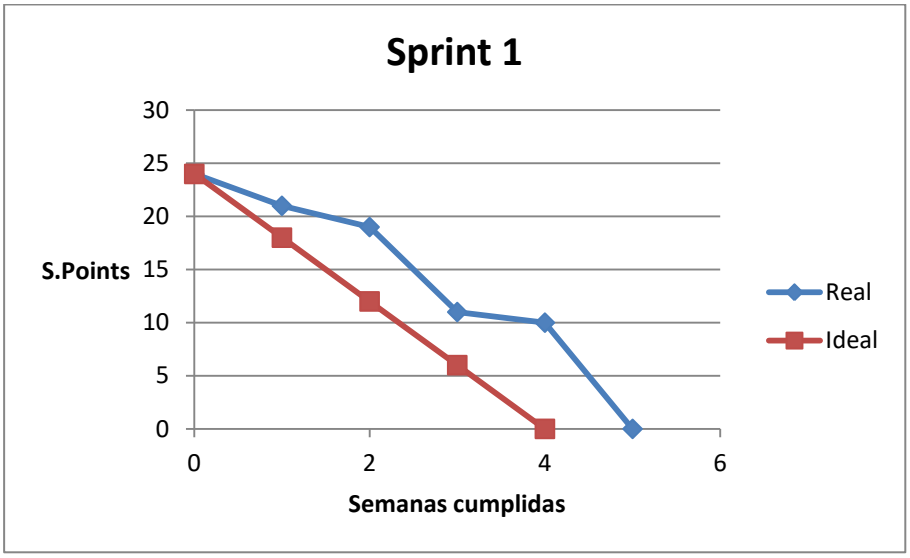

FIGURA 1 - BURNDOWN CHART DEL SPRINT N°1.

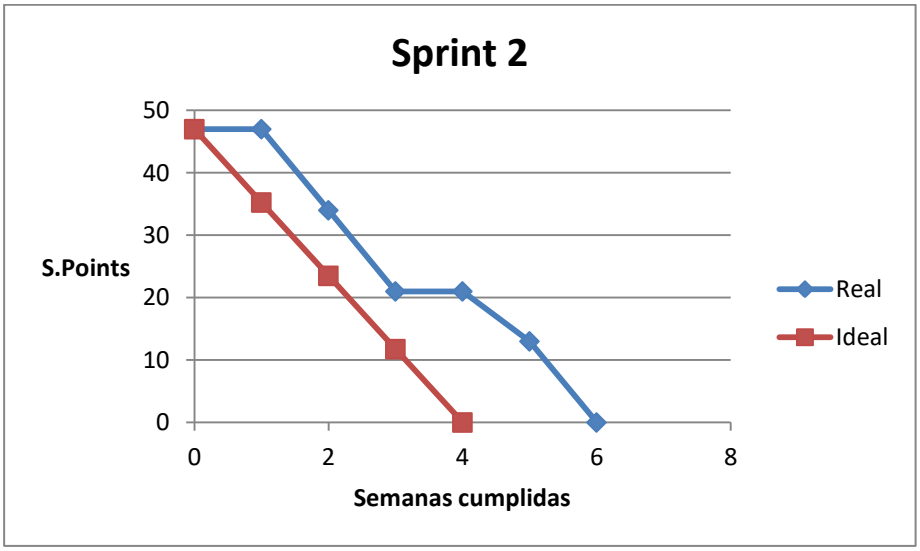

FIGURA 2 - BURNDOWN CHART DEL SPRINT N°2.

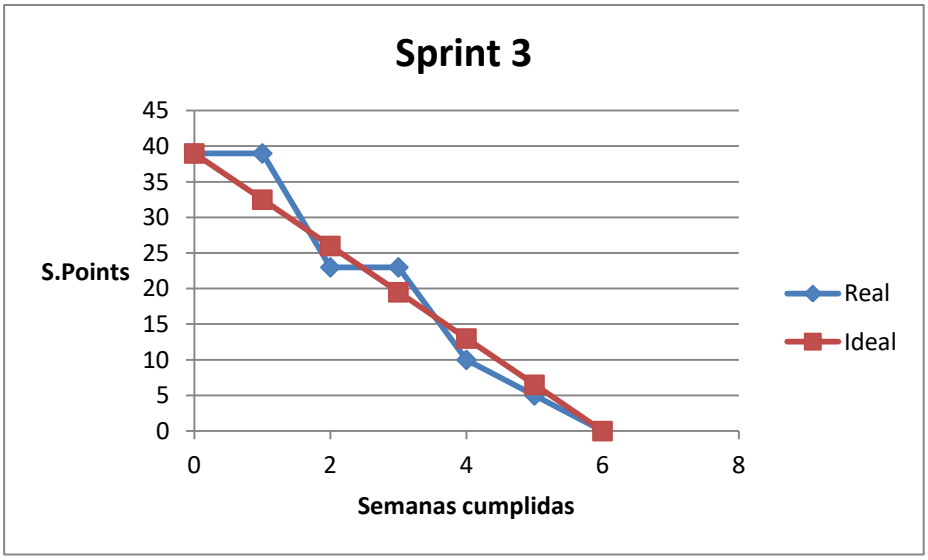

FIGURA 3 - BURNDOWN CHART DEL SPRINT N°3.

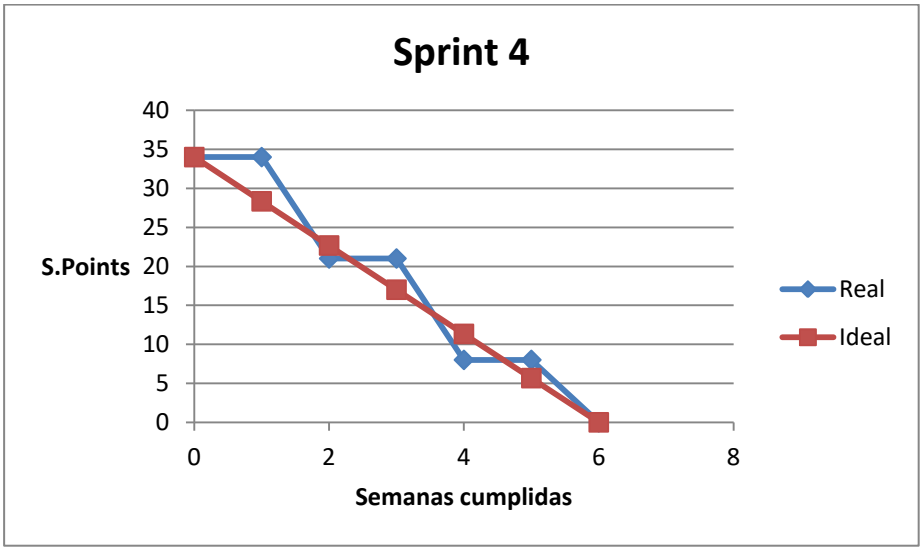

FIGURA 4 - BURNDOWN CHART DEL SPRINT N°4.

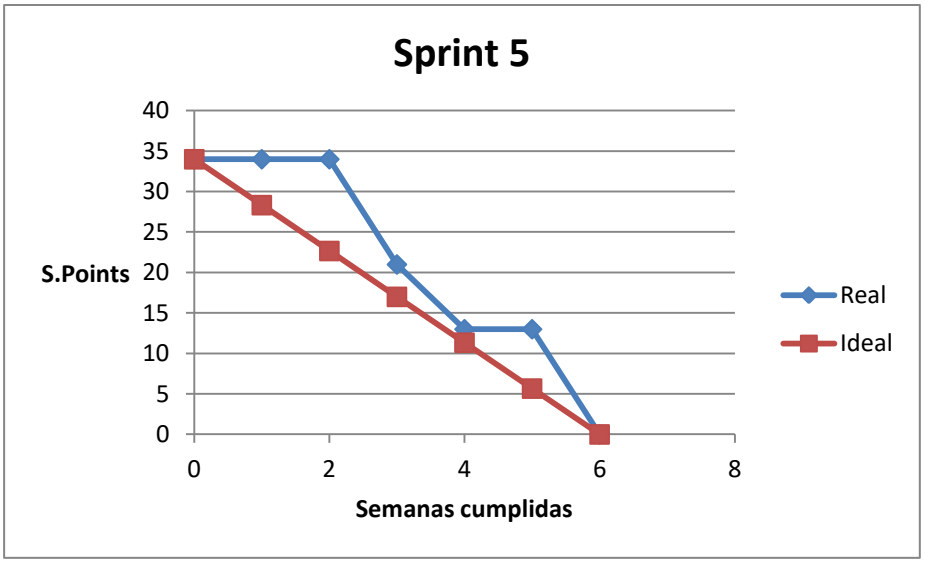

FIGURA 5 - BURNDOWN CHART DEL SPRINT N°5.

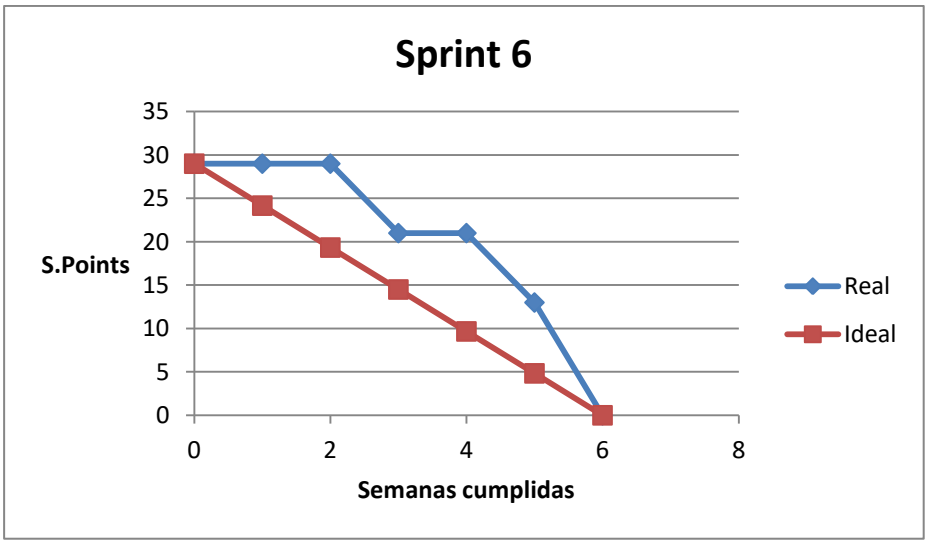

FIGURA 6 - BURNDOWN CHART DEL SPRINT N°6.

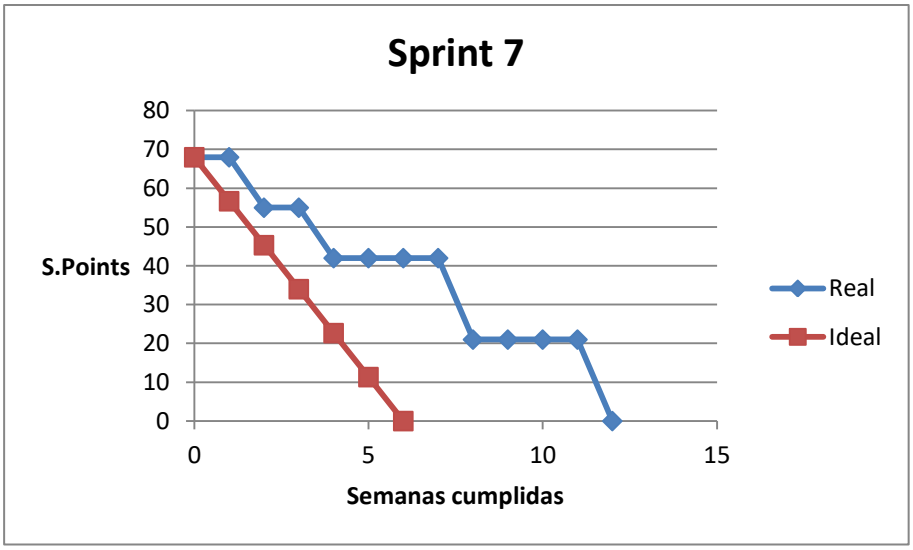

FIGURA 7 - BURNDOWN CHART DEL SPRINT N°7.

ARQUITECTURA DE LA APLICACIÓN

Al iniciar el proyecto, se decidió integrar en una versión simplificada de la aplicación todos los conceptos que consideramos podían ser utilizados en futuras versiones de la misma. Es por ello que en la versión correspondiente al Sprint 1, integramos los siguientes componentes:

- OpenCV
- Engine 3D (al menos uno a modo de prueba inicial)
- Implementación de código nativo vía Java Native Interface (JNI)[1].
- Servicios locales en segundo plano (background)
- Multithreading (en los servicios vía ThreadPoolExecutor)
- Varias vistas en forma de Fragments

Dada que esta fue la primera vez que los integrantes participaron del desarrollo de una aplicación para la plataforma Android, se decidió investigar acerca de los modelos de arquitectura más comunes y los patrones que se utilizan, partiendo de los conceptos fundamentales que componen una aplicación Android[2].

Debe tenerse en cuenta que el presente trabajo implica el desarrollo de dos aplicaciones: FIUBAAR Cliente y FIUBAAR Servidor. La complejidad se encuentra en la aplicación cliente y allí es donde se aplican los mayores esfuerzos, por lo que la documentación de la arquitectura está centrada también en ésta, mostrando detalles de la aplicación servidor cuando se lo considera necesario o conveniente.

ARQUITECTURA DE ANDROID

Para comenzar, resulta importante comprender en términos generales los componentes de la arquitectura de la plataforma Android[3].. La misma posee como capa base un sistema operativo Linux y posee un componente principal relacionado a las aplicaciones llamado Dalvik VM (Máquina Virtual). Cada aplicación es ejecutada en un proceso y el mismo posee una instancia de una DVM[4]. Es sin embargo importante mencionar que DVM evolucionó y, a partir de la versión 4.4 Kit Kat de Android, su reemplazo lleva el nombre de ART: Android Run Time. ART incorpora compilación AOT

[1] Como referencia visitar http://developer.android.com/training/articles/perf-jni.html.
[2] Consultar https://developer.android.com/guide/components/fundamentals.html.
[3] Para mayor detalle, http://www.tutorialspoint.com/android/android_architecture.htm.
[4] Consultar https://developer.android.com/guide/components/processes-and-threads.html.

(Ahead-of-Time) y un recolector de basura (GC, Garbage Collector), que fue mejorado. AOT implica que lo que se almacena y ejecuta en nuestros dispositivos ya no es Bytecode (o, mejor dicho, archivos .dex), sino código nativo. Esto deriva en una mejora en la velocidad, que particularmente se nota en el menor tiempo de inicio de las aplicaciones. Dalvik funcionaba con compilación JIT (Just-in-Time), de la cual se esperaba que compilara -y optimizara- el código al tiempo que lo leía (on-the-fly). La contrapartida de esto es que sucedía mientras ejecutábamos la aplicación, posiblemente afectando en forma negativa el desempeño. AOT hace lo mismo que JIT, pero una única vez, al momento de instalar la aplicación. Como mencionamos anteriormente, ART está presente a partir de la versión 4.4 de Android, aunque no por defecto, sino que debe ser habilitado[5].

EL PATRÓN MODEL-VIEW-PRESENTER

En general, para el desarrollo de aplicaciones se utiliza el patrón Model-View-Controller (MVC). Si bien la plataforma Android no implementa por defecto el patrón MVC[6], es posible implementarlo de forma muy coherente, tal como puede apreciarse en diversos ejemplos[7]. Para ser más estrictos, en el caso del presente proyecto, el patrón que se utilizó en realidad es Model-View-Presenter (MVP).[8]. MVP es un derivado, o en algunos casos considerado un subgrupo de MVC. El mismo aplica mejor a la lógica que quisimos utilizar, donde el modelo no notificará directamente a las vistas, sino que los controladores centralizan todo y son éstos los que actualizan y notifican a las vistas. Por esta razón se determinó el uso de MVP en la aplicación.

PERFORMANCE

Inicialmente se construyó una aplicación que empleó funcionalidades básicas de la librería OpenCV como capturar video de una cámara, capturar imágenes, aplicar filtros de colores, etc. Ésta debió evolucionar hacia una aplicación que pudiera realizar procesamiento intensivo al momento de desarrollar toda la funcionalidad comprometida, por lo que requirió especial atención al buen manejo de la performance.

---

[5] Para habilitar AOT deben seguirse los siguientes pasos: Settings → Developer Options → Select Runtime → Use ART.
[6] http://stackoverflow.com/questions/2925054/mvc-pattern-in-android.
[7] http://www.therealjoshua.com/2012/07/android-architecture-part-10-the-activity-revisited/, http://mindtherobot.com/blog/675/android-architecture-message-based-mvc/, http://www.thinkmind.org/download.php?articleid=patterns_2013_1_20_70039.
[8] Consultar http://stackoverflow.com/questions/19996963/difference-between-asp-net-mvc-and-mvp-are-they-both-same.

Por ello debimos contemplar la opción de implementar servicios que se ejecutaran en segundo plano (*background*), con múltiples hilos (*threads*) para realizar diferentes tareas fuera del hilo (*thread*) principal de la interfaz de usuario (UI). Android provee la posibilidad de que los servicios ejecuten en el mismo proceso de la aplicación o en otros procesos externos. Dependiendo de la forma en la que los servicios funcionen, se deberán utilizar diferentes formas de comunicación, tal como lo explica la documentación oficial[9].

Durante una etapa inicial de investigación, encontramos mucha información sobre cómo ejecutar servicios en otros procesos y cómo realizar la comunicación entre servicios y actividades[10]. Algo frecuentemente destacado en múltiples sitios (foros, blogs, guías, etc.), es que utilizar servicios en diferentes procesos y comunicar entre ellos vía IPC es costoso, con lo cual para poder tener mejor performance la mejor alternativa en nuestro caso es utilizar servicios locales.

Se realizó una primera investigación sobre la mejora de performance de la librería OpenCV de manera de maximizar el valor de cuadros por segundo (*FPS* o *Frames Per Second*) que se pueden obtener en Android. Una de las alternativas planteadas para mejorar la performance y FPS requiere hacer uso directo de la API de cámara de Android en lugar de hacer uso de la librería OpenCV. Esto permitiría obtener las vistas previas de cada cuadro (*frame*) evitando parte del costo adicional de performance (*overhead*) agregado por el uso de OpenCV y varios callbacks y listeners asociados.

En la aplicación inicial, entonces a modo de prueba, se implementó esta última propuesta junto con una versión que utiliza la cámara mediante clases de OpenCV para así poder realizar pruebas de performance de cada una y compararlas.

---

[9] Consultar https://developer.android.com/guide/components/services.html, https://developer.android.com/reference/android/app/Service.html.
[10] Consultar http://stackoverflow.com/questions/4083756/android-service-process-vs-not, http://stackoverflow.com/questions/2463175/how-to-have-android-service-communicate-with-activity, http://stackoverflow.com/questions/4300291/example-communication-between-activity-and-service-using-messaging, https://android-developers.googleblog.com/2010/04/multitasking-android-way.html, http://mindtherobot.com/blog/37/android-architecture-tutorial-developing-an-app-with-a-background-service-using-ipc/, http://www.techotopia.com/index.php/Android_Remote_Bound_Services_%E2%80%93_A_Worked_Example, http://stackoverflow.com/questions/15524280/service-vs-intentservice.

EVOLUCIÓN DE LA ARQUITECTURA

Al finalizar el desarrollo del proyecto, se realizó un análisis en retrospectiva. A continuación, se exponen las instancias por las que ha atravesado su arquitectura desde el inicio. Se enfatiza *cuál* fue la solución adoptada originalmente, *qué* se decidió hacer en su lugar, en caso de requerir modificación y *por qué* se tomó esta decisión.

ARQUITECTURA 1 - INICIAL

Al comienzo del proyecto, el equipo se propuso utilizar la librería OpenCV para el procesamiento de imágenes y determinó que para poder proveer una buena performance haría uso de múltiples hilos (*threads*). Mediante el uso de servicios que ejecutarían tareas en paralelo. Dichas tareas se realizan, al tiempo que el video es mostrado directamente al usuario. Con esto esperábamos lograr que cada cuadro del video se mostrase inmediatamente, mientras que se realizan todas las acciones de procesamiento de imágenes en paralelo. Teniendo así, video en tiempo real, sin los previsibles retrasos motivados por el costo de dicho procesamiento.

Concluidos la investigación referente a los marcadores de realidad aumentada y las pruebas de implementaciones existentes, se decidió hacer uso de una librería llamada Aruco. Ésta contaba con características que suponían contribuir a una mejor performance y facilitar la implementación de la detección de un marcador propio. La implementación elegida de Aruco se encontraba desarrollada en lenguaje C, y su utilización era posible gracias al uso de Java Native Interface (JNI) que permite la integración de llamadas entre código Java y código nativo (compilado mediante el uso de Android NDK). El uso de código nativo *suele* resultar en mayor eficiencia -en términos de tiempo de procesamiento- que su equivalente en Java. Razón por la cual, estando involucrando el procesamiento de imágenes con OpenCV, que implicaba operaciones costosas en este sentido, no dudamos en optar por la librería Aruco.

Sin embargo, a lo largo del proyecto el equipo fue encontrando ciertos problemas con Aruco, detectando errores en su manejo de memoria que causaban que la aplicación se cerrara o congelara inesperadamente. Luego de invertir mucho tiempo en pruebas y depuración, se identificaron las causas de esos problemas, pero no se encontraba una solución que dotara a dicha librería de la suficiente confiabilidad y estabilidad como para construir todo un desarrollo sobre ella. No obstante, se decidió avanzar con otras tareas del calendario para más tarde darle una nueva oportunidad a Aruco e intentar resolver sus problemas con el manejo de memoria.

A esta altura del proyecto, mediante el desarrollo de implementaciones personalizadas, y haciendo uso de la librería Aruco, se lograba detectar los marcadores definidos para FIUBAAR y hacer un seguimiento (*tracking*) de ellos. Aunque aún no se encontraba implementada la solución para la lectura del código QR comprendido en el interior de los mismos. Para ello se investigó sobre las diferentes librerías disponibles para reconocer y leer códigos QR, con implementaciones tanto en Java como en código nativo (C/C++), optando por utilizar la librería ZBar de implementación nativa. De este modo se facilitó la integración con Aruco mediante el uso de las callbacks que dicha librería ya proveía.

ARQUITECTURA 2

En el Sprint 4, se presentó la necesidad de integrar el motor 3D con la detección de marcadores para poder actualizar las posiciones de los modelos 3D cargados, de modo de reflejar los cambios de dirección y ángulo del marcador. Aquí surgió un desafío importante, ya que esto implicaba el uso de algoritmos con un significativo nivel de complejidad, relacionados al manejo de diferentes matrices representando planos y direcciones. Dicha integración mostró no ser tan simple de alcanzar, probando que la estimación de tiempos asignados a la tarea del calendario no fuera acertada.

En ese momento, el equipo se encontraba investigando activamente respecto a implementaciones de Realidad Aumentada de todo tipo tales como ejemplos básicos, librerías implementaciones para Android y otras plataformas de escritorio (Windows y Linux), integraciones con motores 3D, etc.

Como motor 3D, se había decidido hacer uso de del proyecto Rajawali y se encontraron desarrollos que ya integraban el uso de dicho motor 3D con la librería Vuforia de realidad aumentada.

Al investigar y poner a prueba las características de Vuforia, también se analizó su integración con el motor 3D elegido, tanto en aplicaciones de ejemplo como integrándolo -a modo de ensayo- en el prototipo implementado y disponible hasta ese momento de FIUBAAR. Fue notoria la simplicidad de integración y la mejora en cuanto a la detección de los marcadores, sumándose además la ventaja de eludir por completo los problemas de memoria observados en la etapa anterior, ya que se evitaba el uso de Aruco.

Por todo lo antes descrito, se decidió entonces reemplazar el uso de la librería Aruco con Vuforia para realizar con ello el seguimiento de marcadores, perfeccionando a continuación su integración con el motor 3D Rajawali.

Como resultado de este cambio, fue necesario también adaptar la lectura del código QR y se realizó el cambio de la librería ZBar implementada en código nativo por su versión implementada en Java.

ARQUITECTURA 3

Durante el Sprint 5 el equipo se encontraba realizando pruebas con diferentes implementaciones para realizar la detección y seguimiento de manos y dedos, resultado de la investigación realizada en el Sprint anterior. De acuerdo a la planificación, al final de este quinto sprint se esperaba contar con una versión de la aplicación FIUBAAR cliente que pudiera realizar la detección de manos y dedos.

Se decidió integrar en simultáneo dos implementaciones de un algoritmo bastante similar, pero implementado en un caso en C (nativo) y en otro en Java, con la intención de poder realizar pruebas exhaustivas de funcionamiento y performance para así elegir la mejor alternativa.

La implementación en Java se pudo integrar de manera simple, ya que existía un proyecto en el que se contaba con una aplicación Android completa.

En cambio, la implementación nativa tuvo que ser adaptada ya que originalmente fue pensada para plataformas de escritorio. En este caso se implementó además el código necesario para el uso de JNI, con el que se lograba la interacción entre código Java y nativo.

Todavía la arquitectura general de FIUBAAR cliente contaba con un servicio que poseía un conjunto de hilos (*thread pool*) que podían ejecutar tareas en paralelo procesando cuadros (*frames*) de video. El servicio hacía uso de "tareas" (*tasks*) que procesaban estos cuadros de video para detectar en ellos la existencia de manos y dedos, haciendo uso de las dos implementaciones antes mencionadas.

Fue este el momento en el que el equipo se encontró con uno de los mayores inconvenientes del proyecto hasta el momento, el cual generó considerables retrasos en el calendario, así como también la necesidad de investigación, desarrollo y prueba de nuevas ideas para la arquitectura de la aplicación.

Los inconvenientes encontrados se referían, en principio, a que las tareas que se debían ejecutar en paralelo para detectar manos y dedos demoraban demasiado. Dichas demoras, en un principio, parecían deberse a las prioridades de hilos secundarios respectos del hilo principal. Dicho hilo presenta mayor prioridad dado que tiene como mayor responsabilidad el manejo de la interfaz gráfica (UI) con fluidez. También se pudo observar que, debido al uso del conjunto de hilos, era muy probable no poder asegurar que los cuadros de video se procesaran en el orden secuencial original. Encontrándose, entonces, situaciones sin lógica en la secuencia de puntos detectados.

Durante esta etapa el equipo realizó la prueba de las varias alternativas de programación concurrente que presenta la plataforma Android. Se implementaron y probaron las siguientes opciones:

- Servicios de Android
- En segundo plano (Background)
- En primer plano (Foreground), mayor prioridad.
- En el mismo proceso
- En un proceso externo a la aplicación
- ThreadPoolExecutor (manejador de hilos y colas de tareas)
- AsyncTasks y FutureTasks
- Looper thread
- HandlerThread
- Ejecución de Runnables en el hilo principal UI

Para todos estos casos se intentó utilizar uno o múltiples hilos, así como también intentar aprovechar al máximo el uso de los múltiples núcleos del procesador en los casos en los que el dispositivo presentara dicha capacidad.

La única alternativa que no se llegó a implementar y probar fue el uso de "programación reactiva" mediante el uso de RxJava, que es una librería que implementa el patrón observador y el patrón iterador en conjunto con ideas de programación funcional. El uso de programación reactiva parece estar en auge en el último tiempo, pero no se había contemplado con anterioridad al inicio del presente proyecto. Al momento en el que el equipo descubrió y se interiorizó en el uso de la librería RxJava, la reestructuración necesaria del proyecto, incluso para realizar simples pruebas, hubiera tomado demasiado tiempo que afectaría en gran medida el calendario propuesto.

Fue preocupante, durante las pruebas de cada una de las alternativas, el hecho de que se agregaba una demora significativa en la detección de manos y dedos. Esto se manifestaba en que, al momento de encontrarlos, la posición real de la mano del usuario ya había cambiado, no logrando nunca una detección que presente tiempos de respuesta aceptable.

Toda esta investigación, implementación y pruebas se extendieron hasta el Sprint 6 y tomó todo el tiempo asignado al mismo, ya que de todas formas la tarea de la implementación de detección de manos y dedos correspondía al mismo. Nuevamente, esto impactó de manera apreciable en el tiempo estimado para el calendario tentativo propuesto.

Luego de muchas tareas de pruebas de diferentes implementaciones y cambios en varios puntos de la aplicación, se decidió que la arquitectura pensada inicialmente -para ganar performance haciendo uso de hilos paralelos- no era viable. Razón por la cual, se optó por procesar cada uno de los cuadros de video en el hilo principal y en el orden original en el cual es provisto por Vuforia.

Dicho cambio fue importante y requirió la reestructuración (*refactoring*) de gran parte de la implementación y la reagrupación de algunos paquetes.

A pesar de este gran cambio en la arquitectura, se mantuvo el código que implementa el servicio con hilos paralelos, ya que la probabilidad de resultar útil para la detección de gestos era considerable.

ARQUITECTURA 4 - FINAL

Durante el Sprint 6, tras investigar sobre detección de gestos en el Sprint 5 y analizar posibles implementaciones para lograr una operación eficiente se concluyó que se encontraría el mismo problema que con la detección de la mano y sus dedos, en el caso de hacer uso de hilos paralelos. Ya que podrían generar el mismo retraso en la detección de gestos que el generado en la detección de mano y dedos.

Es así, entonces, que el servicio con un conjunto de hilos existe en el proyecto, aunque actualmente, no está siendo iniciado ni utilizado para ninguna tarea de procesamiento de imágenes sobre los cuadros de video.

En un principio se contempló la idea de implementar algún algoritmo simple con el cual se pudiera detectar gestos en base a las coordenadas de los dedos detectados en cada

cuadro. Así, se procedió a definir cuál sería la secuencia básica esperada para cada uno de los gestos que se propuso soportar en la aplicación. Al mismo tiempo, se investigó al respecto del reconocimiento de gestos en aplicaciones Android basados en la interacción con la pantalla del dispositivo. Durante esta investigación se descubrió que era posible generar programáticamente los "eventos de toque en pantalla" (*motion touch events*) especificando coordenadas arbitrarias deseadas. Al realizar pruebas con dichos eventos se pudo comprobar que era viable simular una secuencia completa de un usuario interactuando con la pantalla del dispositivo. Para la detección de gestos básicos, entonces, se decidió crear una clase controladora que procese los dedos detectados mediante el análisis de las imágenes de video y genere eventos de toque en pantalla como si el usuario estuviera interactuando en contacto físico con la misma. Esto permite aprovechar ciertas APIs provistas por la plataforma Android que de por sí pueden procesar y manejar gestos como el "pinchado" (*scale*). Esta clase controladora debe implementar la lógica necesaria para manejar los diferentes tipos de eventos de tocado para la plataforma Android (ACTION_DOWN, ACTION_MOVE, ACTION_UP, y sus variantes para eventos "multi-touch" que involucran más de un dedo).

Gracias a que Android ya posee APIs para manejar ciertos gestos, se logra simplificar la lógica que inicialmente se pensaba implementar, delegando directamente en las mismas el reconocimiento de los gestos y permitiendo además una integración más estandarizada para la plataforma en base a métodos "listeners" que son invocados de manera asincrónica cuando algún gesto sucede.

A pesar de esta nueva y prometedora idea, Android al momento del desarrollo del presente proyecto, parece proveer una clase completa solo para el reconocimiento del pinchado (*scale*) y para el caso de otros gestos uno debe agregar un poco de código adicional. Se encontraron librerías de código abierto que permiten reconocer otros gestos y luego de realizar pruebas con las mismas se decidió incluir la librería "Almeros Android multitouch gesture detectors" que permite, por ejemplo, rápidamente reconocer gestos de rotación e incluso gestos con múltiples dedos en pantalla.

El código de la aplicación cliente se distribuye en los siguientes paquetes y librerías:

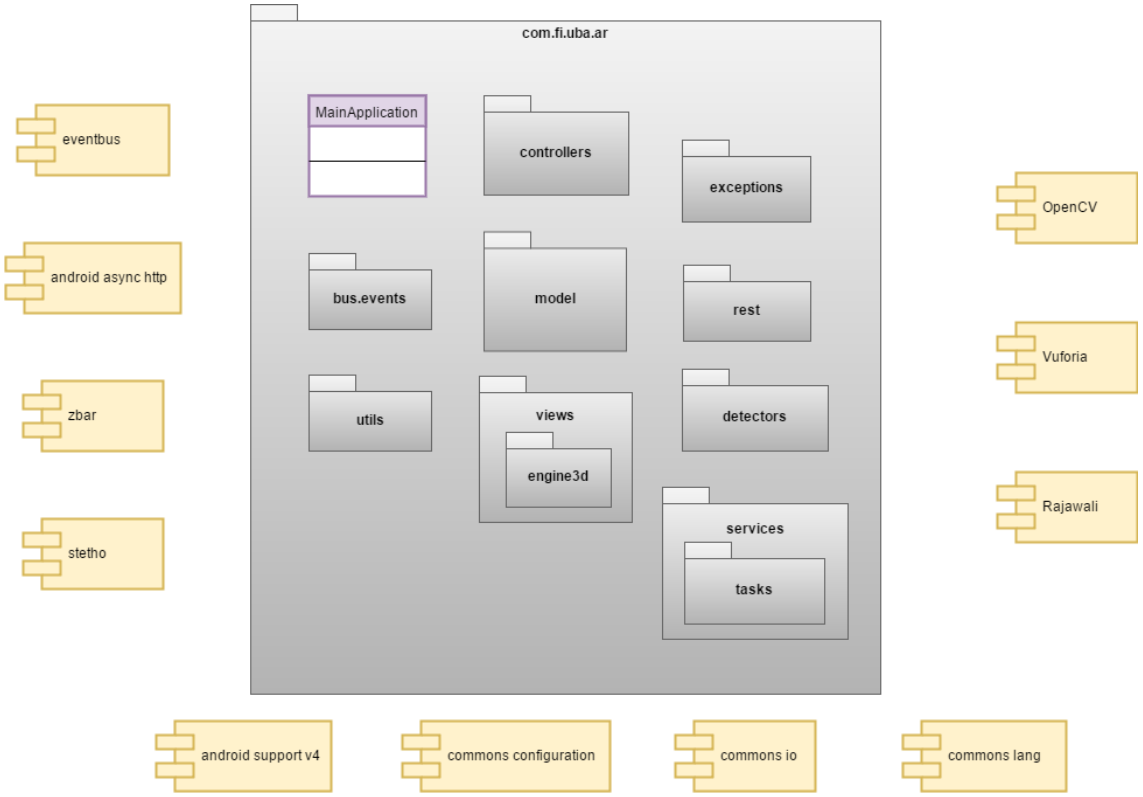

FIGURA 8. VISTA GENERAL DE PAQUETES Y COMPONENTES DE FIUBAAR CLIENTE.

TABLA 1 – DESCRIPCIÓN DE LOS PAQUETES DE FIUBAAR CLIENTE.

Paquetes de código fuente	
Nombre del Paquete	Descripción
com.fi.uba.ar.controllers	Agrupa las clases controladoras que implementan el componente “Presenter” del patrón MVP. Las clases de este paquete son las que contienen la lógica principal y fundamental de la aplicación FIUBAAR cliente.
com.fi.uba.ar.model	Agrupa las clases que implementan el “modelo” del patrón MVP.
com.fi.uba.ar.views	Agrupa las clases que implementan las vistas del patrón

	MVP. Las vistas son “fragmentos” de Android que se utilizan para mostrar al usuario las diferentes pantallas que componen la UI.
com.fi.uba.ar.view.engine3d	Agrupa las clases que implementan vistas únicamente relacionadas al motor 3D y objetos 3D.
com.fi.uba.ar.rest	Agrupa las clases que implementan la lógica necesaria para la conexión e interacción con la aplicación FIUBAAR servidor y permiten la descarga de modelos 3D.
com.fi.uba.ar.detectors	Agrupa las clases que implementan la detección de marcadores, manos y dedos, así como también los diferentes gestos soportados.
com.fi.uba.ar.services	(Deprecado) Agrupa las clases que implementan los servicios que se ejecutan de fondo y permiten procesar acciones en paralelo.
com.fi.uba.ar.services.tasks	(Deprecado) Agrupa las clases que implementan las acciones que se pueden ejecutar en los servicios de fondo.
com.fi.uba.ar.utils	Agrupa clases que implementan diferentes operaciones de utilidad general para diferentes aspectos de la aplicación, tales como: registro de mensajes (*logs*), manejo de archivos, operaciones de transformación de imágenes, etc.
com.fi.uba.ar.exceptions	Agrupa las clases que implementan excepciones personalizadas de la aplicación.
com.fi.uba.ar.bus.events	Agrupa las clases que implementan eventos personalizados que pueden ser utilizados para comunicación interna en la aplicación mediante la librería EventBus.

TABLA 2 – DESCRIPCIÓN DE LOS COMPONENTES .JAR DE FIUBAAR CLIENTE.

Componentes binarios (.JAR)	
Nombre del componente	Descripción
OpenCV	Librería de código abierto (*open source*) de visión artificial destinada al procesamiento de imágenes.

Vuforia	Librería (SDK) de realidad aumentada para plataformas móviles que utiliza visión artificial y provee APIs para seguimiento de marcadores
Rajawali	Motor gráfico 3D de código abierto basado en tecnologías OpenGL ES.
zbar	Librería de código abierto que permite leer diferentes tipos de códigos (QR, código de barras, etc.) a partir de videos o imágenes.
Android async http	Librería de código abierto que provee un cliente para realizar conexiones HTTP e interactuar de manera asincrónica mediante callbacks.
Stetho	Librería de código abierto desarrollada por Facebook que permite realizar la depuración interactiva remota mediante un navegador web.
Event Bus	Librería de código abierto que permite implementar notificaciones centralizadas dentro de la aplicación mediante el envío de eventos a un bus de datos.
Android Support V4	Librería que ofrece versiones de funciones nuevas que son compatibles con versiones anteriores de Android y proporciona elementos de UI útiles que no se incluyen por defecto en el framework Android.
Apache Commons Configuration	Librería de código abierto que permite trabajar con archivos de configuración en diferentes formatos
Apache Commons IO	Librería de código abierto que provee funciones generales para el trabajo con archivos.
Apache Commons Lang	Librería de código abierto que provee funciones útiles para manipulación de Strings, funciones numéricas, concurrencia, serialización y acceso a propiedades de Sistema entre otras cosas.

El diagrama a continuación muestra las asociaciones entre paquetes y librerías, destacándose la implementación del patrón MVP. También se identifican las librerías externas de mayor relevancia.

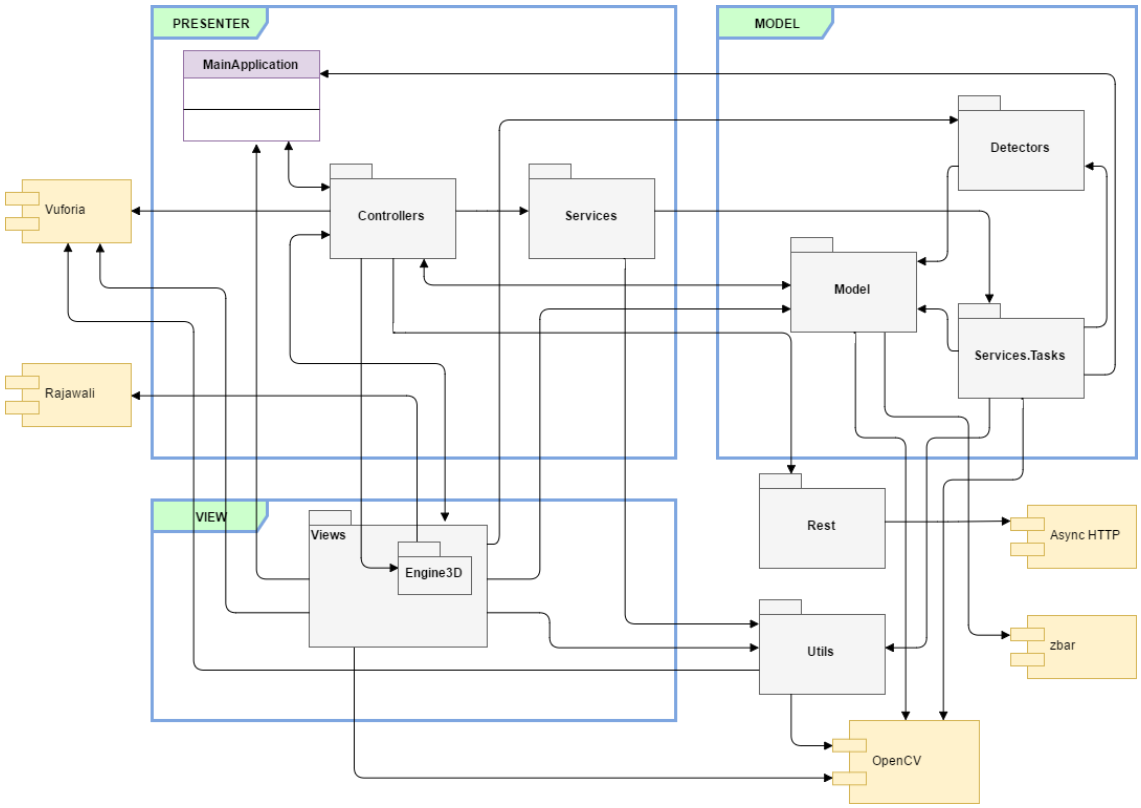

FIGURA 9 – RELACIONES ENTRE PAQUETES ENFATIZANDO IMPLEMENTACIÓN DE MVP.

El siguiente diagrama explicita las clases contenidas dentro de cada paquete.

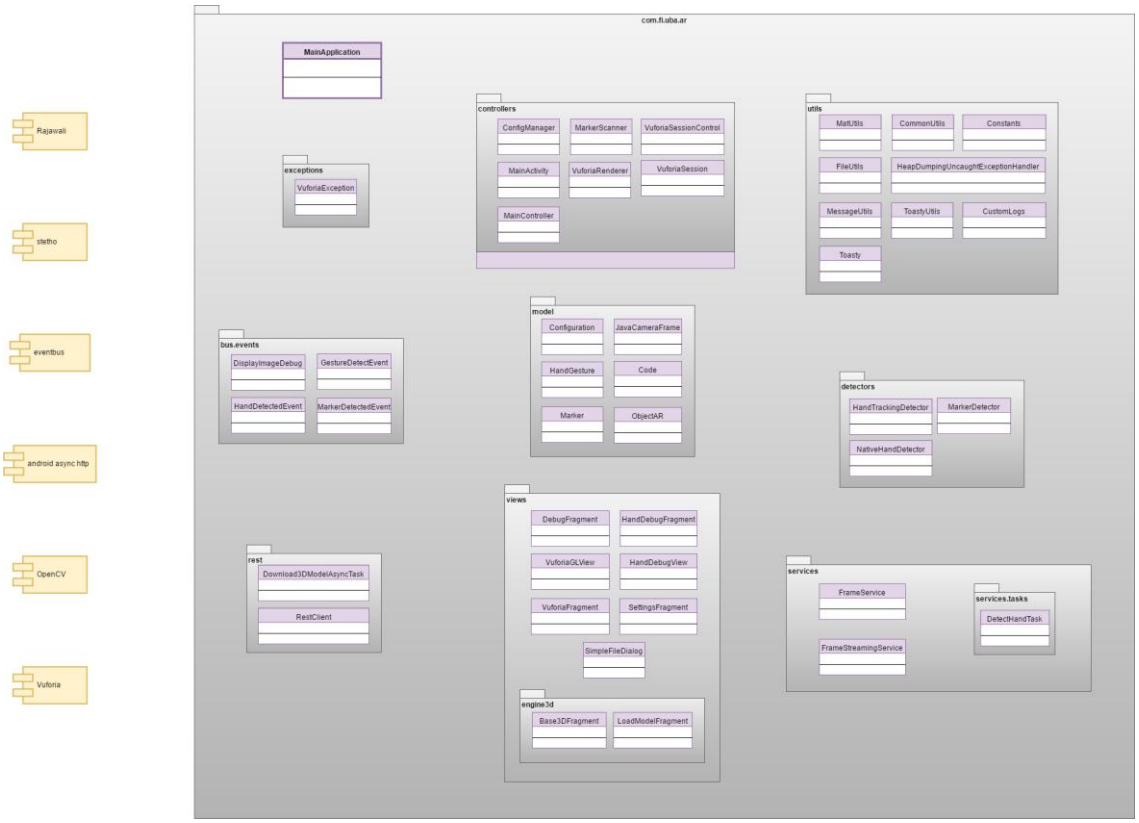

FIGURA 10 – CLASES COMPONENTES DE CADA PAQUETE.

TABLA 3 – DESCRIPCIÓN DE LAS CLASES COMPONENTES DE FIUBAAR CLIENTE.

Descripción de Clases		
Nombre del Paquete	Nombre de la Clase	Responsabilidad
com.fi.uba.ar	MainApplication	Manejo y persistencia del estado general de la aplicación.
com.fi.uba.ar.controllers	MainController	Componente central que maneja la lógica necesaria para orquestar las acciones de los usuarios y el correcto manejo los de objetos de realidad aumentada.
	MainActivity	Actividad principal de Android que representa la pantalla base de la interfaz de usuario de la aplicación.
	MarkerScanner	Reconocimiento y lectura de marcadores en la imagen provista por la cámara.
	VuforiaSession	Implementa la lógica necesaria para mantener una sesión de realidad aumentada dentro de la librería Vuforia.
	VuforiaSessionControl	Interfaz que define los métodos que deben implementar las clases que hacen uso de VuforiaSession.
	VuforiaRenderer	Procesa cada uno de los cuadros de video para realizar el seguimiento de marcadores y actualizar la posición acordemente.
	ConfigManager	Maneja y persiste la configuración general de la aplicación.
com.fi.uba.ar.model	Configuration	Mantiene los valores de configuración de la aplicación que son manejados por la clase

		ConfigManager
	JavaCameraFrame	Representa un cuadro de video que es provisto por la cámara para poder utilizarlo con la librería OpenCV.
	Marker	Representa y contiene toda la información provista por un marcador.
	Code	Representa el código interno dentro del marcador. (Deprecado)
	ObjectAR	Representa un objeto de realidad aumentada que asocia un marcador con un objeto 3D.
	HandGesture	Detecta y representa una mano y las posiciones de sus dedos. (Deprecado)
com.fi.uba.ar.views	VuforiaFragment	Muestra la imagen de video provista por la cámara en tiempo real haciendo uso de VuforiaGLView
	VuforiaGLView	Configura y maneja la superficie OpenGL para poder mostrar video e imágenes con Vuforia.
	HandDebugFragment	Mostrar el contenido de la vista HandDebugView.
	HandDebugView	Muestra al usuario las posiciones en las que se detectaron dedos, dibujando círculos cuya ubicación se actualiza constantemente.
	DebugFragment	Mostrar, a modo de depuración, imágenes que fueron procesadas y modificadas con OpenCV. (Deprecado)
	SettingsFragment	Muestra en pantalla todos los valores de configuración de la

		aplicación.
	SimpleFileDialog	Permite la búsqueda y selección de un archivo local en el dispositivo móvil.
com.fi.uba.ar.view.engine3d	Base3DFragment	Provee la estructura base para poder implementar una vista que muestre un objeto 3D.
	LoadModelFragment	Muestra un objeto 3D
com.fi.uba.ar.rest	RestClient	Permite realizar conexiones HTTP para interactuar con servicios REST en un servidor web
	Download3DModel AsyncTask	Descarga archivos de un servidor web.
com.fi.uba.ar.detectors	MarkerDetector	Permite la detección de un marcador dentro de un cuadro de video.
	NativeHandDetector	Permite la detección de manos y dedos en un cuadro de video (implementación nativa en C++ mediante JNI)
	HandTrackingDetector	Permite la detección de manos y dedos en un cuadro de video (implementación pura en Java - Deprecada)
com.fi.uba.ar.services	FrameService	Procesa de forma paralela los cuadros de video provistos por Vuforia.
	FrameStreamingService	Envía los cuadros de video a un servidor remoto.
com.fi.uba.ar.services.tasks	DetectHandTask	Implementa la acción -ejecutada por los servicios- que realiza el procesamiento de imagen necesario para detectar una mano.
com.fi.uba.ar.exceptions	VuforiaException	Representa un error generado

		por Vuforia al procesar imágenes.
com.fi.uba.ar.utils	MatUtils	Implementa funciones generales para trabajar con matrices de OpenCV
	CommonUtils	Implementa funciones útiles para conversión y compresión de datos binarios.
	MessageUtils	Provee funciones generales para realizar comunicaciones al usuario a través de diferentes medios.
	FileUtils	Provee funciones generales para el manejo de archivos de datos.
	Constants	Define valores estáticos utilizados en la aplicación
	CustomLogs	Implementa funciones para enviar mensajes al Logcat de Android.
	Toasty	Permite realizar notificaciones al usuario mediante Toasts de Android de manera personalizada.
	ToastyUtils	Expone funciones genéricas para creación de Toasts de Android de manera simple
	HeapDumpingUncaughtExceptionHandler	Maneja globalmente las excepciones no capturadas.
com.fi.uba.ar.bus.events	MarkerDetectedEvent	Representa un evento que notifica la detección de un marcador en una imagen.
	HandDetectedEvent	Representa un evento que notifica la detección de manos en una imagen.
	GestureDetectedEvent	Representa un evento que notifica la detección de gestos en una

		imagen.
	DisplayImageDebug	Representa un evento que permite el envío de una imagen de depuración entre componentes de la aplicación.

Se muestran a partir de este punto, diagramas de las clases más importantes que pretenden resaltar los componentes principales de cada una, así como también las clases con las que interactúan y aquellas que la referencian.

CLASE MAINAPPLICATION

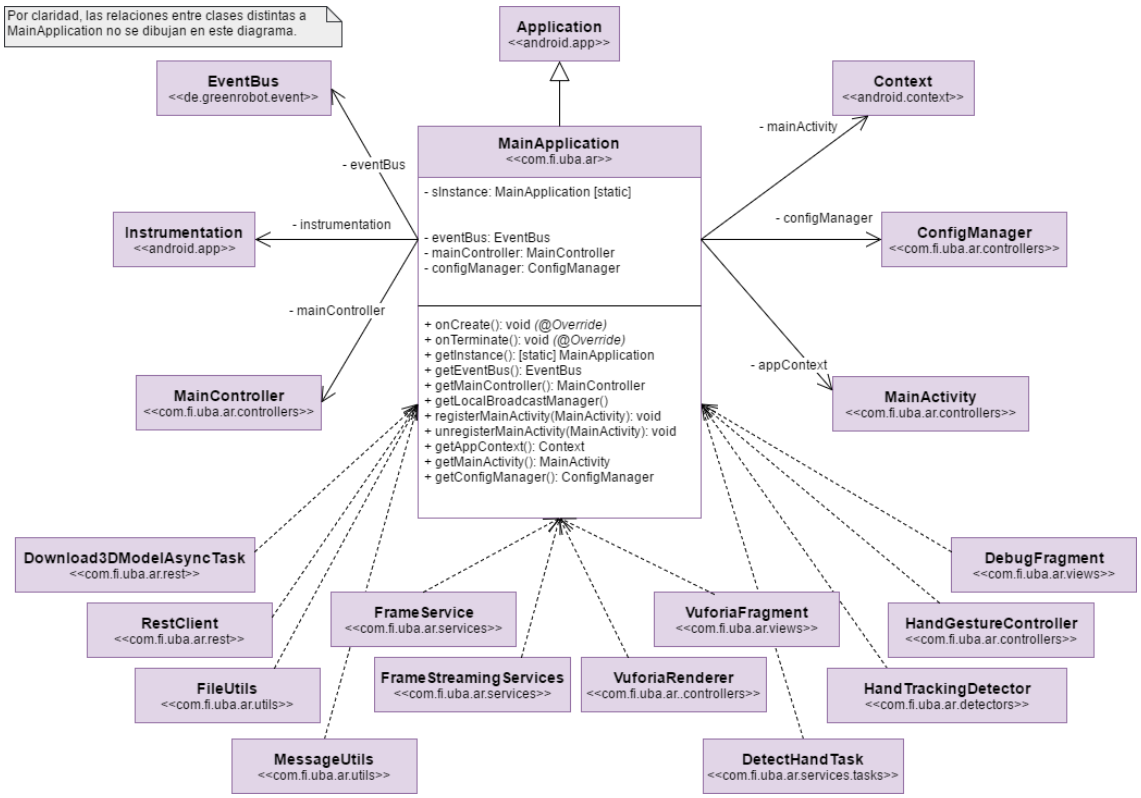

FIGURA 11 – CLASE MAINAPPLICATION.

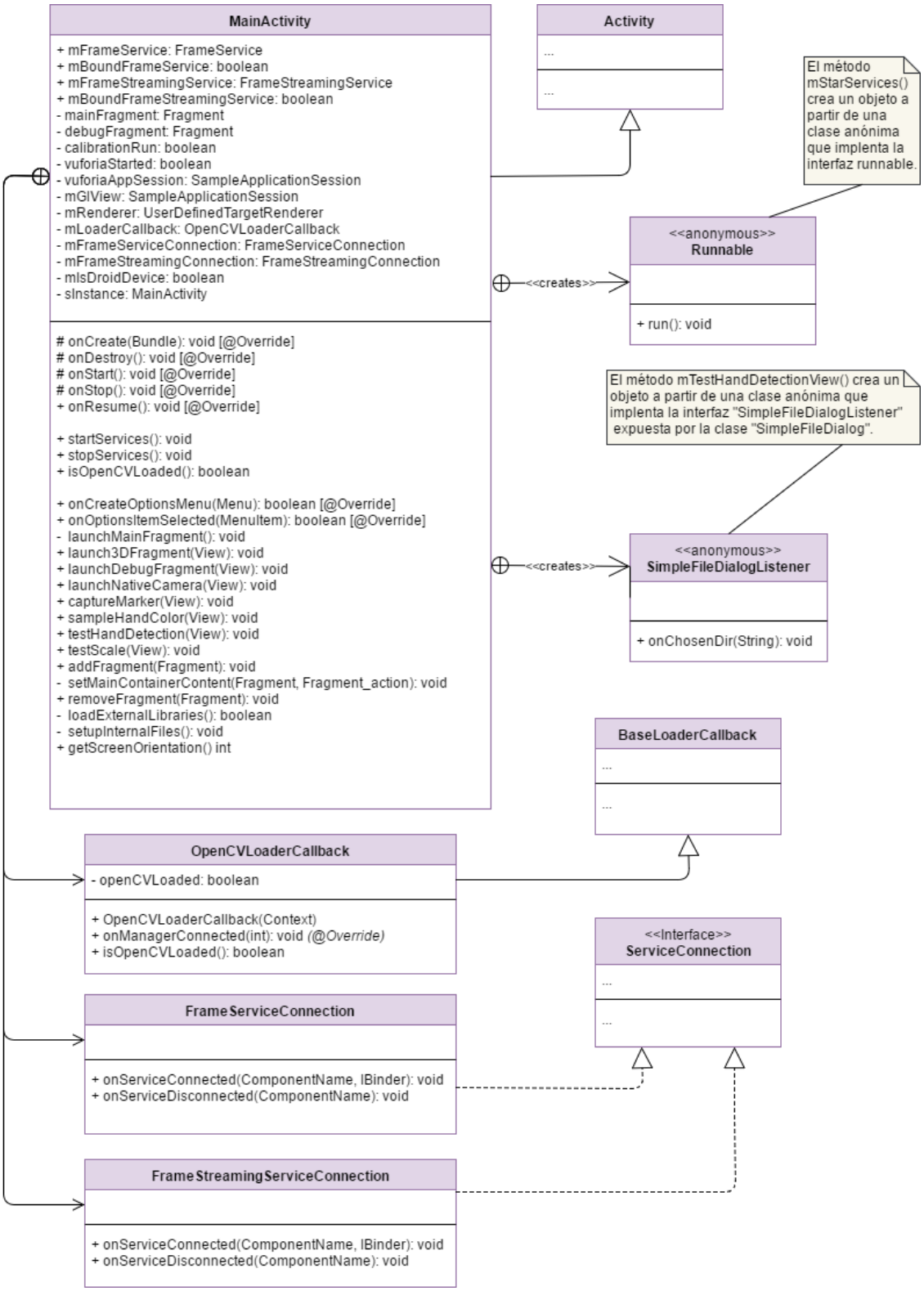

FIGURA 12 – CLASE MAINACTIVITY.

CLASE MAINCONTROLLER

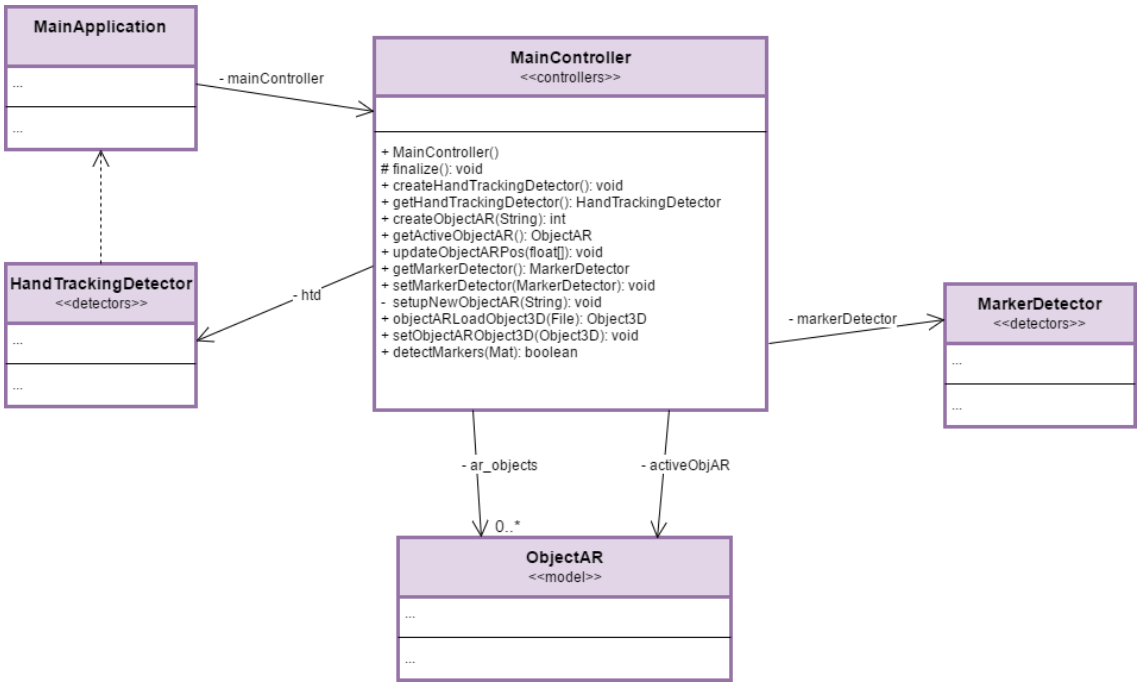

FIGURA 13 – CLASE MAINCONTROLLER.

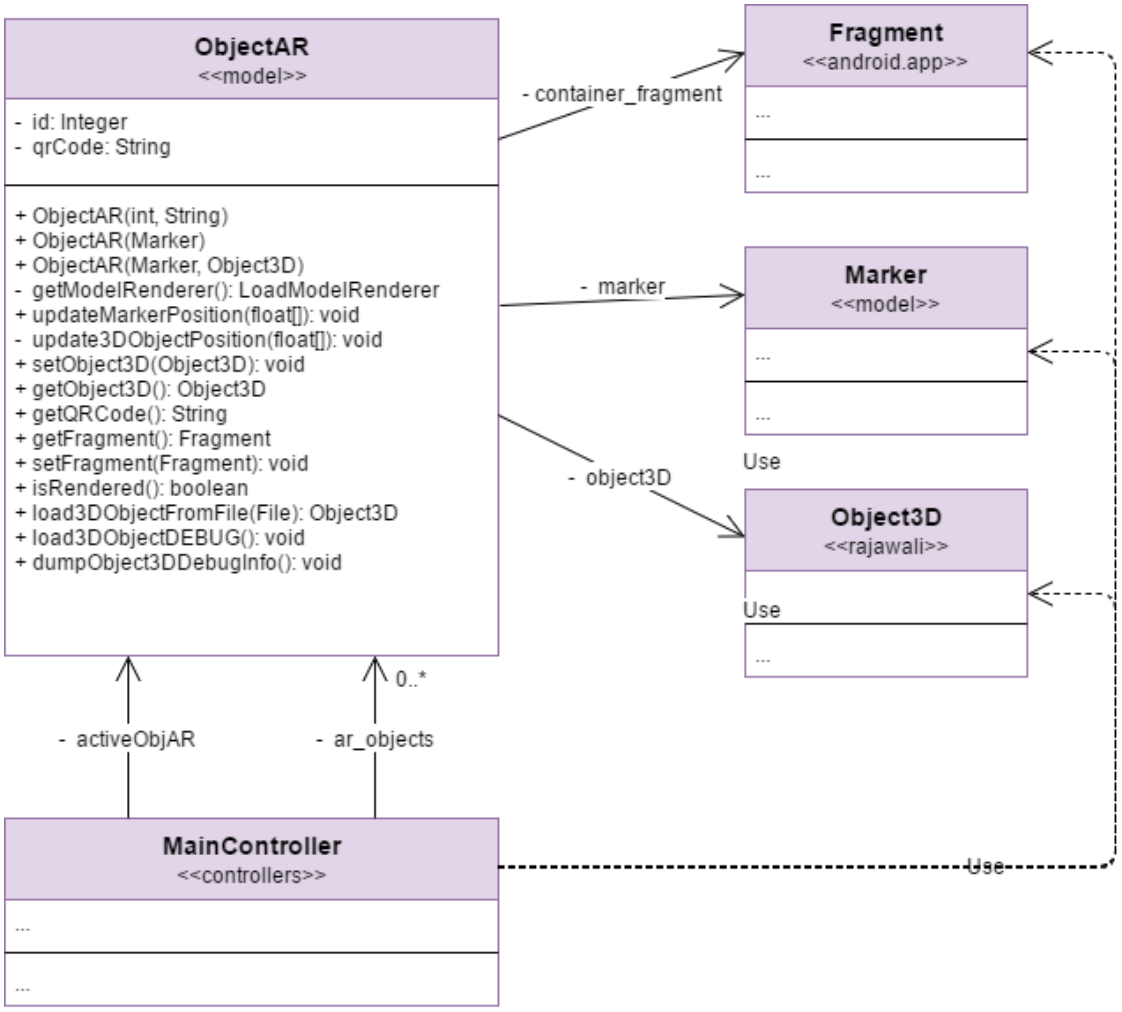

FIGURA 14 – CLASE OBJECTAR.

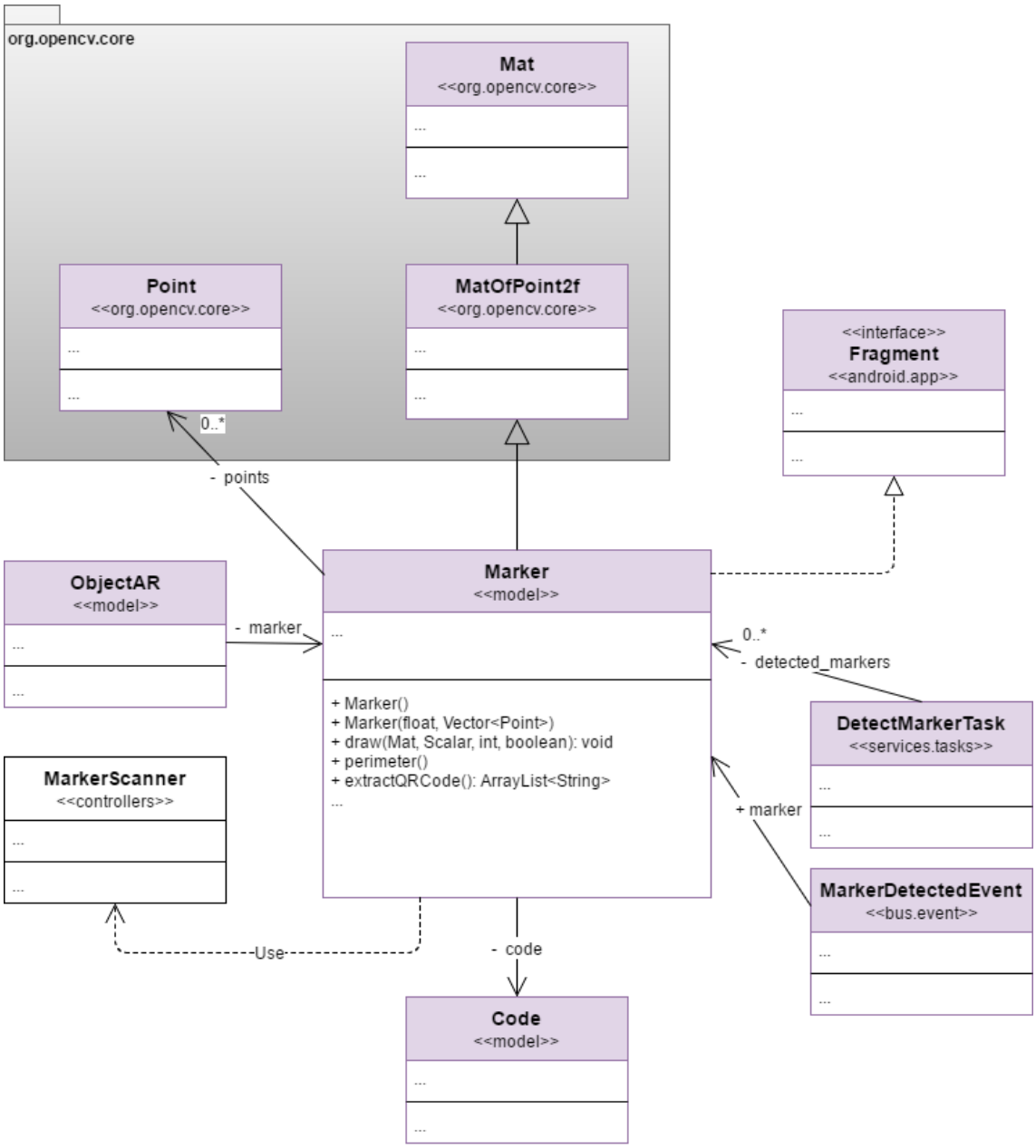

FIGURA 15 – CLASE MARKER.

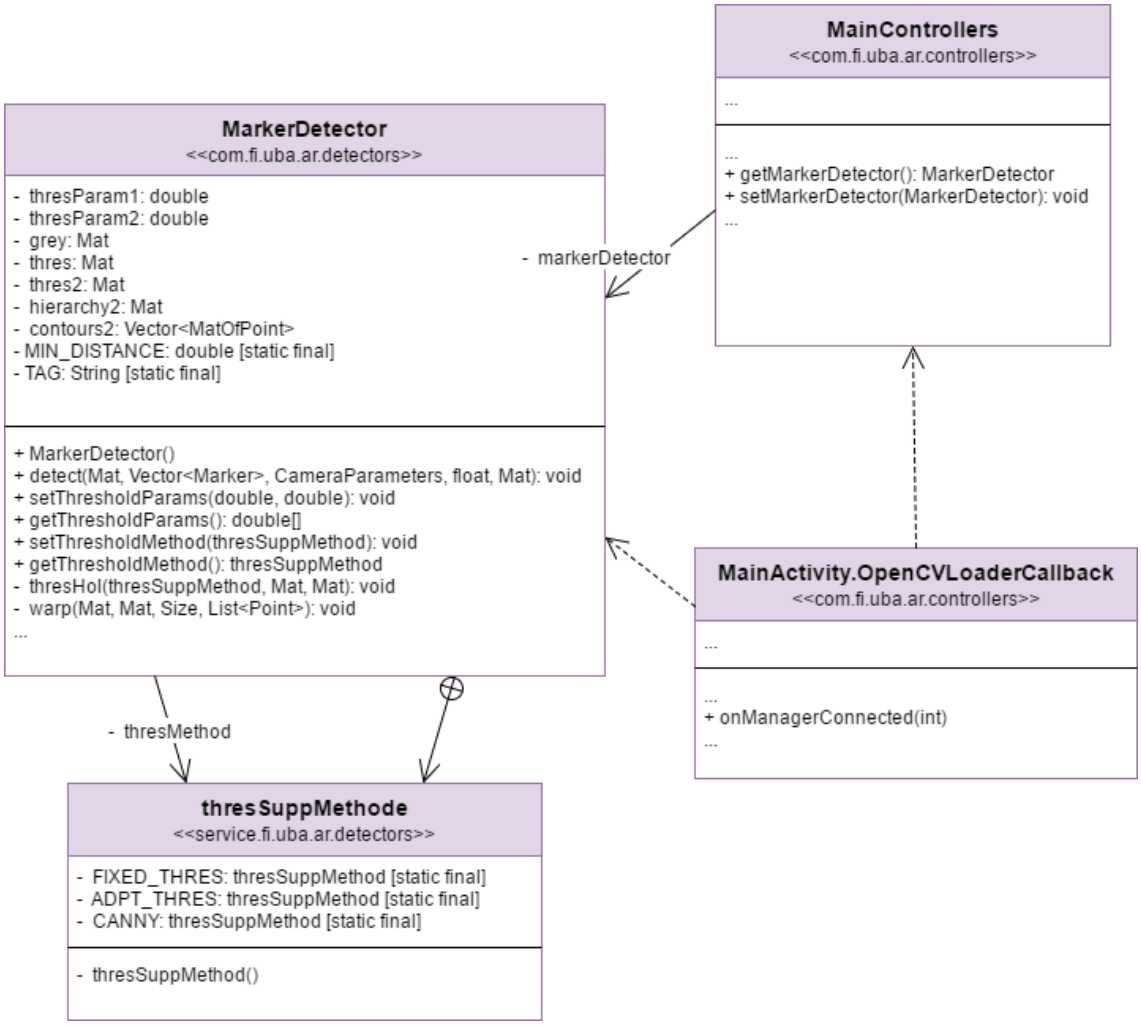

FIGURA 16 – CLASE MARKERDETECTOR.

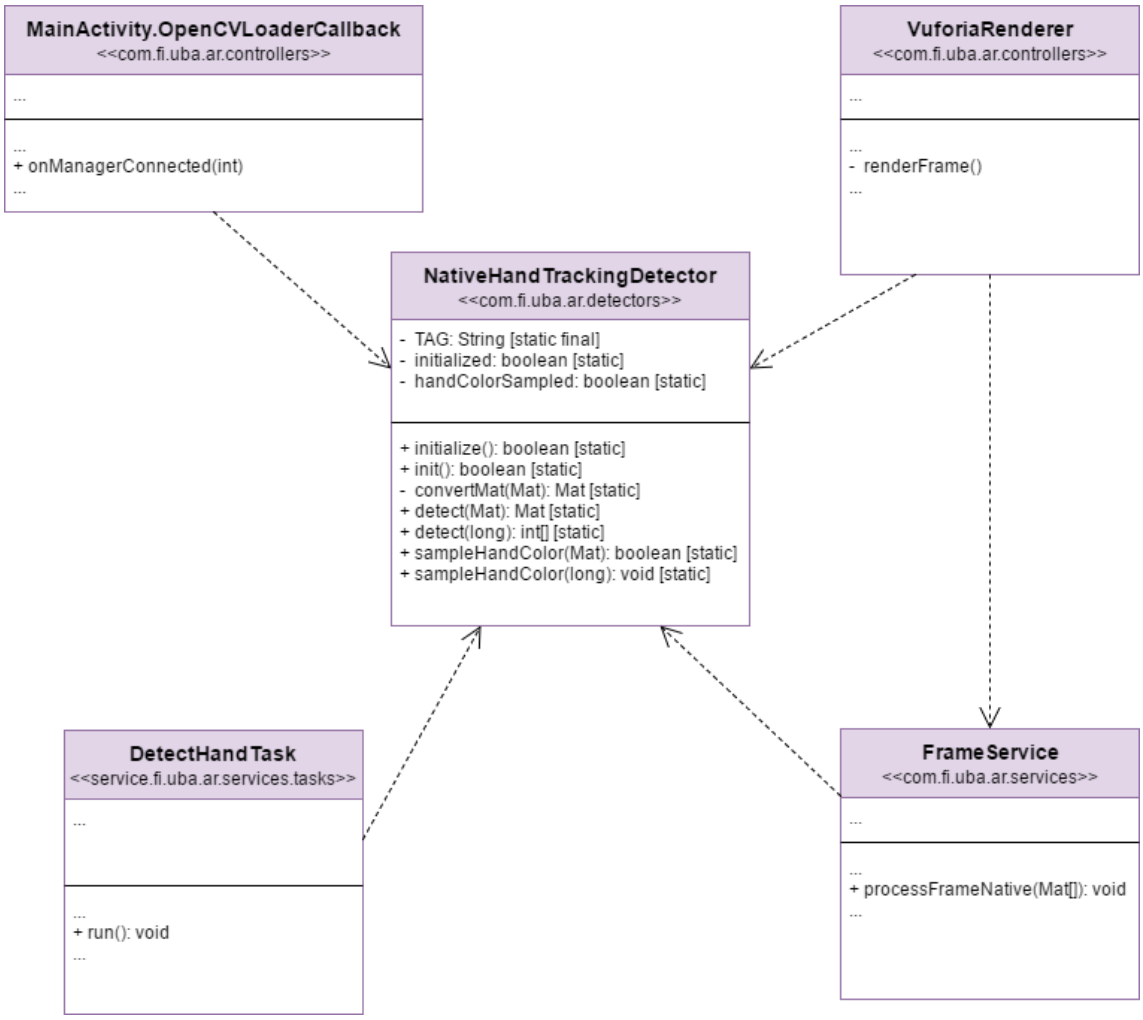

FIGURA 17 – CLASE NATIVEHANDTRACKINGDETECTOR.

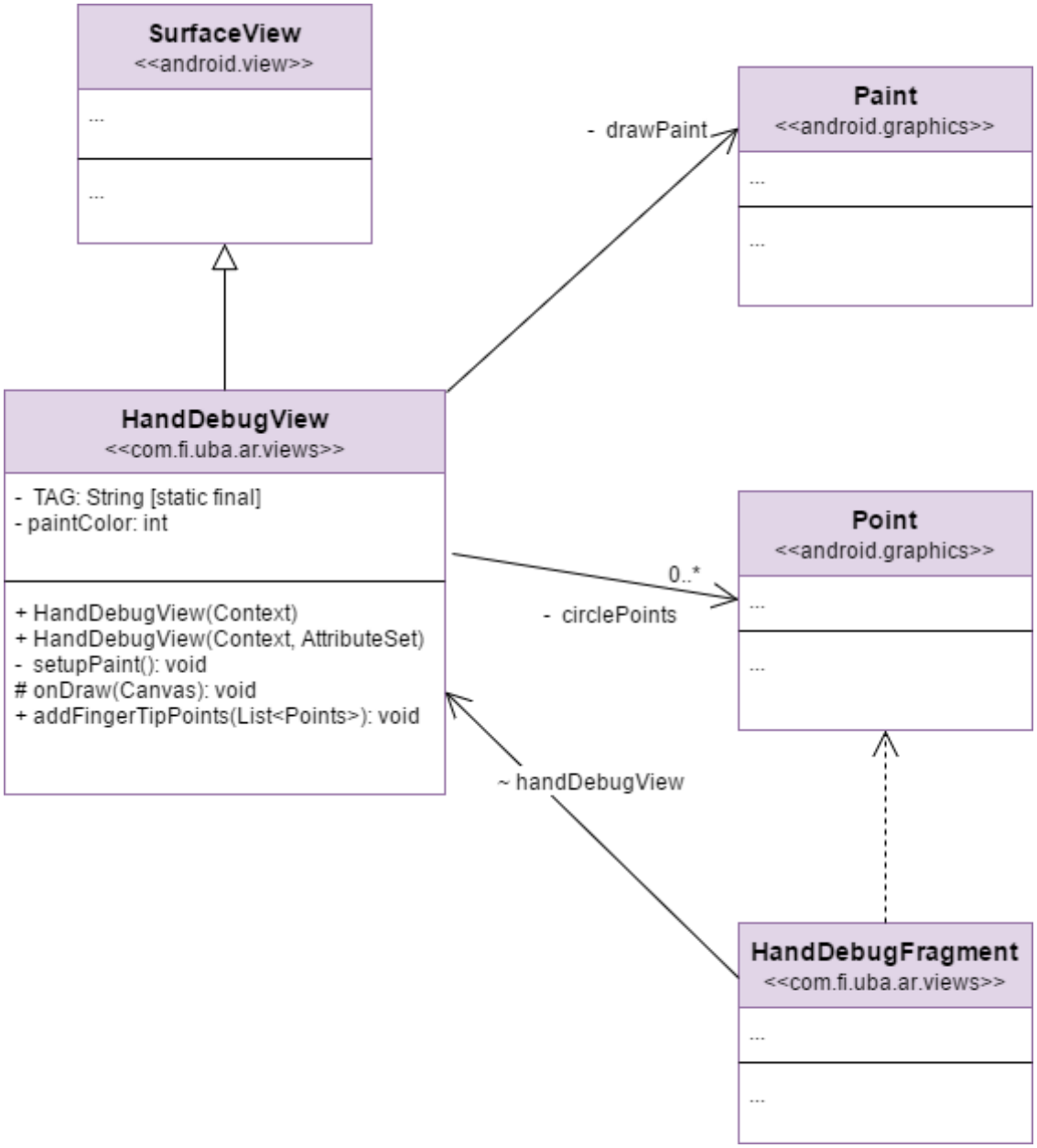

FIGURA 18 – CLASE HANDDEBUGVIEW.

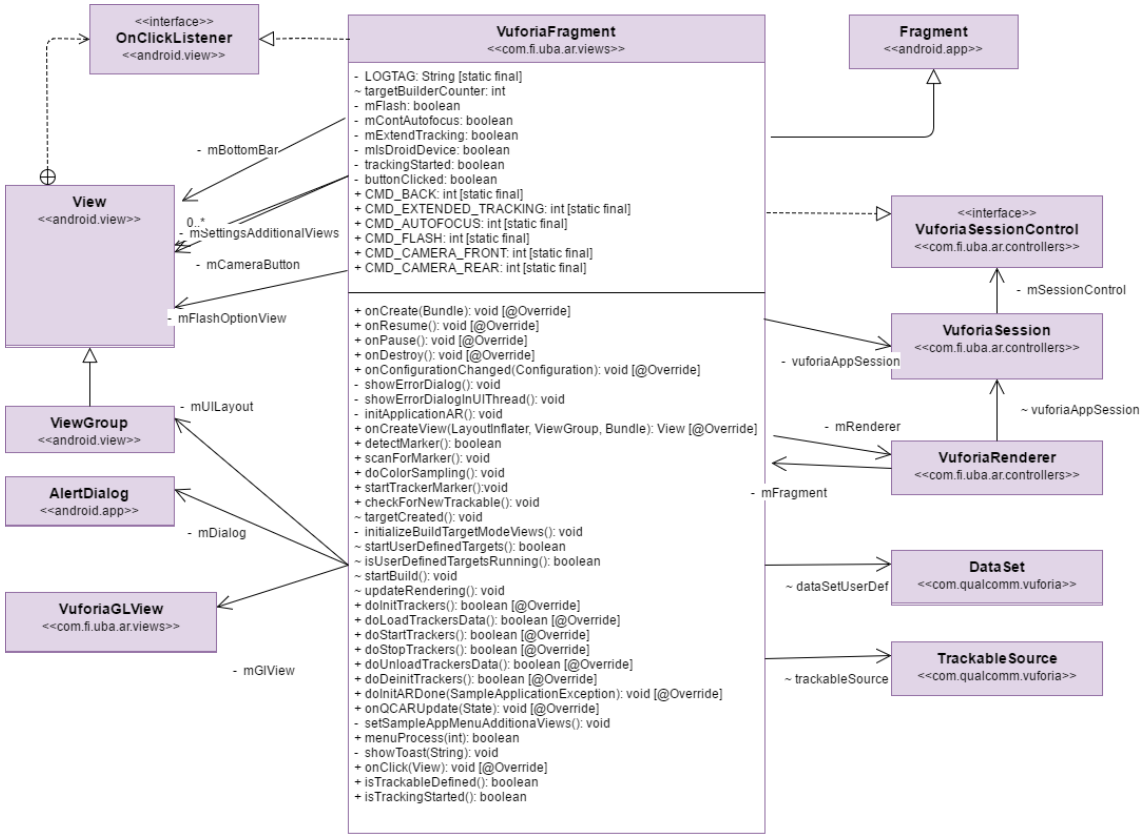

FIGURA 19 – CLASE VUFORIAFRAGMENT.

CLASE VUFORIARENDERER

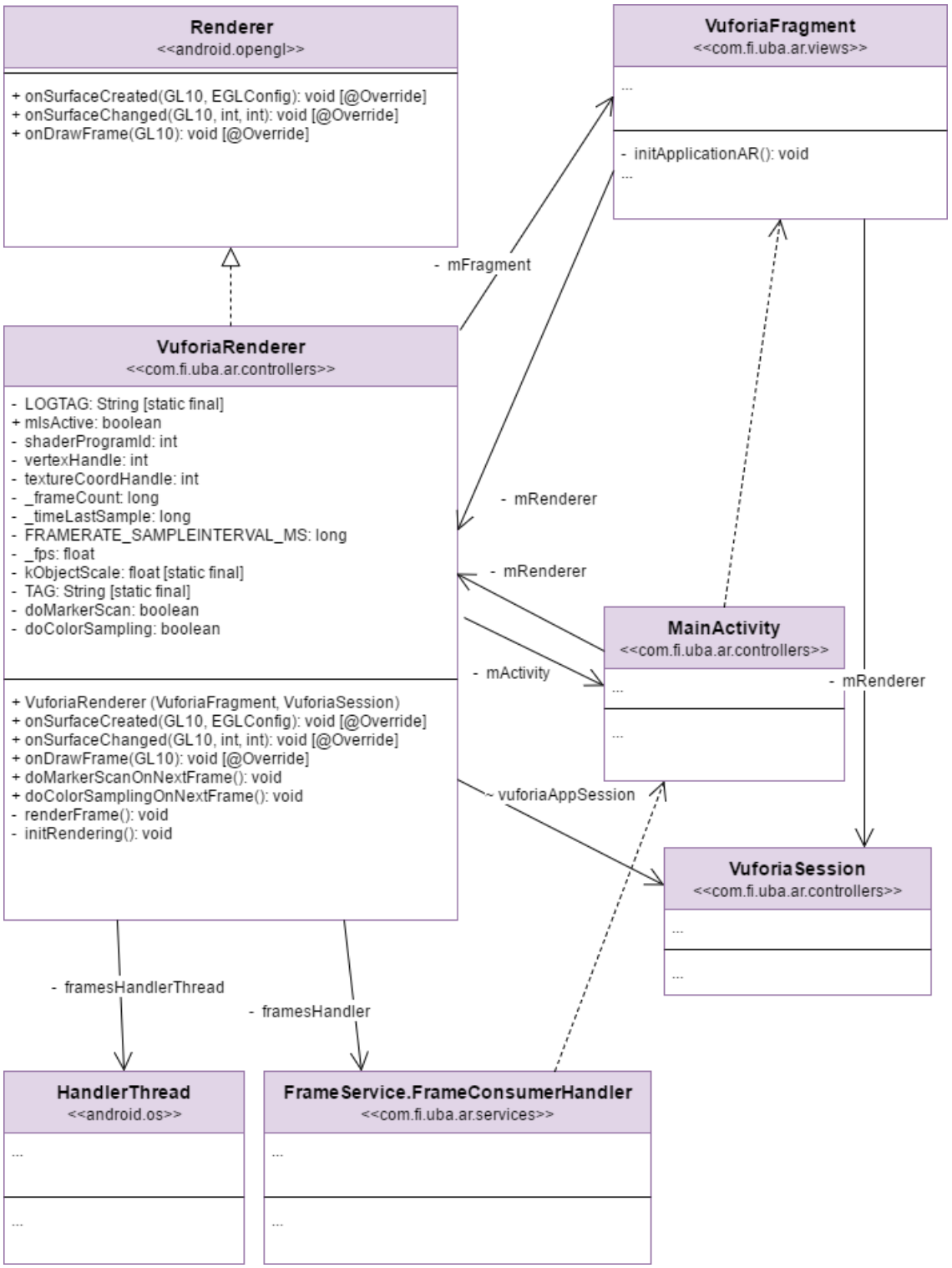

FIGURA 20 – CLASE VUFORIARENDERER.

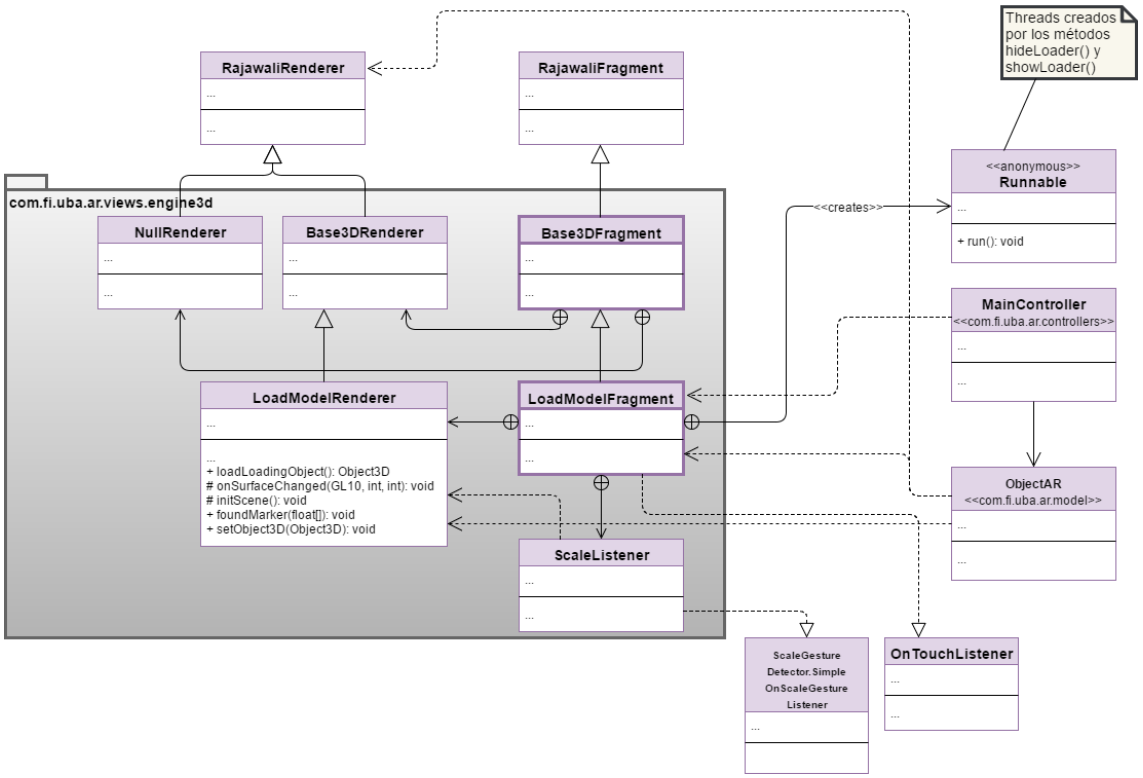

FIGURA 21 – CLASES DEL PAQUETE ENGINE3D.

CLASES DEL PAQUETE REST: RESTCLIENT Y DOWNLOAD3DMODELASYNCTASK

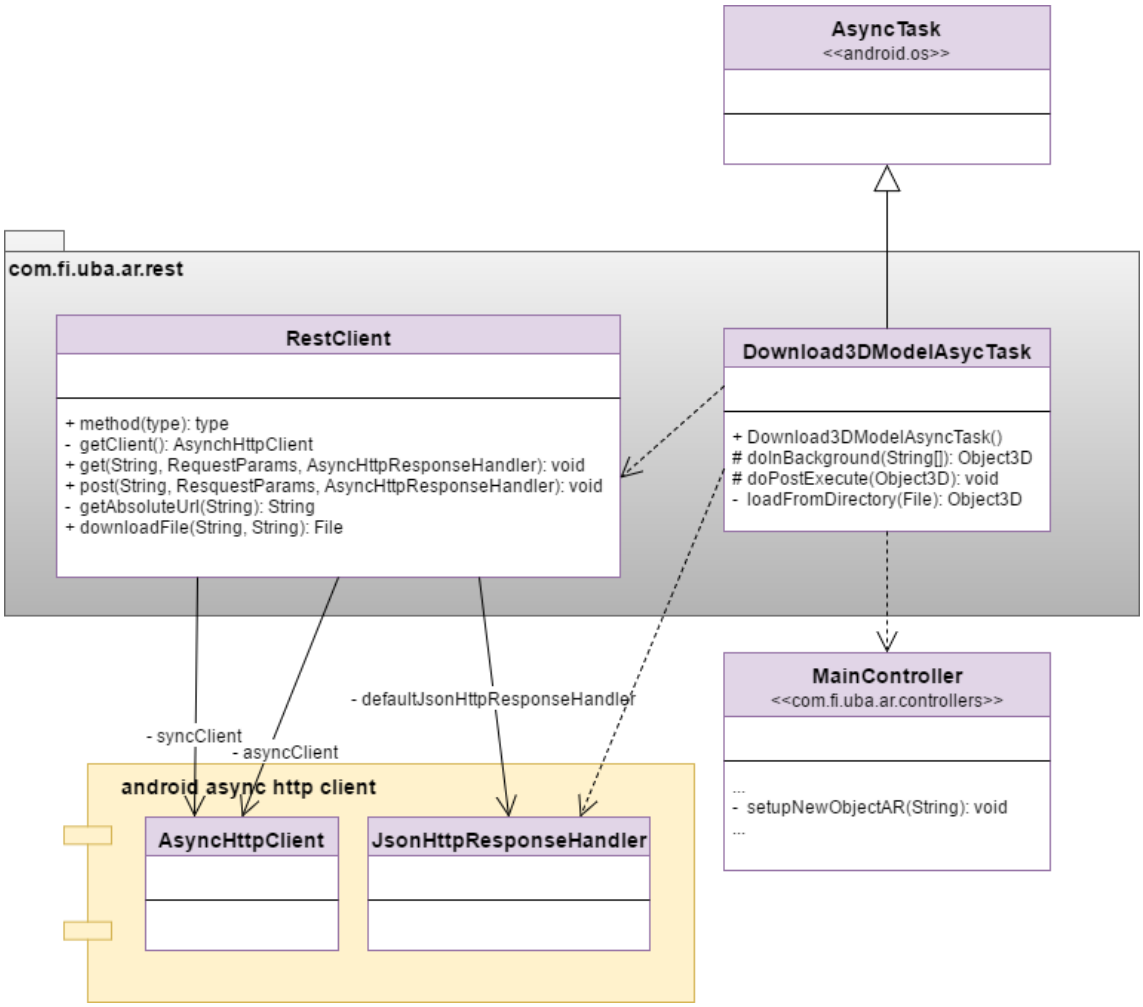

FIGURA 22 – CLASES DEL PAQUETE REST.

Es necesario señalar que estos diagramas representan a la Aplicación Cliente de FIUBAAR. Esta es la que representa el mayor esfuerzo de desarrollo, y por esta razón se muestra con todo detalle. Para entender la Aplicación FIUBAAR véase el diagrama de despliegue, en su sección específica.

SECUENCIA: ESCANEO DE UN MARCADOR

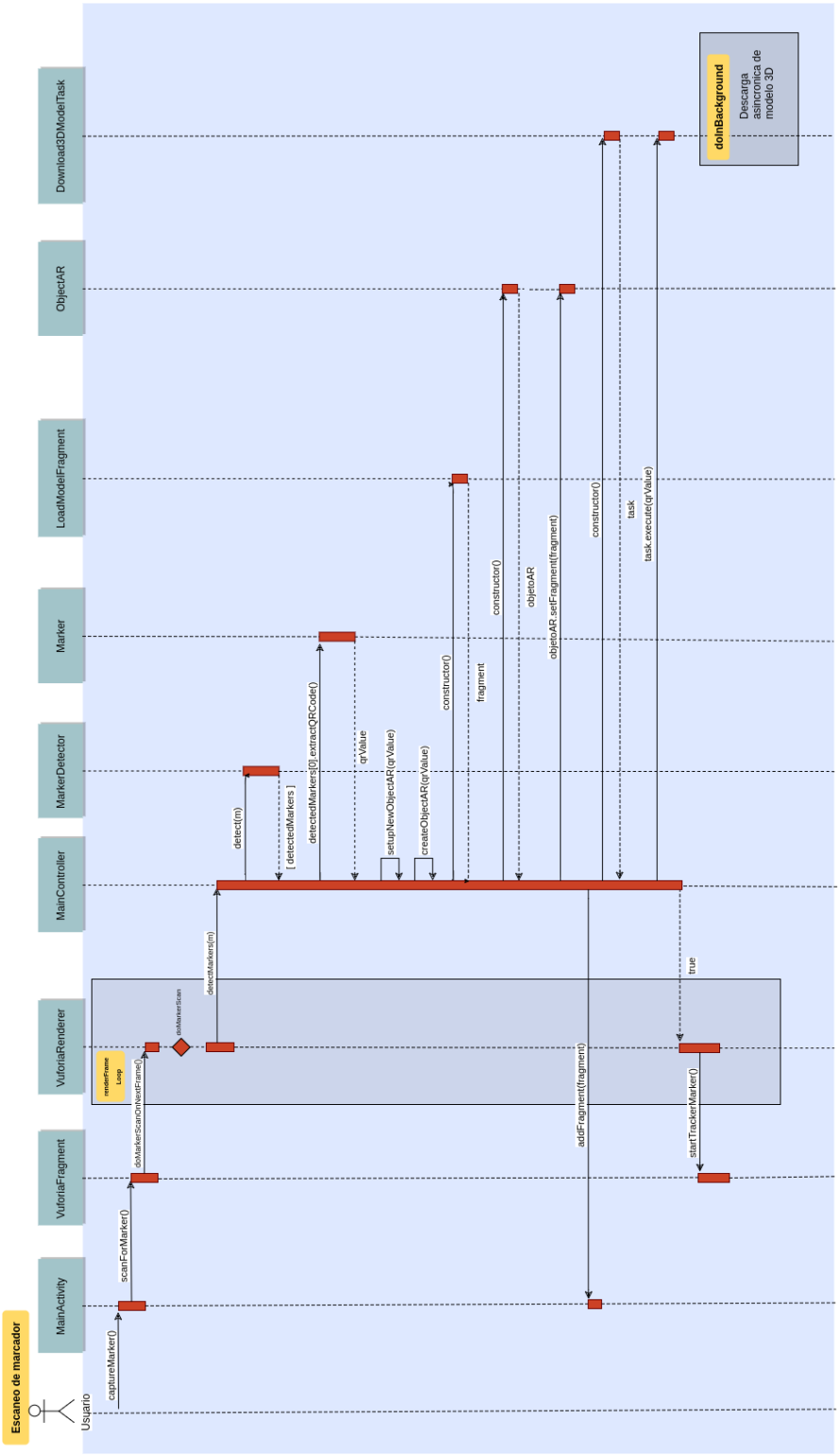

FIGURA 23 – DIAGRAMA DE SECUENCIA: ESCANEO DE UN MARCADOR.

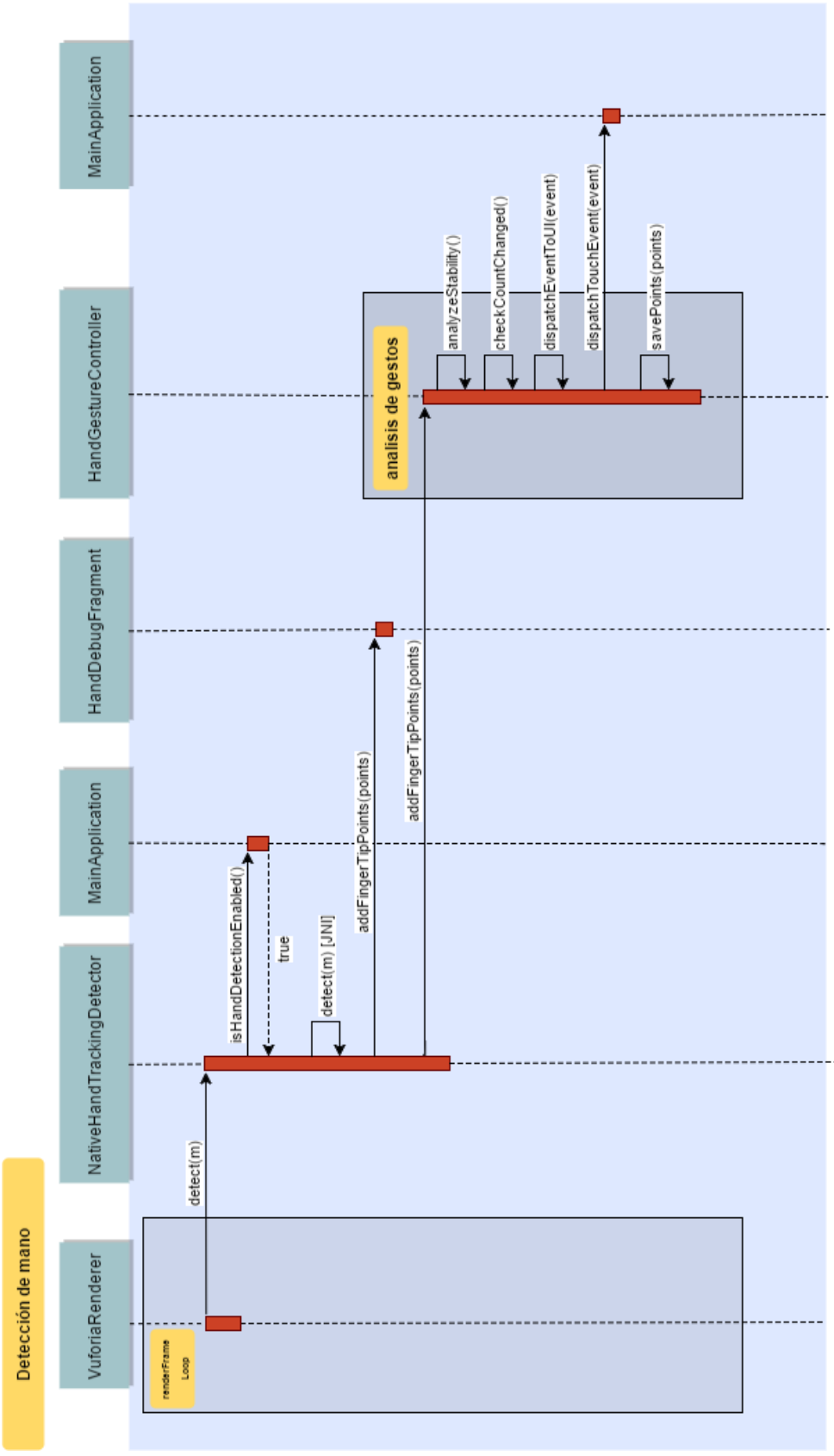

FIGURA 24 – DIAGRAMA DE SECUENCIA: DETECCIÓN DE MANO.

DIAGRAMA DE DESPLIEGUE

La aplicación FIUBAAR Cliente es la única de las desarrolladas en el presente proyecto que el usuario final conocerá, y deberá descargarla desde la página oficial del proyecto. Para que funcione, deberá además descargar en el dispositivo móvil, la aplicación OpenCV Manager, desde Google Play. Ambas aplicaciones ejecutarán en el dispositivo móvil del usuario. Sin embargo, el proyecto implicó la construcción de otra aplicación, la FIUBAAR Servidor, que es aquella desde la cual la aplicación cliente descargará los modelos destinados a "aumentar la realidad", dependiendo del identificador reconocido en el marcador.

La aplicación servidor, ejecuta en un equipo remoto, y la comunicación entre ambas -siempre iniciada por el cliente- se realizará mediante servicios REST sobre HTTP. Se hizo uso del Framework Play para implementar esta aplicación.

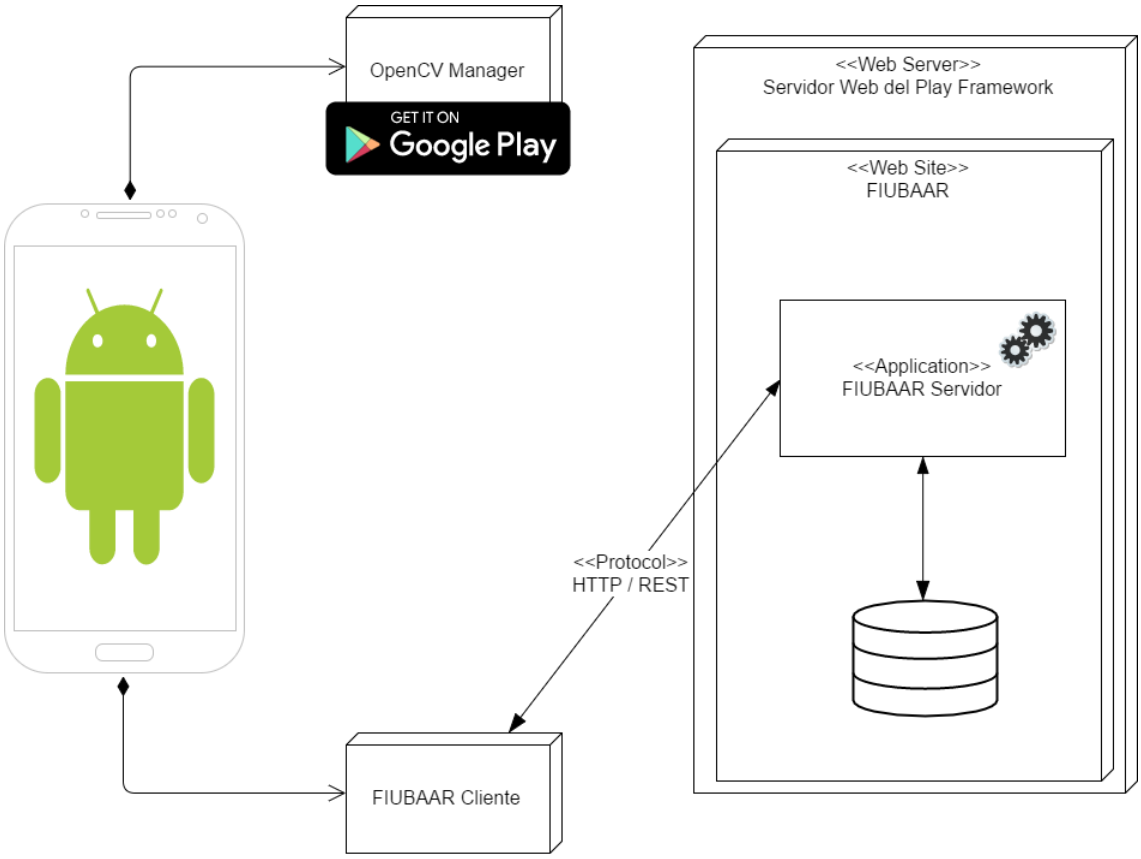

FIGURA 25 – DIAGRAMA DE DESPLIEGUE DE LAS APLICACIONES FIUBAAR CLIENTE Y FIUBAAR SERVIDOR.

Para la interacción con el usuario se ha recurrido a la utilización de “fragments”, los cuales representan vistas que se superponen entre sí. De este modo, un fragment mostrará la imagen tal cual es registrada por la cámara (VuforiaFragment), en otro cobrarán vida los objetos 3D que aumentarán la realidad (LoadModelFragment). De esta forma hemos mantenido aislados entre sí, dos conceptos que forman parte de la aplicación pero que no deben entrelazarse en la vista. Algo así como “separación de incumbencias” (*separation of concerns*) aplicado a reducir el acoplamiento entre componentes de la vista.

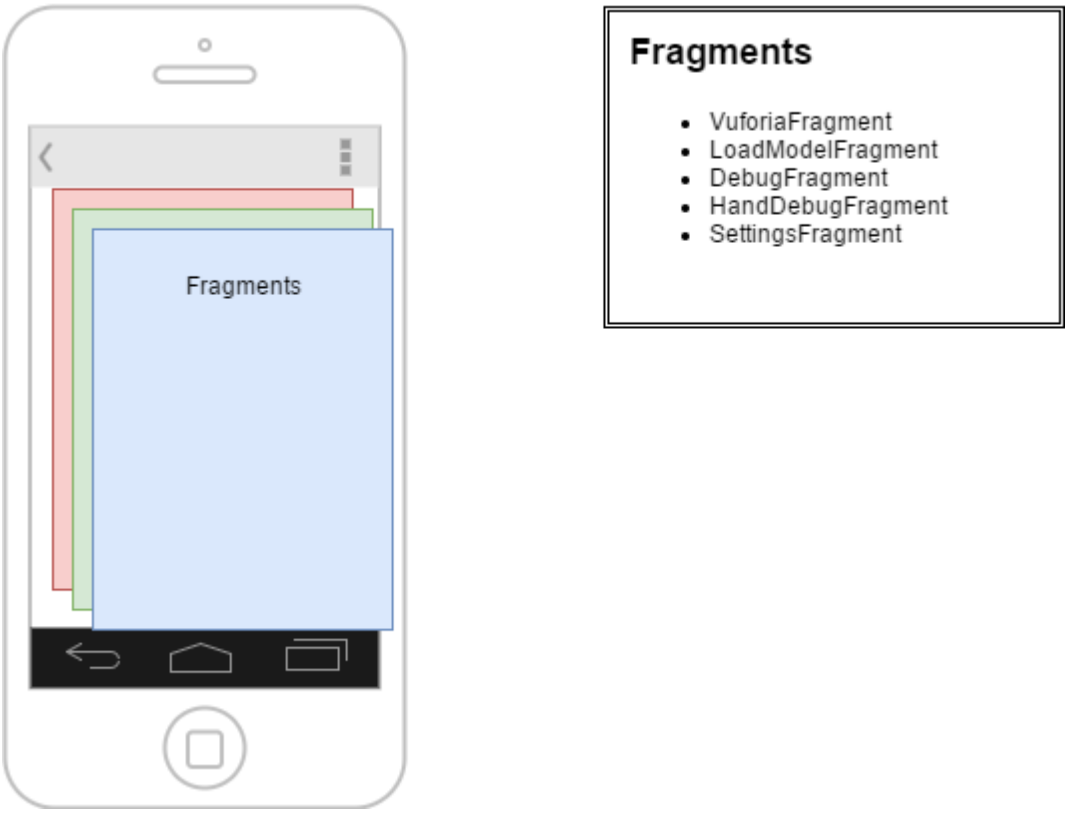

FIGURA 26 – REPRESENTACIÓN DEL USO DE FRAGMENTOS DE ANDROID EN LA CONSTRUCCIÓN DE UNA INTERFAZ DE USUARIO.

La aplicación servidor es el único componente del sistema que cuenta con una base de datos para mantener información. Consecuentemente, se definió para ella el modelo de datos que a continuación se describe.

El modelo de datos en esta instancia está enfocado a reunir los datos de cada usuario y relacionarlos con los múltiples proyectos sobre los que puede accionar como propietario.

Los modelos 3D a visualizarse no forman parte de la base de datos propiamente dicha, sino que son archivos ZIP separados, los cuales son referenciados por la base de datos a partir de su nombre en el sistema de archivos (*file system*).

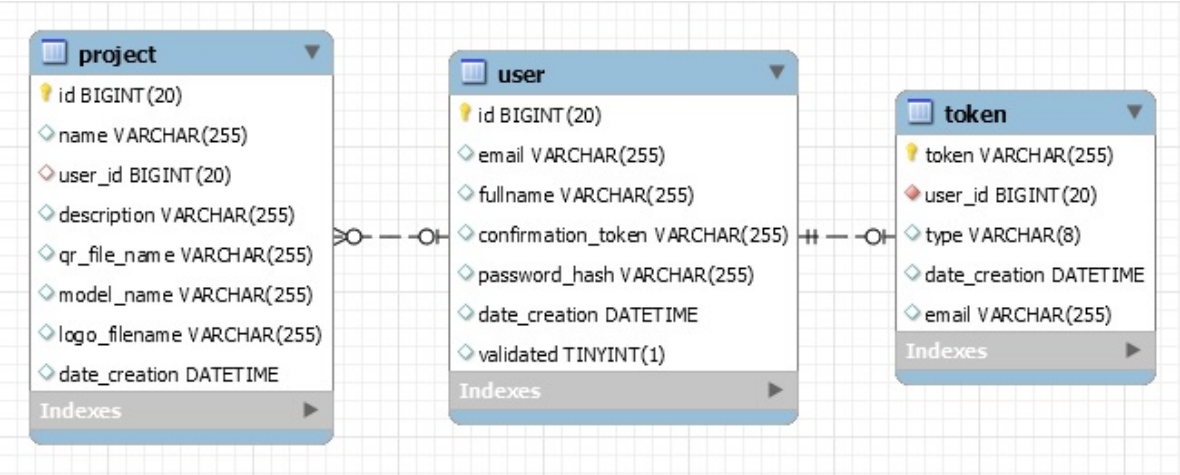

FIGURA 27 – PERSISTENCIA EN FIUBAAR SERVIDOR.

TABLA 4 – ESPECIFICACIÓN DE LA TABLA "USER".

Tabla	***user***		
Guarda la lista de usuarios del sistema, habilitados a operar o a la espera de validación.			
Columnas			
Id	PK	bigint(20)	Id del usuario.
Email	Único	varchar(255)	e-mail del usuario.
fullname		varchar(255)	Nombre completo del usuario.
confirmation_token		varchar(255)	Token enviado a su dirección de correo electrónico -integrado en un hipervínculo- para verificar la autenticidad de su e-mail.
password_hash		varchar(255)	Hash obtenido a partir del password suministrado por el usuario.
date_creation		datetime	Fecha y hora de la validación del usuario.
validated		tinyint(1)	En '1' si el usuario validó su dirección de correo electrónico.

Relaciones				
Nombre de la relación	Tabla FK	Tabla PK	Join	Card.
fk_project_user_1	project	user	user_id = id	0..*
fk_token_user_1	token	user	user_id = id	0..1

TABLA 5 – ESPECIFICACIÓN DE LA TABLA "PROJECT".

Tabla	***project***			
Guarda la información de todos los proyectos -con sus características o referencias a sus componentes- para cada usuario.				
Columnas				
Id	PK	bigint(20)	Id del proyecto.	
Name	Único	varchar(255)	Nombre del proyecto.	
user_id	FK	bigint(20)	FK con tabla *<<user>>*.	
description		varchar(255)	Descripción del proyecto.	
qr_file_name		varchar(255)	Ruta del código QR creado para el proyecto.	
model_name		varchar(255)	Ruta al archivo .zip que contiene el modelo 3D a recrear en la aplicación móvil.	
logo_filename		varchar(255)	Ruta al logo del proyecto.	
date_creation		datetime	Fecha y hora de la creación del proyecto.	
Relaciones				
Nombre de la relación	Tabla FK	Tabla PK	Join	Card.
fk_user_project_1	user	project	user_id = id	1

TABLA 6 – ESPECIFICACIÓN DE LA TABLA "TOKEN".

Tabla	***token***			
Contiene información necesaria para la validación de la dirección de e-mail del usuario.				
Columnas				
token	PK	varchar(255)	Id del token.	
user_id	FK	bigint(20)	FK con tabla <<user>>.	
type		varchar(8)	Sin uso.	
date_creation		datetime	Fecha y hora de creación del token.	
email		varchar(255)	E-mail al que fue enviado.	
Relaciones				
Nombre de la relación	Tabla FK	Tabla PK	Join	Card.
fk_user_token_1	user	token	user_id = id	1

PRUEBAS DURANTE EL PROYECTO

A cada User Story se le asigna un puntaje que se incrementa a medida que la complejidad de la funcionalidad a implementar aumenta. Las User Stories pueden adoptar como puntaje los siguientes valores: 1, 2, 3, 5, 8, 13, 21. En función de lo dicho, una User Story implementando una funcionalidad con la menor complejidad posible adoptará el valor 1 como puntaje, mientras que aquella que implemente una funcionalidad con la máxima complejidad, tomará el valor 21.

Los roles involucrados son: "usuario" y "usuario registrado". El primero representa a un usuario anónimo -que puede serlo tanto de FIUBAAR Cliente como de FIUBAAR Servidor- mientras que el segundo es un usuario que se registró en la aplicación FIUBAAR Servidor, seguramente con la intención de poner un proyecto (representado por un modelo 3D) a disposición de los usuarios de la aplicación FIUBAAR Cliente.

SPRINT N°1

US1.1 INSTALACIÓN DE FIUBAAR CLIENTE

Story Points: 3

Como usuario,

quiero obtener el instalador de FIUBAAR Cliente como archivo APK,

de modo de instalar la aplicación en mi dispositivo móvil Android.

CRITERIO DE ACEPTACIÓN US1.1

Dado un usuario que obtiene el archivo FiubaAR.apk,

cuando selecciona el archivo en su dispositivo móvil el mismo permite instalarlo,

entonces un nuevo ícono de FiubaAR aparece en la lista de aplicaciones disponibles.

Estado: aprobado.

US1.2 EJECUCIÓN DE FIUBAAR CLIENTE

Story Points: 3

Como usuario,

quiero ejecutar la aplicación FIUBAAR Cliente antes instalada en el dispositivo,

de modo de interactuar con la misma.

CRITERIO DE ACEPTACIÓN US1.2

Dado un usuario que instaló la aplicación FIUBAAR Cliente

cuando presiona el ícono en la lista de aplicaciones del dispositivo

entonces se ejecuta la aplicación y muestra una pantalla de bienvenida

Estado: aprobado.

US1.3 FIUBAAR CLIENTE BÁSICA HACE USO DE STACK DE FRAGMENTS

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente haga uso de un stack de Fragments,

de modo de poder navegar en diferentes vistas hacia adelante y atrás.

CRITERIO DE ACEPTACIÓN US1.3

Dado un usuario que ejecuta la versión básica de FIUBAAR Cliente

cuando presiona los botones "atrás" o "adelante",

entonces ve nuevas pantallas (Fragments).

(Nota: los botones desaparecerán en futuras versiones y esto permite simplemente contar con una implementación de base para poder luego superponer vistas de Modelos 3D por sobre la imagen de video)

Estado: aprobado.

US1.4 OPENCV INTEGRADO

Story Points: 5

Como usuario,

quiero que FIUBAAR Cliente integre OpenCV en un dispositivo móvil con SO Android con cámara,

de modo de "capturar video de la cámara, capturar imágenes y aplicar filtros de colores".

CRITERIO DE ACEPTACIÓN US1.4

Dado un usuario en uso de la aplicación,

cuando el usuario presiona el botón "Gray Scale" dentro del menú "filtros",

entonces el video capturado por la cámara pasa a verse en escala de grises

y

Dado un usuario en uso de la aplicación,

cuando el usuario presiona el botón "Canny" dentro del menú "filtros"

entonces del video capturado por la cámara solo se ve el contorno de los objetos.

Estado: aprobado.

US1.5 ENGINE3D INTEGRADO

Story Points: 5

Como usuario,

quiero que FIUBAAR Cliente integre un Engine3D en un dispositivo móvil con pantalla,

de modo de poder mostrar una animación 3D en pantalla.

CRITERIO DE ACEPTACIÓN US1.5

Dado un usuario,

cuando el usuario presiona el botón "Ver modelo 3D",

entonces una animación 3D predefinida aparece en pantalla rotando.

Nota: esta funcionalidad es temporal, para verificar el correcto funcionamiento (en futuros sprints, la funcionalidad desaparece).

Estado: aprobado.

SPRINT N°2

US2.1 RECONOCIMIENTO PERPENDICULAR DE MARCADOR

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente reconozca un marcador cuando la cámara lo apunta sobre la recta normal a ambos (zenit) y con su contorno alineado con el cuadro de la imagen,

de modo de poder reconocer su presencia en el campo de visión.

CRITERIO DE ACEPTACIÓN US2.1

Dado un usuario visualizando un escenario a través de la aplicación,

cuando el usuario encuentra un marcador exactamente frente a la cámara y en un plano paralelo al propio,

entonces la aplicación reconoce al marcador y lo recuadra en color verde.

Estado: aprobado.

US2.2 RECONOCIMIENTO PERPENDICULAR DE MARCADOR ROTADO

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente reconozca un marcador cuando la cámara lo apunta sobre recta normal a ambos (zenit),

de modo de poder reconocer su presencia en el campo de visión.

CRITERIO DE ACEPTACIÓN US2.2

Dado un usuario visualizando un escenario a través de la aplicación,

cuando el usuario encuentra un marcador exactamente frente a la cámara y en un plano paralelo al propio, pero con distintas rotaciones,

entonces la aplicación reconoce al marcador y lo recuadra en color verde.

Estado: aprobado.

US2.3 TRACKING DE MARCADOR

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente reconozca un marcador (en plano paralelo al de la cámara) en cada cuadro de video al mover ligeramente la cámara sobre su mismo plano

de modo de poder hacer un seguimiento del mismo.

CRITERIO DE ACEPTACIÓN US2.3

Dado un usuario visualizando un escenario a través de la aplicación,

cuando el usuario encuentra un marcador exactamente frente a la cámara y en un plano paralelo al propio, y desplaza ligeramente la cámara hacia los lados,

entonces la aplicación reconoce al marcador y lo recuadra en color verde en cada cuadro de video.

Estado: aprobado.

US2.4 RECONOCIMIENTO DE MARCADOR MULTIANGULAR

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente reconozca un marcador en cada cuadro de video al cambiar en ángulo entre el plano de la cámara y el del marcador,

de modo de poder reconocer al marcador en cualquier ángulo (entre -45° y +45°).

CRITERIO DE ACEPTACIÓN US2.4

Dado un usuario visualizando un escenario a través de la aplicación,

cuando el usuario cambia severamente el ángulo relativo (sin sacar de escena al marcador) entre los -45° y los +45°,

entonces FIUBAAR Cliente reconoce al marcador y lo recuadra en color verde en cada cuadro de video.

y

Dado un usuario visualizando un escenario a través de la aplicación,

cuando el usuario desplaza la cámara (sin sacar de escena al marcador),

entonces FIUBAAR Cliente recuadra al marcador en color verde en cada nueva posición que adopta.

Estado: aprobado.

SPRINT N°3

US3.1 FIUBAAR SERVIDOR BÁSICO CON REGISTRO DE USUARIOS

Story Points: 8

Como usuario,

quiero que FIUBAAR Servidor me permita registrarme como usuario registrado

de modo de poder ingresar al sitio web de FIUBAAR con contraseña y ver mis proyectos.

CRITERIO DE ACEPTACIÓN US3.1

Dado un usuario,

cuando el usuario ingresa a la aplicación FIUBAAR servidor y solicita registrarse

entonces FIUBAAR Servidor le solicita nombre y contraseña, y le envía un mail de confirmación

y

dado un usuario que llenó la solicitud de registro

cuando responde al mail de confirmación

entonces el usuario se convierte en usuario registrado.

Estado: aprobado.

US3.2 FIUBAAR SERVIDOR BÁSICO CON RESPUESTA

Story Points: 8

Como usuario,

quiero que FIUBAAR Servidor responda las solicitudes recibidas

de modo de verificar que cumple con la funcionalidad básica.

CRITERIO DE ACEPTACIÓN US3.2

Dado un usuario,

cuando el usuario solicita un recurso a FIUBAAR Servidor mediante un cliente REST

entonces FIUBAAR responde con el recurso por defecto o con un aviso de error.

Estado: aprobado.

US3.3 FIUBAAR CLIENTE LEE QR

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente lea el código QR dentro del marcador

de modo de poder recuperar el proyecto asociado desde FIUBAAR Servidor.

CRITERIO DE ACEPTACIÓN US3.3

Dado un usuario utilizando FIUBAAR Cliente,

cuando el usuario apunta con la cámara al marcador con código QR válido,

entonces FIUBAAR Cliente muestra el texto leído del código QR.

Estado: aprobado.

US3.4 FIUBAAR CLIENTE EDITA Y GUARDA CONFIGURACIÓN

Story Points: 5

Como usuario,

quiero que FIUBAAR Cliente permita editar valores de configuración y los guarde entre ejecuciones

de modo de poder apuntar a diferentes direcciones URL de FIUBAAR Servidor.

CRITERIO DE ACEPTACIÓN US3.4

Dado un usuario,

cuando el usuario presiona el menú “Settings” en FIUBAAR Cliente

entonces FIUBAAR Cliente presenta un listado de valores de configuración posibles de editar.

Estado: aprobado.

US3.5 FIUBAAR CLIENTE SE COMUNICA CON FIUBAAR SERVIDOR

Story Points: 5

Como usuario,

quiero que FIUBAAR Servidor responda las solicitudes de FIUBAAR Cliente

de modo de verificar la comunicación entre ambas aplicaciones.

CRITERIO DE ACEPTACIÓN US3.5

Dado un usuario,

cuando el usuario presiona el botón “Solicitar recurso” en FIUBAAR Cliente

entonces FIUBAAR Cliente responde “Ok” si FIUBAAR Servidor contestó la solicitud.

Estado: aprobado.

SPRINT N°4

US4.1 FIUBAAR SERVIDOR PERMITE CREAR PROYECTOS

Story Points: 8

Como usuario registrado autenticado en el sistema,

quiero que FIUBAAR me permita editar mi lista de proyectos

de modo de poder cargar uno nuevo.

CRITERIO DE ACEPTACIÓN US4.1

Dado un usuario registrado en FIUBAAR Servidor,

cuando el usuario presiona "Crear nuevo proyecto" en FIUBAAR Servidor

entonces FIUBAAR Servidor presenta un formulario con los campos y botones necesarios para (1) dar nombre al proyecto, (2) subir el modelo 3D y (3) subir el ícono del proyecto. Luego de presionar "Aceptar", se mostrará la lista de proyectos en la cual aparecerá el proyecto recién cargado.

Estado: aprobado.

US4.2 FIUBAAR SERVIDOR PERMITE ELIMINAR PROYECTOS

Story Points: 5

Como usuario registrado autenticado en el sistema,

quiero que FIUBAAR me permita editar mi lista de proyectos

de modo de poder borrar un proyecto existente.

CRITERIO DE ACEPTACIÓN US4.2

Dado un usuario registrado en FIUBAAR Servidor,

cuando el usuario presiona botón "Eliminar" junto al nombre del proyecto,

entonces el proyecto en cuestión desaparece de la lista de proyectos y también de la BD.

Estado: aprobado.

US4.3 FIUBAAR CLIENTE UBICA UN MODELO 3D SOBRE EL MARCADOR

Story Points: 13

Como usuario,

quiero que FIUBAAR muestre un modelo 3D sobre el marcador con código QR

de modo de permitir al usuario su visualización.

CRITERIO DE ACEPTACIÓN US4.3

Dado un usuario,

cuando el usuario apunta a un marcador con código QR legible,

entonces FIUBAAR Cliente muestra un modelo 3D del texto "Loading…"

Estado: aprobado.

US4.4 FIUBAAR CLIENTE LEE QR Y RECIBE MODELO DEL PROYECTO

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente represente sobre el marcador el modelo 3D correspondiente al marcador leído.

de modo de verificar que FIUBAAR Cliente recupera el modelo 3D correcto desde FIUBAAR Servidor y puede representarlo correctamente.

CRITERIO DE ACEPTACIÓN US4.4

Dado un usuario de FIUBAAR Cliente,

cuando el usuario apunta su cámara al marcador y éste contiene un QR correspondiente a un proyecto válido,

entonces FIUBAAR Cliente se conecta a FIUBAAR Servidor con la solicitud correspondiente y recupera de este el modelo 3D, representándolo a continuación sobre el marcador.

Estado: aprobado.

SPRINT N°5

US5.1 FIUBAAR CLIENTE RECONOCE UNA MANO EN LA ESCENA

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente reconozca la presencia de una mano en la escena,

de modo de verificar que la reconoce.

CRITERIO DE ACEPTACIÓN US5.1

Dado un usuario de FIUBAAR Cliente,

cuando el usuario muestra su mano a la cámara y presiona el botón “sampling”

entonces FIUBAAR Cliente muestra claramente en pantalla que reconoce su contorno.

Estado: aprobado.

US5.2 FIUBAAR CLIENTE PUEDE SEGUIR A LA MANO EN LA ESCENA

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente siga la ubicación de una mano en los sucesivos cuadros de video

de modo de verificar que pueda reconocer gestos que impliquen un desplazamiento de la mano.

CRITERIO DE ACEPTACIÓN US5.2

Dado un usuario de FIUBAAR Cliente,

cuando el usuario muestra su mano a la cámara y presiona el botón “sampling”

entonces FIUBAAR Cliente confirma que el sampling se ejecutó con éxito

y

dado un usuario de FIUBAAR Cliente,

cuando el usuario mueve se mano frente a la cámara,

entonces FIUBAAR Cliente muestra con claridad que sigue sus movimientos.

Estado: aprobado.

US5.3 FIUBAAR CLIENTE CUENTA LOS DEDOS DESPLEGADOS

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente indique cuántos dedos reconoce en la mano

de modo de verificar la exactitud y estabilidad del procedimiento.

CRITERIO DE ACEPTACIÓN US5.3

Dado un usuario de FIUBAAR Cliente,

cuando el usuario realiza la detección y muestra la mano con uno, dos y tres dedos desplegados

entonces FIUBAAR Cliente muestra un texto con la cantidad de dedos que reconoce.

Estado: aprobado.

SPRINT N°6

US6.1 FIUBAAR CLIENTE AGRANDA MODELO 3D CON CONTACTO

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente efectúe un zoom in o zoom out sobre el modelo 3D al tocar la pantalla del dispositivo móvil,

de modo de interactuar con el modelo 3D de manera dinámica.

CRITERIO DE ACEPTACIÓN US6.1

Dado un usuario de FIUBAAR Cliente,

cuando el usuario toca la pantalla del dispositivo con 2 dedos y los separa entre si

entonces FIUBAAR Cliente agranda al modelo 3D (zoom in).

y

Dado un usuario de FIUBAAR Cliente,

cuando el usuario toca la pantalla del dispositivo con 2 dedos y los acerca entre si

entonces FIUBAAR Cliente reduce al modelo 3D (zoom out)

Estado: aprobado.

US6.2 FIUBAAR CLIENTE GIRA MODELO 3D CON CONTACTO

Story Points: 8

Como usuario,

quiero que FIUBAAR Cliente rote el modelo 3D al tocar la pantalla del dispositivo móvil,

de modo de interactuar con el modelo 3D de manera dinámica.

CRITERIO DE ACEPTACIÓN US6.2

Dado un usuario de FIUBAAR Cliente,

cuando el usuario toca la pantalla del dispositivo con 2 o más dedos girando los mismos,

entonces FIUBAAR Cliente rota el modelo 3D tanto como el usuario gire su mano.

Estado: aprobado.

US6.3 FIUBAAR CLIENTE INTERACTÚA CON EL MODELO 3D SIN CONTACTO

Story Points: 13

Como usuario,

quiero que FIUBAAR Cliente rote el modelo 3D o haga un zoom in o zoom out en función de la cantidad de dedos reconocida (uno, dos o tres).

de modo de verificar la correcta interacción entre lo que se reconoce de la mano y la forma en que responde el modelo 3D.

CRITERIO DE ACEPTACIÓN US6.3

Dado un usuario de FIUBAAR Cliente,

cuando el usuario muestra un dedo a la cámara con un QR válido

entonces FIUBAAR Cliente pone el modelo 3D a rotar

y

Dado un usuario de FIUBAAR Cliente,

cuando el usuario muestra dos dedos a la cámara con un QR válido

entonces FIUBAAR Cliente agranda al modelo 3D.

y

Dado un usuario de FIUBAAR Cliente,

cuando el usuario muestra tres dedos a la cámara con un QR válido

entonces FIUBAAR Cliente reduce al modelo 3D.

Estado: aprobado.

SPRINT N°7

US7.1 FIUBAAR CLIENTE EVALÚA ESTABILIDAD EN RECONOCIMIENTO

Story Points: 13

Como usuario de FIUBAAR Cliente,

quiero que FIUBAAR Cliente analice la estabilidad al realizar reconocimiento de mano y dedos,

de modo de poder comenzar a realizar gestos para interactuar con el modelo 3D.

CRITERIO DE ACEPTACIÓN US7.1

Dado un usuario de FIUBAAR Cliente,

cuando coloca su mano frente a la cámara en una misma posición por más de 3 segundos,

entonces FIUBAAR Cliente evalúa la estabilidad de la detección e informa el resultado mediante un mensaje "toast"

Estado: aprobado.

US7.2 FIUBAAR CLIENTE GENERA EVENTOS "TOUCH"

Story Points: 13

Como usuario de FIUBAAR Cliente,

quiero que FIUBAAR Cliente genere eventos de toque de pantalla (*touch events*) en los lugares de la imagen en donde se detectaron visualmente dedos,

de modo de poder utilizar la detección de gestos para interactuar con modelos 3D.

CRITERIO DE ACEPTACIÓN US7.2

Dado un usuario de FIUBAAR Cliente,

cuando posiciona su mano frente a la cámara y dedos son detectados en diferentes posiciones,

entonces FIUBAAR Cliente informa mediante mensajes las coordenadas de la pantalla donde se produjo un evento de tocado y las mismas corresponden a los puntos en donde se detectan dedos.

Nota: estas notificaciones de coordenadas no formarán parte de la versión final entregable y se utilizarán solo en concepto de prueba necesaria como paso intermedio.

Estado: aprobado.

US7.3 FIUBAAR CLIENTE RECONOCE GESTO DE ZOOM

Story Points: 21

Como usuario de FIUBAAR Cliente,

quiero que FIUBAAR Cliente reconozca un gesto válido frente a la cámara para ampliar o reducir el modelo 3D.

de modo de tomar la acción correspondiente sobre el modelo 3D.

CRITERIO DE ACEPTACIÓN US7.3

Dado un usuario,

cuando el usuario efectúa un acercamiento entre dos dedos (pinzamiento o pellizco) frente a la cámara en el momento que visualiza un modelo 3D creado sobre el marcador,

entonces FIUBAAR Cliente reduce el tamaño del modelo 3D,

y

Dado un usuario,

cuando el usuario efectúa un alejamiento entre dos dedos (liberando el pinzamiento o pellizco) frente a la cámara en el momento que visualiza un modelo 3D creado sobre el marcador,

entonces FIUBAAR Cliente aumenta el tamaño del modelo 3D

Estado: aprobado.

US7.4 FIUBAAR CLIENTE RECONOCE GESTO DE ROTADO

Story Points: 21

Como usuario de FIUBAAR Cliente,

quiero que FIUBAAR Cliente reconozca un gesto válido frente a la cámara para rotar el modelo 3D

de modo de tomar la acción correspondiente sobre el modelo 3D.

CRITERIO DE ACEPTACIÓN US7.4

Dado un usuario,

cuando el usuario efectúa un giro de su mano frente a la cámara mostrando dos o más dedos,

entonces FIUBAAR Cliente pone a rotar al modelo 3D en el sentido del giro y con un ángulo proporcional a este.

Estado: aprobado.

INVESTIGACIONES DURANTE EL DESARROLLO

INVESTIGACIÓN DE MARKERS

Se conoce como marcador o "marker" en Realidad Aumentada a una imagen fija, probablemente con algún patrón simple, fácilmente distinguible y que la aplicación puede reconocer o detectar.

El tipo de marker comúnmente utilizado por la mayoría de las aplicaciones de realidad aumentada está basado en una imagen en blanco y negro para mayor facilidad de detección debido al contraste de colores y con una imagen geométrica simple que, generalmente, es un cuadrado.

También existen aplicaciones de realidad aumentada "sin marcadores" (*markerless*). Éstas no hacen uso de un marker, sino que realizan un seguimiento de cualquier tipo de imagen, sin necesidad de contar con características particulares o patrones fijos.

ESTADO DEL ARTE

En general, las aplicaciones de realidad aumentada poseen una cantidad limitada y fija de marcadores que son capaces de reconocer. Dichos marcadores no poseen información adicional y consisten simplemente en una imagen fija que la aplicación puede reconocer fácilmente. Ejemplo de esto son los markers utilizados por frameworks como ARToolkit[11]

Existen herramientas que permiten generar nuevos markers con los patrones que uno desee[12].

REQUERIMIENTOS FIUBAAR

En el caso de la aplicación FIUBAAR es necesario que aquel marker que se utilice pueda contener información reconocible en su patrón.

---

[11] Ver http://www.artoolworks.com/support/library/Creating_and_training_new_ARToolKit_markers

[12] Ejemplo de esto es la referenciada por http://www.arined.org/?page_id=6, disponible en http://flash.tarotaro.org/blog/2008/12/14/artoolkit-marker-generator-online-released/.

Es por esto que dicho marker deberá estar compuesto por dos elementos:

- Patrón fijo y reconocible, que llamaremos *marker contenedor* o *patrón exterior*
- Código 2D que pueda contener información, que llamaremos *marker núcleo* o *patrón interior*.

Existe hoy en día una aplicación de realidad aumentada que utiliza un marker con el concepto aquí expuesto. Esta aplicación es Zappar.[13] La misma, utiliza un marker llamado Zapcode[14].

Se puede apreciar que en el caso del Zapcode, los componentes del marker se encuentran invertidos ya que el patrón interior es fijo y es el patrón exterior el que cuenta con la información adicional dinámica. Su página explica claramente el funcionamiento[15] y allí mismo presentan una comparación con los códigos QR, explicando porque los zapcodes son -a su criterio- mejores[16].

Para poder definir la forma más conveniente de componer el marker completo que utilizará la aplicación FIUBAAR deberemos analizar los diferentes tipos de códigos de barras 2D[17].

ELECCIÓN DEL MARKER NÚCLEO

Es conocido que el código QR es uno de los más difundidos y es comúnmente reconocido por el público en general. Esto lo convierte en potencialmente el tipo de código más conveniente para la implementación, ya que podría ser fácilmente incluido en un patrón cuadrado de alto contraste tales como los que utiliza ARToolkit y además facilitaría el desarrollo el hecho de que existen ya librerías capaces de proveer su reconocimiento, como así también el generarlos.

TRACKING DEL MARKER

La aplicación deberá poder realizar tracking del marker para determinar la posición y luego proyectar allí un objeto 3D. Es importante notar que el reconocimiento del marker núcleo (código QR) es solo necesario realizarlo una única vez para obtener la información necesaria que indica desde donde obtener el modelo de objeto 3D.

---

[13] Zappar está disponible en http://www.zappar.com/.
[14] Ver https://zapcode.it/ para másdetalles.
[15] Ver https://zapcode.it/how-zapcodes-work/.
[16] Ver https://zapcode.it/zapcodes-vs-qr/.
[17] Ver http://en.wikipedia.org/wiki/Barcode y http://en.wikipedia.org/wiki/Mobile_tagging.

Esto entonces implica que solo será necesario realizar tracking del marker contenedor (patrón fijo exterior) lo que simplificaría notoriamente el algoritmo utilizado.

MARKER SELECCIONADO

Tal como se ha explicado anteriormente el marker completo está compuesto por un contenedor y un núcleo.

No existen medidas estrictamente necesarias para el contenedor, y la única restricción es que posea una forma rectangular con un borde de ancho considerable de color negro. Cabe aclarar que no hace falta que ambos lados posean la misma medida para convertirlo en un cuadrado perfecto.

El núcleo será un código QR que debe estar separado por al menos algunos milímetros del borde del borde contenedor para facilitar la lectura.

A continuación, se presenta un marker completo a modo de ejemplo:

FiubaAR

FIGURA 28 – MARKER DE EJEMPLO DE FIUBAAR.

DOCUMENTACIÓN ADICIONAL ÚTIL

Durante la investigación hemos encontrado abundante información y herramientas que, si bien no se han incluido en el desarrollo central de esta sección, merece una mención especial debido a su interés.

Podemos mencionar a la librería QRLive[18], como también a ZXing (Zebra Crossing)[19], librería esta última que reconoce varios tipos de código y posee una implementación para Android.

La tesis de Tamino Hartmann[20] de la Universidad de Ulm en Alemania, en donde se implementa detección de markers que -si bien no son de tipo QR- considera el reconocimiento de marcadores. El código de su implementación está disponible en Github[21].

En Wikipedia también pueden encontrarse definiciones que son centrales para comprender y ampliar conceptos sobre realidad aumentada[22].

Comparación entre AR con marcadores y sin estos (*markerless*). Además del QR, menciona otros formatos utilizados[23]

Listas de librerías comúnmente utilizadas para construir aplicaciones en Android[24].

Así mismo, una significativa cantidad de dudas que van surgiendo, tanto durante la investigación como también durante los primeros intentos de implementación, encuentran sus respuestas en igual numerosos artículos dispersos por Internet[25].

Entre las herramientas de utilidad, se pueden encontrar aquellas que permiten generar dinámicamente una imagen QR con Google charts[26] y otras que facilitan el uso de otras librerías de lectura de códigos de barras[27].

---

[18] En https://github.com/blopker/QRlive, antes en https://github.com/blopker/QRlive.
[19] Disponible en https://github.com/zxing/zxing.
[20] Disponible en http://dbis.eprints.uni-ulm.de/986/1/BA_Hartmann_13.pdf
[21] Código en https://github.com/xamino/bachelor_framework_git.
[22] Como ejemplo, http://en.wikipedia.org/wiki/Fiduciary_marker.
[23] Ver http://researchguides.dartmouth.edu/content.php?pid=227212&sid=1891183
[24] Ver https://goo.gl/kSOCNk
[25] http://stackoverflow.com/questions/22743213/detect-the-size-of-a-qr-code-in-python-using-opencv-and-zbar, http://stackoverflow.com/questions/22145084/scanning-qr-codes-using-opencv-and-zxing-for-android, http://karanbalkar.com/2013/12/tutorial-65-implement-barcode-scanner-using-zxing-in-android/, http://dsynflo.blogspot.com.ar/2010/03/2d-barcodes-form-data-matrix-to-qr-code.html, http://blog.ayoungprogrammer.com/2013/07/tutorial-scanning-barcodes-qr-codes.html, http://blog.ayoungprogrammer.com/2014/04/real-time-qr-code-bar-code-detection.html, http://www.blackdogfoundry.com/blog/zbar-bar-code-qr-code-reader-android/.
[26] http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=|1&chl=TEXTO_ACA.
[27] https://github.com/dm77/ZBarScanner, sin mantenimiento pero con un sucesor.

INVESTIGACIÓN DETECCIÓN DE MARKERS

Existen numerosas implementaciones disponibles para la detección de Markers utilizando OpenCV. Se reunió información acerca de un conjunto de ellas, para luego testearlas, analizando su arquitectura y verificando su funcionamiento.

IMPLEMENTACIONES ANALIZADAS

Aruco: a minimal library for Augmented Reality applications based on OpenCV.
Sitio: http://www.uco.es/investiga/grupos/ava/node/26
Se trata de una librería en C++ que presenta la posibilidad de realizar una compilación para plataforma Android.

AndroidARMarkerDetection: Android port of Aruco
Sitio: https://github.com/dszafir/AndroidARMarkerDetection
Un port directo de la clase MarkerDetector de Aruco a una implementación en Java utilizando OpenCV

ARma library: Pattern tracking for Augmented Reality
Sitio: http://xanthippi.ceid.upatras.gr/people/evangelidis/arma/
Librería en C++ que presenta una implementación más simple que Aruco

Implementation of a Java Framework for Marker Based Detection in Augmented Reality
Sitio: http://dbis.eprints.uni-ulm.de/986/1/BA_Hartmann_13.pdf
Proyecto final de tesis que implementa una aplicación para Android que realiza detección y tracking de markers para realidad aumentada. La implementación es con OpenCV y puramente en Java. La implementación del proyecto también se encuentra disponible[28].

Marker Detection for Augmented Reality Applications
Sitio:
http://www.infi.nl/blog/view/id/56/Marker_Detection_for_Augmented_Reality_Applications

[28] URL https://github.com/xamino/bachelor_framework_git

Presenta la implementación y explicación de un algoritmo para detectar markers cuadrados.

OpenCV Square Tracking Android JNI

Sitio: https://github.com/victorkp/OpenCV-Square-Tracking-Android-JNI

Presenta una aplicación para Android que realiza la detección de cuadrados con OpenCV cuya implementación está realizada con JNI.

DOCUMENTACIÓN ÚTIL

Además de las implementaciones antes mencionadas encontramos artículos que proveen información útil adicional.

- Black and White Marker Detection[29] presenta la idea básica detrás de un algoritmo elemental para realizar la detección de markers con forma cuadrada.
- Qt and OpenCV[30] explica cómo realizar una corrección de perspectiva en una imagen. Esto es esencial en el momento en el que se detecta un marker y se debe analizar la información dentro del mismo. Si el marker no se encuentra de frente a la cámara y existe inclinación, esta debe compensarse para poder leer los datos.
- Corrección automática de perspectiva para cuadriláteros[31] explica también cómo realizar la corrección de perspectiva.

TESTING DE IMPLEMENTACIONES

ARUCO

Compila directamente con todos los valores por defecto del proyecto, descargado desde su sitio web[32] en su versión 1.2.5.

Ejecutando la aplicación de testing "aruco_test live" que toma los frames desde una webcam podemos ver que funciona detectando los markers rápidamente siendo estable sin presentar crashes. Al detectar markers se puede ver que la librería tiene información de tracking como si fueran algún estilo de coordenadas y detecta los markers con algunos ángulos muy interesantes respecto a la cámara que realiza la captura.

[29] URL http://iplimage.com/blog/cv-img-tec-black-white-marker-detection/
[30] URL https://goo.gl/m9otKE
[31] URL https://github.com/bsdnoobz/opencv-code/blob/master/quad-segmentation.cpp
[32] URL http://sourceforge.net/projects/aruco/files/1.2.5/

Inicialmente, no logramos compilar el código para la plataforma Android, pero se pensó que la solución al inconveniente quizás solo fuera cuestión de modificar archivos CMake.

Esta librería parece ser una muy buena opción estable para detectar y realizar tracking de marcadores, por lo tanto, consideramos que debíamos invertir más tiempo en lograr realizar la compilación para Android o, de no obtenerse esto, al menos una adaptación para Android con solo lo mínimo indispensable.

Para compilar para Android hay que bajar una versión actualizada de android-cmake para ndk r9 desde Github[33], ya que de otro modo vemos algunos errores[34].

Además, fue necesario hacer lo siguiente:

```
export ANDROID_NDK=/opt/android-ndk-r9
export CMAKE_C_COMPILER=/opt/android-ndk-r9/toolchains/arm-linux-androideabi-
4.8/prebuilt/linux-x86_64/bin/arm-linux-androideabi-gcc
export CMAKE_CXX_COMPILER=/opt/android-ndk-r9/toolchains/arm-linux-androideabi-
4.8/prebuilt/linux-x86_64/bin/arm-linux-androideabi-g++
sudo cp /opt/OpenCV-2.4.9-android-sdk/sdk/native/jni/OpenCVModules_armeabi.cmake
/usr/local/share/OpenCV/
sudo chmod 644 /usr/local/share/OpenCV/OpenCVModules_armeabi.cmake
cd /aruco-1.2.5/android

# Por defecto el cmake nos genera todo para compilar aruco como librería estática (extensión
.a)
# pero es más fácil si compilamos como librería shared (extensión .so)
# Así al momento de importarla en nuestra implementación en android todo es más simple
# Para esto hay que modificar el archivo CMakeCache.android.initial.cmake
# y buscar donde define la variable BUILD_SHARED_LIBS y especificarla en ON, quedando así:
# set(BUILD_SHARED_LIBS ON CACHE BOOL "" )

# Además, modificamos el script cmake_android_armeabi.sh para incluir en la línea de cmake
el siguiente parámetro:
# -DOpenCV_DIR=/opt/OpenCV-2.4.9-android-sdk/sdk/native/jni
sh ./scripts/cmake_android_armeabi.sh

# hay que modificar algunos flags porque quedaron mal los includes
# buscamos el archivo build_armeabi/src/CMakeFiles/aruco.dir/flags.make
```

[33] https://github.com/taka-no-me/android-cmake
[34] https://code.google.com/p/android-cmake/issues/detail?id=15

```
# -I/usr/sdk/native/jni/include se modifica por -I/opt/OpenCV-2.4.9-android-
sdk/sdk/native/jni/include
# Una vez modificado eso hacemos:
cd build_armeabi/
make
```

Esto va a compilar una librería en /aruco-1.2.5/android/build_armeabi/libs/armeab/libaruco.so que deberíamos poder utilizar en alguna implementación con JNI en nuestra aplicación Android.

Más tarde, también comprobamos que es posible compilar directamente con Android NDK si se agregan todos los archivos fuente (cpp y hpp) de aruco en un proyecto y se configura correctamente el archivo Android.mk.

Algo interesante a destacar de esta librería, es que presenta la posibilidad de extenderla para detectar markers con un formato diferente al que Aruco maneja por defecto.

No existe suficiente documentación sobre la librería, a excepción de lo explicado en la página web de la librería, pero el código presenta gran cantidad de comentarios e información, documentando las clases, métodos y funciones.

Se puede ver en el archivo *aruco/markerdetector.h* que el método setMakerDetectorFunction de la clase MarkerDetector puede ser utilizado para registrar una nueva función callback que realice la detección de un marker diferente. Esta callback debe respetar una cierta estructura donde recibirá como parámetro una imagen que contendrá un "potencial" marker. Estos potenciales markers serán regiones rectangulares que posean el color negro.

Esto simplifica en gran medida el trabajo necesario para la detección y tracking de los marcadores definidos previamente para la aplicación FIUBAAR.

En este[35] sitio, su autor efectúa una evaluación técnica de la librería.

ARMA

Para poder compilar tuvimos que manualmente crear el archivo CMakeLists.txt con la configuración necesaria para compilar los archivos cpp. El código descargado de su

[35] http://iplimage.com/blog/aruco-technical-evaluation/

sitio para OpenCV 2.4.3 no compila directamente, presentando varios errores. Se realizaron las correcciones necesarias en el código, en algunos include, tipos de datos y se logró compilar la librería correctamente.

Realizando pruebas con la aplicación "ARma_demo" se puede apreciar que la aplicación no es nada estable en la práctica, causando un crash que cierra la aplicación apenas un marker es detectado.

El crash indica lo siguiente:

```
OpenCV Error: Assertion failed (CV_IS_MAT(objectPoints) && CV_IS_MAT(imagePoints) &&
CV_IS_MAT(A) && CV_IS_MAT(rvec) && CV_IS_MAT(tvec)) in cvFindExtrinsicCameraParams2,
file /opencv-2.4.8/modules/calib3d/src/calibration.cpp, line 1177
terminate called after throwing an instance of 'cv::Exception'
 what():  /opencv-2.4.8/modules/calib3d/src/calibration.cpp:1177:  error:  (-215)
CV_IS_MAT(objectPoints) && CV_IS_MAT(imagePoints) && CV_IS_MAT(A) && CV_IS_MAT(rvec)
&& CV_IS_MAT(tvec) in function cvFindExtrinsicCameraParams2
```

Se abandona la librería ARMA, pero dejando constancia de que se han encontrado comentarios en Internet que quizás ayuden a acomodar el código para no tener problemas con los asserts que causan el crash[36, 37].

El proyecto compila correctamente para Android 4.x con OpenCV 2.4.8 y 2.4.9. La aplicación ejecuta y no dio errores de ejecución, pero aún así no hubo ninguna muestra de que la aplicación detectará marker alguno o mostrara cambio alguno en la ejecución. Quizás no se estaba realizando el testing de manera correcta.

Para compilar tuvimos que crear manualmente el archivo CMakeLists.txt con la configuración necesaria para compilar los archivos cpp. Además, tuvimos que acomodar varias directivas de #include para utilizar los headers correctos de OpenCV.
Luego de los cambios todo compila correctamente y el algoritmo parece funcionar para detectar los cuadrados con bordes de markers. No se experimentaron crashes con tests

[36] http://answers.opencv.org/question/3023/solvepnp-assertion-failed/
[37] https://github.com/kylemcdonald/ofxFaceTracker/issues/15

simples, pero parece que se debería adaptar bastante el código fuente para obtener la funcionalidad deseada para tracking y no solo detección.

OPENCV SQUARE TRACKING ANDROID JNI

Utilizamos el código de este proyecto para la primera versión básica de la aplicación cliente FIUBAAR del sprint 1 donde se pudo comprobar que la detección básica funciona. Aun así, esto solo indica si un cuadrado fue detectado o no. No se realizaron suficientes pruebas con respecto a los ángulos soportados respecto de la cámara.

VUFORIA

Utilizando los ejemplos provistos en el portal de desarrollo de Vuforia[38], se pudo adaptar el código correspondiente a "User Defined Targets" para utilizar el port de Aruco para Android de modo de validar que la imagen cumpla con el borde exterior del marker y que contiene un código QR válido en el centro, previo a generar la imagen que se utiliza para el tracking.

Es importante notar que se realizaron pruebas con las versiones del SDK 3.0.9 y 4.2.3.

Si bien la versión 4.x del SDK contiene mejoras y avances, también presenta algunos cambios que no resultaron convenientes para el presente trabajo, entre ellos la inclusión de una marca de agua en la pantalla[39], el requerimiento de una clave para la aplicación generada en el portal de desarrollo[40] y el cambio de algunas clases que implican un pequeño refactoring del código. Debido a esto se decide hacer uso de la versión 3.0.9 del SDK.

IMPLEMENTACIÓN SELECCIONADA

Es claro -dados los resultados de las pruebas realizadas- que la librería "Aruco" presenta la implementación más estable que encontramos en proyectos libres y/o open source.

Adicionalmente, se pudo incorporar el código fuente de Aruco a un proyecto android y compilar con Android NDK sin problemas y sin necesidad de utilizar cmake con todas las modificaciones antes mencionadas. Todo esto, sumado a la posibilidad de definir

[38] Ver https://developer.vuforia.com/downloads/samples.
[39] Ver http://developer.vuforia.com/library/articles/FAQ/Watermark.
[40] Ver http://developer.vuforia.com/library/articles/Training/Vuforia-License-Manager.

fácilmente un nuevo callback para detectar markers con otro patrón, no deja dudas de que Aruco es la mejor alternativa disponible a integrar en el proyecto para realizar la detección de markers.

Revisión Sprint4: Luego de realizar el test de Vuforia utilizando los "User Defined Targets" se pudo comprobar que simplifica abismalmente la implementación del tracking de markers y la ubicación de objetos 3D, pero se limita el soporte solo a plataformas ARM no pudiendo ejecutar en x86 (que de todas formas no es lo más común). La utilización de un SDK como este, que presenta gran madurez, elimina varios de los problemas que se presentaron cuando se hacía uso de la implementación de Aruco como librería nativa y así también aumenta en gran medida la estabilidad de la aplicación en general.

INVESTIGACIÓN DETECCIÓN DE CÓDIGO QR

Al momento de iniciar la investigación sobre este tema, nos encontramos en una instancia en la que ya habíamos tomado la decisión de utilizar la librería Aruco para la detección de marcadores. Naturalmente, el siguiente paso consistió en investigar las opciones existentes para la detección de códigos QR.

Entre los aspectos a tener en cuenta, se destacó la conveniencia de que la implementación fuera en C/C++ -accediendo al código nativo mediante JNI- para contar con mejores tiempos de respuesta, así como para facilitar la integración con Aruco.

IMPLEMENTACIONES ENCONTRADAS

Nombre: ZBar
Sitio: http://zbar.sourceforge.net/

Nombre: ZXing
Sitio: https://github.com/zxing/zxing/

Nombre: CodeProject Simple QRCode Library
Sitio:http://www.codeproject.com/Articles/593591/Simple-C-Cplusplus-QRCode-Library

Nombre: Quirc
Sitio: https://github.com/dlbeer/quirc

Nombre: QRCode
Sitio: http://qrcode.sourceforge.jp/index.html.en

DOCUMENTACIÓN ÚTIL

- Preparando el IDE para ZBar en Android[41]. Para compilar ZBar con NDK fue de utilidad la información provista en este blog.
- Instrucciones de descarga y compilación de LibDecodeQR[42]. Otra librería para la lectura de códigos QR en C/C++. Finalmente, no fue considerada para un análisis más profundo. Windows, no Android.
- LibDecodeQR instability[43]. En el sitio de PiXCL hicieron muestran inestabilidades en LibDecodeQR y analizan una posible solución.
- QR Code decoder using C++[44]. Alternativa a LibDecodeQR propuesta por PiXCL, también en C/C++ pero de muy antiguo desarrollo y sin mantenimiento.
- Artículos varios sobre ZXing[45].
- How to decode data using ZXing in C++[46].
- Example of a QR Code Reader in C++[47].
- ZXing usando Qt[48].
- A quicker QR Code scanner[49]. Experiencia de uso de ZXing en iOS.
- ZBar memory leak in iOS[50]. Propuesta de posible solución para los problemas de memory leaks de ZBar para iOS.

TESTING DE IMPLEMENTACIONES

ZBar da como resultado problemas de heap corruption (quizás, memory leaks)

Se hizo un branch en el repositorio probando quirc -que funcionó- pero que también dio lugar a errores de heap corruption (al parecer, invocaciones a la función free (de direcciones de memoria ya liberadas).

---

[41] Ver http://www.blackdogfoundry.com/blog/zbar-bar-code-qr-code-reader-android/
[42] Ver http://dsynflo.blogspot.com.ar/2010/06/libdecodeqr-libdecodeqr-is-cc-library.html
[43] Ver http://pixcl.com/libdecodeqr-instability.htm.
[44] Ver http://www.pixcl.com/QR-Code-Decoder.htm.
[45] Ver https://www.openhub.net/p/zxing/rss_articles?page=5.
[46] Ver http://stackoverflow.com/questions/17973641/how-to-decode-data-using-zxing-c.
[47] Ver http://wiki.ssrrsummerschool.org/doku.php?id=robocup2012:qrcode-cppexample.
[48]Ver https://goo.gl/nSZKHL
[49] Ver https://conra.dk/2013/01/06/a-quicker-qr-code-scanner.html.
[50] Ver http://stackoverflow.com/questions/18638319/zbar-memory-leak-on-ios.

Se consideró probable un problema en aruco relacionado al callback, cuando éste no es el standard de aruco.

IMPLEMENTACIÓN SELECCIONADA

Por las referencias y pruebas, tanto ZBar como Quirc son buenas opciones. La decisión respecto de cual utilizar dependía de la resolución del problema de memoria que ambas traían.

La solución adoptada finalmente consistió en utilizar la variante en código Java de la librería ZBar, debido a que cambios en la arquitectura eliminaron la implementación nativa de Aruco.

INVESTIGACIÓN ENGINE 3D ANDROID

La aplicación FIUBAAR Cliente debe poder realizar carga y rendering de modelos 3D. Para ello se hará uso de un engine 3D para plataforma Android.

IMPLEMENTACIONES ENCONTRADAS

Nombre: jPCT 3D Engine
Sitio: http://www.jpct.net/jpct-ae/index.html

Nombre: Rajawali
Sitio: https://github.com/MasDennis/Rajawali

Nombre: min3D
Sitio: https://code.google.com/p/min3d/

Nombre: jMonkeyEngine
Sitio: http://jmonkeyengine.org/

Nombre: libGDX
Sitio: http://libgdx.badlogicgames.com/

DOCUMENTACIÓN ÚTIL

A continuación, se presenta una lista de links con información útil y ejemplos.

- Wiki del proyecto Rajawali[51]. Cuenta con abundante información sobre esta librería, así como numerosos tutoriales para su utilización.
- Integración de Rajawali con Vuforia SDK[52].
- Integración de jPCT con Vuforia SDK[53]. Documentación de esta integración.
- Integración de jPCT con Vuforia SDK[54]. Código fuente de la integración.
- Android Support in the jMonkeyEngine[55].
- Armado del entorno para jMonkeyEngine[56]. Uso del motor jME para Android con JNI/NDK en Eclipse.

IMPLEMENTACIÓN SELECCIONADA

De todas las implementaciones encontradas, se encontró que Rajawali es estable. La compilación fue exitosa sin necesidad de introducir modificaciones en el código. Se trata de una implementación ampliamente utilizada (con numerosas menciones a través de distintos sitios) y es parte de muchos otros proyectos. Como si esto fuera poco, está ampliamente probada la integración con Vuforia, por lo que se considera la mejor opción para el proyecto en curso.

INVESTIGACIÓN CLIENTES REST ANDROID

La aplicación FIUBAAR Cliente debe poder conectarse con el server mediante la invocación de servicios REST. Para ello se hará uso de alguna librería que resuelva la comunicación HTTP y consumo de servicios REST. Esto nos permitirá solo enfocarnos en la lógica del consumo de datos.

IMPLEMENTACIONES ENCONTRADAS

Nombre: Android Asynchronous Http Client

Sitio: http://loopj.com/android-async-http/

[51] URL: https://github.com/MasDennis/Rajawali/wiki
[52] URL: https://github.com/MasDennis/RajawaliVuforia
[53] URL: http://www.jpct.net/wiki/index.php/Integrating_JPCT-AE_with_Vuforia
[54] URL: https://github.com/sidneibjunior/vuforia-jpct
[55] URL: https://jmonkeyengine.github.io/wiki/jme3/android.html
[56] URL: https://jmonkeyengine.github.io/wiki/jme3/eclipse_jme3_android_jnindk.html

Nombre: Retrofit

Sitio: http://square.github.io/retrofit/

Nombre: Spring for Android

Sitio: http://projects.spring.io/spring-android/

Nombre: Volley

Sitio: https://android.googlesource.com/platform/frameworks/volley/

Nombre: RESTDroid

Sitio: https://github.com/PCreations/RESTDroid

Nombre: RoboSpice

Sitio: https://github.com/stephanenicolas/robospice

Nombre: Simple rest API client library

Sitio: https://github.com/darko1002001/android-rest-client

DOCUMENTACIÓN ÚTIL

A continuación, se presenta una lista de links con información útil y ejemplos de cómo realizar requests vía HTTP o utilizar algunas de las librerías encontradas.

- Google I/O – Android REST Client Applications[57].
- Transmitting network data using Volley[58].
- Consuming a Restful web service with Spring for Android[59].
- Retrofit REST client for Android60.
- RoboSpice: Asynchronous Network Requests made easy61.

[57] La URL es https://www.youtube.com/watch?v=xHXn3Kg2IQE.
[58] La URL es https://developer.android.com/training/volley/index.html.
[59] URL http://spring.io/guides/gs/consuming-rest-android/
[60] URL https://content.pivotal.io/blog/retrofit-rest-client-for-android
[61] URL https://goo.gl/oOMIGP

TESTING DE IMPLEMENTACIONES

Los primeros resultados encontrados al buscar librerías para consumir servicios REST en Android fueron Android Asynchronous Http Client y Retrofit.

Android Asynchronous Http Client parece ser utilizada por varias aplicaciones muy conocidas, tal como se puede ver en su página web y en el reporte de AppBrain[62].

Se debe destacar que ambas librerías son muy simples de utilizar, proveen toda la funcionalidad necesaria y además no presentan problemas al consumir una variada selección de servicios REST. La diferencia principal es que Retrofit no permite por su cuenta realizar requests de manera asincrónica y para ello se debe utilizar la librería RoboSpice, que contiene un módulo para integrar con varias de las librerías REST.

Otro framework encontrado fue Spring. Se trata de uno muy completo, pero también suele ser demasiado "pesado" ya que incorpora mucha funcionalidad que no es necesaria en esta aplicación en particular. Debido a esto descartamos Spring for Android.

Por su parte, la librería RESTDroid se encuentra en un release alpha en busca de colaboradores con lo que no nos provee suficiente confianza de que, llegado el caso de encontrarnos con un problema, encontremos la documentación o ayuda necesaria.

IMPLEMENTACIÓN SELECCIONADA

Android Asynchronous Http Client se presenta como la mejor alternativa ya que el carácter asincrónico es muy importante para nuestra aplicación, por realizarse en la misma, tareas intensivas constantemente. Esto último hace que sea necesario poder ejecutar en paralelo diferentes acciones. Además, esta librería, ha sido elegida para importantes desarrollos, como los casos de Pinterest, Instagram, MercadoLibre, etc. lo que genera mayor confianza al momento de la elección.

[62] URL https://goo.gl/khXZGD.{

La aplicación FIUBAAR Cliente debe realizar reconocimiento y seguimiento de manos y dedos para así poder también detectar una cierta cantidad de gestos que permitirán al usuario interactuar con los objetos 3D.

IMPLEMENTACIONES ENCONTRADAS

Implementación N° 01

Sitio: http://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/

Implementación N° 02

Sitio: http://eaglesky.github.io/2015/12/26/HandGestureRecognition/[63]

Implementación N° 03

Sitio: http://s-ln.in/2013/04/18/hand-tracking-and-gesture-detection-opencv/

Implementación N° 04

Sitio: https://www.dropbox.com/sh/t22mblcu02xf7dt/yNwsZY1aBW

Implementación N° 05

Sitio: https://github.com/browny/hand-tracking

Implementación N° 06

Sitio: https://github.com/iphkwan/high-hand

Implementación N° 07

Sitio: https://github.com/Param-Uttarwar/SimpleHandTracking-openCV

Implementación N° 08

Sitio: https://github.com/royshil/HHParticleFilter

Implementación N° 09

Sitio: https://github.com/felipeue/Hand_tracking

Implementación N° 10

Sitio: https://github.com/minghuam/simple-hand-tracking

Implementación N° 11

Sitio: https://github.com/richhiey1996/Hand-Tracking-and-Segmentation

Implementación N° 12

Sitio: https://github.com/AhmedRiahi/hands-tracking

[63] La URL expuesta es la de la última versión al momento de la revisión final del presente informe. La dirección original era https://eaglesky.github.io/blog/2014/03/27/color-based-hand-gesture-recognition-on-android/, la cual ya no está disponible.

TESTING DE IMPLEMENTACIONES

Todas las implementaciones que se han probado están basadas en OpenCV e implementadas mayormente en C y solo unas pocas en Java para la plataforma Android. Esto es útil, ya que la intención es poder realizar la implementación en Android de forma nativa para poder ganar en performance.

En su mayoría, los proyectos encontrados son aplicaciones de escritorio, con lo cual se intentaron compilar los mismos y ejecutar para poder así validar el funcionamiento de cada una más allá de los videos de demostración que se encontraron para algunos de ellos.

IMPLEMENTACIÓN SELECCIONADA

Tras numerosos ensayos, se adoptó la implementación N°1 por logar ésta un mejor desempeño en términos de tiempos de respuesta.

Como mencionamos, hemos seleccionado dos implementaciones de seguimiento de manos y dedos:

1.- https://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/ (nativa)

2.- http://eaglesky.github.io/2015/12/26/HandGestureRecognition/ (java)

De ambas podemos obtener información sobre la cantidad y posición de dedos en cada frame de video resultante.

Sin embargo, esto no es suficiente para "detectar gestos" porque para ello necesitamos un continuo de varios cuadros de video, a partir de los cuales analizaremos cambios en la posición y cantidad de dedos.

Actualmente el FrameService está reuniendo una determinada cantidad de cuadros (configurable desde com.fi.uba.ar.services.FrameService.frameGroupSize) en una cola y recién en ese momento se hace la detección de mano de cada una de ellas de forma individual.

Debemos destacar que, después de realizar pruebas sobre las implementaciones de detección de manos y dedos, se pudo determinar que la utilización de un servicio que procese de forma asincrónica los gestos agrega una demora demasiado grande, lo que impide la consecución de un comportamiento aceptable, ni cercano a tiempo real. La demora (*delay*) que se obtuvo en diferentes dispositivos -con diferentes características de hardware- mostró ser algo inaceptable para la interacción del usuario con la aplicación. Debido a esto, se decide optar por una implementación que realice la detección frame por frame.

Una idea inicial de cómo podríamos detectar gestos es tener una cola de datos detectados para cada frame y mantener esa cola como un historial con una cantidad fija en la cual ingresan datos nuevos y los viejos se van descartando (FIFO) y donde usemos una "ventana" que correspondería a lo que aceptamos como duración de un gesto. De esta ventana tendríamos que ver si hay datos que indiquen el posible inicio de un gesto.

Los gestos a detectar -indicados en nuestra propuesta- serían:

- Si la presencia de una mano realizando un gesto de pinzado delante de la cámara fuera reconocida, la aplicación cliente aplicará un zoom in/out sobre la animación.
- Si la presencia de una mano abierta girando delante de la cámara fuera reconocida, la aplicación cliente aplicará una rotación sobre la animación.
- Si la presencia de una mano realizando un gesto de "cerrar en forma de puño" delante de la cámara fuera reconocida, la aplicación cliente hará desaparecer la animación (concepto de cerrar).
- Si la presencia de una mano cerrada con sólo el dedo índice extendido delante de la cámara fuera reconocida, la aplicación cliente interpretará la existencia de un "puntero" con el que se podrán realizar "clicks" sobre la animación.
- Si la presencia de una mano cerrada con dos dedos extendidos (índice y medio) delante de la cámara fuera reconocida, la aplicación cliente interpretará la existencia de un "puntero" que sujeta y puede arrastrar la animación sobre la pantalla.

Según la investigación realizada, existen tesis dedicadas por completo a la detección de gestos, ya que esto es por sí solo un problema bastante complejo de resolver. Ciertas implementaciones proponen utilizar redes neuronales para entrenar y luego reconocer gestos. Este tipo de sistemas superan la complejidad que se le intenta dar al presente proyecto.

En este proyecto, implementaremos una forma simplificada de detectar un gesto que consistirá en el reconocimiento de las siguientes sucesiones de estados (se debe considerar que Vuforia podría entregar aproximadamente 30 FPS –Cuadros por segundo o *Frames Per Second*-, pero asumiremos que existen dispositivos en los cuales la performance dista de ser óptima, ubicándose en promedio en torno a los 20 FPS):

- INICIO_GESTO: comprende al menos 2 (y no más de 5) frames consecutivos
- SECUENCIA_GESTO: comprende al menos 15 (y no más de 100) frames consecutivos. Aquí se está imponiendo que un gesto no puede tardar más de 5 segundos, aunque podría el usuario decidir mover un objeto alrededor de la pantalla durante dicho tiempo, en cuyo caso luego de 5 segundos se dejará de procesar el gesto y se iniciará la detección nuevamente.
- FINAL_GESTO: ídem INICIO_GESTO, comprende al menos 2 (y no más de 5) frames consecutivos

En cada uno de estos estados se debe encontrar en los frames procesados una cantidad particular de dedos, con ciertas características y posiciones.

Es la intención del equipo, hacer una detección de gestos básica para demostrar el uso que se le puede dar a la aplicación en conjunto con la realidad aumentada dejando claro que la implementación podría ser mejorada o reemplazada en futuras versiones, logrando así mejor o mayor funcionalidad.

A continuación, se definen los estados que componen cada uno de los gestos antes descriptos:

Gesto: Pinzado

- 1.1 INICIO_GESTO: Debe haber solo 2 dedos donde la longitud de uno de ellos debe ser menor (asumimos que sería el pulgar) y además el ángulo entre los dedos debe ser superior a un valor a definir. Ambos dedos deben estar en la misma posición en todos los frames.
- 1.2 SECUENCIA_GESTO: Debe haber solo 2 dedos (pulgar e índice) en un ángulo diferente al INICIO_GESTO. Dicho ángulo debe ir cambiando.
- 1.3 FINAL_GESTO: Se debe detectar una cantidad diferente a 2 dedos.

Gesto: Mano abierta girando

- 2.1 INICIO_GESTO: Deben estar presentes (detectados) los 5 dedos (palma abierta completa) en una misma posición durante todos los frames.
- 2.2 SECUENCIA_GESTO: deben estar presentes los 5 dedos, pero los dedos 0 y 5 (los extremos) deben haber cambiado de posición.
- 2.3 FINAL_GESTO: se debe detectar una cantidad de dedos diferente a 5.

Gesto: Cerrar puño

- 3.1 INICIO_GESTO: Deben estar presentes (detectados) los 5 dedos (palma abierta completa).
- 3.2 SECUENCIA_GESTO: Deben detectarse entre 1 y 5 dedos.
- 3.3 FINAL_GESTO: Se deben detectar 0 dedos

Gesto: Dedo índice click

- 4.1 INICIO_GESTO: Debe haber un único dedo (sería conveniente intentar identificar si es el índice, quizás usando datos del círculo o bounding rect o

quizás convex hull). En todo caso funcionará igual siempre y cuando haya un único dedo. El dedo debe estar posicionado sobre el objeto 3D.

- 4.2 SECUENCIA_GESTO: Se debe continuar detectando un único dedo, pero en diferente posición. El gesto de click corresponde a que el dedo debe estar en una posición, luego subir solo en el eje vertical y volver a bajar hasta la posición inicial (aproximadamente).
- 4.3 FINAL_GESTO: se deben detectar 0 dedos o al menos más de 1 único dedo.

Gesto: Dos dedos puntero

- 5.1 INICIO_GESTO: Debe haber presentes solo dos dedos sobre el objeto 3D, de similar longitud y un ángulo entre ellos pequeño. Esto nos obliga –probablemente- a no poder tener los dos dedos juntos porque el algoritmo no los detectaría como dos, sino como uno único. Esto implica que resulta necesaria una separación entre ambos dedos.
- 5.2 SECUENCIA_GESTO: Se debe continuar detectado solo 2 dedos y en diferentes posiciones. Se tomará un punto entre los 2 dedos como posición a la cual se debe mover el objeto 3D.
- 5.3 FINAL_GESTO: se deben detectar alguna cantidad diferente a 2 dedos.

IDEAS DE IMPLEMENTACIÓN

Android posee formas de detectar gestos del usuario basados en lo que se toque en la pantalla a partir de sus *"touch events"* (eventos de contacto). Con esto ya se puede realizar detección de pinchado, por ejemplo.

Existe abundante documentación acerca de cómo utilizar la detección de gestos:

- Detecting common gestures[64].
- Dragging and Scaling[65].
- Handling Multi-Touch Gestures[66].
- Clase GestureDetector[67].
- Clase MotionEvent[68].

[64] URL https://developer.android.com/training/gestures/detector.html
[65] URL https://developer.android.com/training/gestures/scale.html
[66] URL https://developer.android.com/training/gestures/multi.html
[67] URL https://developer.android.com/reference/android/view/GestureDetector.html
[68] URL https://developer.android.com/reference/android/view/MotionEvent.html

Algo que podría proveernos con la implementación lista de los gestos posiblemente sería la idea de generar *"touch events"* basados en las posiciones donde detectamos los dedos, como si en realidad los lugares donde aparecen los dedos en la cámara fueran puntos "tocados" en la pantalla del dispositivo.

Para ciertos casos esto resolvería la lógica directamente, por ejemplo, en el caso de "pinchado" (hacer zoom en un objeto, que en Android se conoce como "scale"). Pero para otros gestos habría que hacer algún tipo de traducción, ya que los clicks (llamados *"tap"* en los gestos de android) no se pueden lograr solo marcando un *touch event* y para enviar un *"tap"* nosotros deberíamos detectar cierto movimiento que consideremos "click" / "tap".

Para el caso de girar el objeto, nuestro gesto implica una palma completa (5 dedos) girando, pero en el dispositivo girar implica mantener dos dedos continuamente haciendo touch y girando.

Entonces se presenta la situación donde debemos tener algo que funcione como "traductor" de nuestros gestos a gestos de android.

Quizás como caso inicial el tema del pinchado (zoom) podemos hacerlo directamente con touch y ver si funciona directamente.

Otra alternativa sería utilizar Robotium para generar eventos tipo click.

- "$1" Unistroke Recognizer[69]. Reconocedor de gestos en 2D implentado en JavaScript y C# que emplea técnicas de machine learning. Provee un documento académico que explica los conceptos involucrados y que es la razón por la que incluimos el documento en la presente lista.
- MooseGesture [70] .Código en Phyton que reconoce como gestos, movimientos de un mouse u otro origen en forma vertical, horizontal o en diagonales.
- Cómo crear un MotionEvent en Android[71].
- Cómo simular un MotionEvent en Android[72].
- Por qué esta simulación de MotionEvent no funciona[73]. Ejemplo de errores en la simulación de MotionEvent.
- Cómo enviar MotionEvents sintetizados a través del SO[74]. Implicancias de seguridad en la inyección de MotionEvents.
- Foro sobre plataforma Android[75]. Búsquedas en este grupo relacionadas con simulación de MotionEvents arrojan como resultados discusiones relevantes.
- Robotium[76]. Framework para automatización de tests en Android con soporte para aplicaciones nativas e híbridas.
- Touch, hold, swipe, release gesture simulation in Android Unit test[77].
- Clase TouchUtils[78]. Permite generar eventos Touch, seguidos o no de una acción de dragging, con el fin de testear la interacción con el usuario.
- Simulating Swipes in your Android Tests[79].
- Simulation of Injecting Multi-Touch events in Android[80].
- Injecting events programatically on Android[81].
- Handling Single and Multi-Touch on Android[82]. Sobresaliente tutorial.
- Dispatching TouchEvents in Fragments[83].
- Android Gesture Detector Framework[84]. Framework pensado para operar con dispositivos multi-touch basado en el ScaleGestureDetector de Android.

---

[69] URL https://depts.washington.edu/madlab/proj/dollar/index.html
[70] URL https://github.com/asweigart/moosegesture
[71] URL https://goo.gl/ULYeQw
[72] URL https://goo.gl/NhXwgJ
[73]URL https://goo.gl/ixeC4d
[74] URL https://goo.gl/BHioqe
[75] URL https://goo.gl/emKwua
[76] URL https://goo.gl/o9Aznj
[77]URL https://goo.gl/UWJJ3W
[78] URL https://goo.gl/uPqoAx
[79] URL http://www.jasondl.ee/posts/2013/simulating-swipes-in-your-android-tests.html
[80] URL https://nandhanthiravia.blogspot.com.ar/2014/01/inject-multi-touch-event.html
[81] URL http://www.pocketmagic.net/injecting-events-programatically-on-android/
[82] URL http://www.vogella.com/tutorials/AndroidTouch/article.html
[83] URL https://goo.gl/AsrWOb

- Detección de gestos en Android[85]. Implementada con listener y RxJava Observable.
- Sensey[86]. Otra librería para la detección de gestos.
- Android Guides: Gestures and Touch events[87]. Tutorial claro.
- Android: Detecting a pinch gesture[88].
- Moving views in Android – Drag and Drop[89].
- How to generate zoom/pinch gesture for testing[90].
- Generating multi-touch MotionEvents for testing[91].
- Android MotionEvent pointer index confusion[92].

IMPLEMENTACIÓN SELECCIONADA

Luego de realizar pruebas con la API de gestos provistas por Android y de comprobar que es posible generar eventos de tocado ("Motion Events") de forma programática, se opto por utilizar esto generando un evento de tocado en cada una de las posiciones en las que se detectara un dedo. Y, su vez, actualizando todos esos eventos cuadro por cuadro mediante el seguimiento de la mano

[84] URL https://github.com/Almeros/android-gesture-detectors
[85] URL https://github.com/pwittchen/gesture
[86] URL https://github.com/nisrulz/sensey
[87] URL https://github.com/codepath/android_guides/wiki/Gestures-and-Touch-Events
[88] URL https://goo.gl/FPx2gA
[89] URL https://blahti.wordpress.com/2011/01/17/moving-views-part-2/
[90] URL https://goo.gl/fqtNdP
[91] URL https://goo.gl/Lq0hK1
[92] URL https://goo.gl/fFXPDW

CONCLUSIONES DEL PROYECTO

Tras la realización del proyecto, surgen diversos elementos que podemos considerar como conclusiones.

Consideramos haber cumplido los objetivos propuestos al inicio del proyecto en cuanto a las características del conjunto de las aplicaciones elaboradas, y también en cuanto a lo que hace a la profundización del conocimiento de los autores sobre el diseño y desarrollo de soluciones para problemas complejos en la plataforma Android.

Nos hemos familiarizado con herramientas que hoy constituyen un estándar en el procesamiento de imágenes, al tiempo que hemos advertido la complejidad de lidiar con este tipo de tareas, que hacen uso extensivo de los recursos del sistema y que son muy susceptibles a cambios en el ambiente en el que luego los dispositivos son utilizados (iluminación, contraste, etc.) así como a las características del hardware disponible (especialmente, la cámara del dispositivo).

Hemos disfrutado de las ventajas de contar con NDK para asimilar código heredado pensado inicialmente para otras plataformas y desarrollado en un lenguaje ampliamente difundido en el ámbito científico y técnico como lo es C/C++.

Manejamos la complejidad asociada a la depuración en un entorno de procesos concurrentes, recurriendo a herramientas específicas de ese nicho de desarrollo.

Todo lo mencionado y lo que omitimos por brevedad, ha enriquecido nuestra experiencia en el campo de la Ingeniería en Informática. En cuanto al futuro del proyecto, entendemos que éste no ha alcanzado aún la madurez necesaria para constituir una solución comercial. El primer paso para ello sería sin dudas el de migrar el código de la aplicación cliente al entorno asociado a Android Studio, incorporando para ello un script para Gradle para su compilación, de modo de facilitar la integración con los proyectos de mayor actualidad que podrían complementar al presente o mejorar algunos de sus aspectos. Entre los posibles usos comerciales de una solución sólidamente estable podemos imaginar los de participar en campañas de marketing -dirigidas especialmente a captar la atención del público más joven- así como también los de herramienta pedagógica en la enseñanza en todos los niveles, y en este mismo contexto, como contribución a la construcción de secciones interactivas en libro de texto.

APÉNDICE I - ARMADO DEL ENTORNO DE DESARROLLO

La presente sección constituye una guía práctica para que un programador pueda configurar completamente un entorno de desarrollo que le permita trabajar en el proyecto de la aplicación FIUBAAR.

Lo aquí expuesto es válido para las plataformas Linux y Windows. Todas las herramientas necesarias se encuentran disponibles para ambas plataformas, lo que permite a cada programador elegir aquella en la que encuentre mayor comodidad.

HERRAMIENTAS REQUERIDAS

- **Java JDK** (versión 7.x o superior) [93]
- **Android ADT Bundle** (contiene Android SDK)[94]
- **Android Native Development Kit** (NDK)[95]
- **OpenCV for Android** - (android-sdk)[96]
- **Activator** [97]

CONFIGURACIÓN

Para la correcta configuración se deben seguir los siguientes pasos:

1. Proceder a instalar en primer lugar Java JDK.
2. Luego se debe descomprimir el archivo de Android ADT Bundle siguiendo los pasos descriptos en la guía suministrada por Google Android[98].
3. Descomprimir Android NDK.

---

[93] Disponible en http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html y http://www.oracle.com/technetwork/java/javase/downloads/index.html
[94] Disponible en http://developer.android.com/sdk/index.html , http://web.archive.org/web/20140829113546 , http://developer.android.com/sdk/index.html y http://dl.google.com/android/adt/adt-bundle-windows-x86_64-20140702.zip
[95] Disponible en https://developer.android.com/tools/sdk/ndk/index.html http://web.archive.org/web/20130927074051 , http://developer.android.com/tools/sdk/ndk/index.html , http://dl.google.com/android/ndk/android-ndk-r9-windows-x86_64.zip
[96] Disponible en http://sourceforge.net/projects/opencvlibrary/files/opencv-android/
[97]Disponible en https://www.typesafe.com/activator/download, https://www.lightbend.com/activator/download
[98] Disponible en https://goo.gl/wnlksh.

4. Iniciar Android ADT y seguir los pasos descriptos en la siguiente guía de OpenCV para importar los proyectos OpenCV y compilar la librería que luego deberemos importar, a su vez, en nuestros proyectos[99].
5. Al compilar el proyecto OpenCV obtendremos el archivo /OpenCV-2.4.8-android-sdk/sdk/java/bin/opencv library - 2.4.8.jar (o la versión de OpenCV descargada). La librería OpenCV generada se deberá importar como dependencia en los proyectos de Android que luego se creen.
6. Incorporar la librería Rajawali: 3DEngine[100] al proyecto.

NOTA IMPORTANTE: Una vez instalado el Java JDK 7 o superior, existe una alternativa que facilitaría enormemente el armado del entorno de desarrollo. Se trata del Tegra Android Development Pack (TADP) y es la opción recomendada para una instalación desde cero, propuesta desde el sitio de OpenCV[101]. TADP descarga de Internet e instala las herramientas de desarrollo necesarias y las configura. Adicionalmente contiene otras herramientas que no usaremos en este proyecto, pero la sencillez de uso sugiere al TADP como una opción a explorar si no hay limitaciones en recursos.

DEFINICIÓN DE VARIABLES DE ENTORNO

El proyecto hará uso de algunas variables de entorno para la compilación y referencia de ciertas librerías. Por lo tanto, se deberán definir las siguientes variables de entorno en el sistema operativo

```
OPENCV_HOME="path al proyecto /OpenCV-2.4.9-android-sdk"
NDKROOT="path al directorio de android-ndk"
ACTIVATOR=”path al directorio de Activator”
```

Para hacerlo en Windows, existen numerosas guías paso a paso disponibles en Internet[102].

En Linux se puede ejecutar en una terminal

```
export OPENCV_HOME="<path>/OpenCV-2.4.9-android-sdk"
```

[99] http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html

[100] Disponible en https://github.com/MasDennis/Rajawali.

[101] OpenCV sugiere la instalación del Tegra Android Development Pack según manifiesta en http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_dev_intro.html#android-dev-intro

[102] http://www.computerhope.com/issues/ch000549.htm

```
export NDKROOT="<path>/android-ndk-r9"
```

Es conveniente también editar el archivo $HOME/.bashrc y agregar al final la definición de estas mismas variables para que se definan automáticamente en cada reinicio de sistema.

DISPOSITIVOS ANDROID

Es posible ejecutar las aplicaciones desarrolladas tanto en un emulador como en un dispositivo real.

Es preferible realizar la ejecución de la aplicación en un dispositivo real ya que para el procesamiento de imágenes se necesita contar con una buena velocidad en términos de cuadros por segundo (*FPS, Frames Per Second*), que se consigue con una cámara real y no así tanto con el emulador utilizando una webcam.

Para poder conectar un dispositivo real y hacer uso del mismo desde el IDE, es necesario seguir los pasos indicados la documentación oficial de Android[103] (Desde Diciembre/2014 la explicación incluye algunos detalles de configuración en el script de Gradle)

Se indica allí como configurar dispositivos tanto para Linux como Windows (donde es necesario instalar un driver USB)

HERRAMIENTAS ADICIONALES

- Bitbucket (Repositorio GIT privado, Issue tracking, Wiki y Team management): Registrar una cuenta en el sitio https://bitbucket.org y pedir una invitación al repositorio del equipo FIUBAAR vía email a proyecto.fiubaar@gmail.com
- Cliente para repositorio GIT: Se puede utilizar cualquier cliente para GIT ya sea mediante línea de comando o algún otro con interfaz gráfica. En particular, es conveniente el uso del plugin de eclipse para GIT llamado EGit[104]. Para Windows se recomienda usar msysgit[105], actualmente llamado "git for Windows"[106].
- Aplicación para diagramas UML: Descargar Modelio desde su sitio[107]. Esta herramienta está basada en Eclipse, así que simplemente se descomprime y se ejecuta. Otra opción consiste en recurrir a una herramienta web como "draw.io"[108].

[103] https://goo.gl/awNq1w
[104] http://www.eclipse.org/egit/
[105] http://msysgit.github.io/
[106] https://git-for-windows.github.io/
[107] http://www.modelio.org/

- Gradle (build & dependency management): Android ADT Bundle ya posee integrado lo necesario para hacer uso de Gradle. Más información sobre la herramienta se puede encontrar en el sitio de Google desarrollado a tal fin[109].
- GenyMotion[110]: un emulador alternativo mucho más veloz que el suministrado por Google. Está basado en virtual box y tiene disponible un plug-in para Eclipse que nos permite iniciar los dispositivos desde allí. Está basado en arquitectura x86, por lo que resulta necesario compilar el código nativo también para x86. Lamentablemente, a la fecha de finalización del presente proyecto, ya no es posible su utilización sin el previo pago de una licencia.

NOTAS ADICIONALES

Al inicio del proyecto, la herramienta Android Studio -que actualmente es el único IDE con soporte oficial de Google- se encontraba en estado Beta, no permitiendo el uso de Android NDK. Android NDK es necesario para poder compilar código nativo en C/C++ en los casos en los que se desea hacer uso de librerías de más bajo nivel con interfaces JNI. Sabiendo desde el inicio que NDK sería fundamental en el proyecto para tareas tales como la detección de manos y gestos -en las cuales la performance y velocidad en el procesamiento de cuadros de video (*frames*) son críticas- se adopta desde el comienzo a Eclipse como IDE.

Junto con Android Studio, Google introduce a Gradle como sistema de enlace y compilación (*build system*).

Considerando la complejidad del sistema desarrollado y que sus autores están familiarizados con el entorno de desarrollo actual; sumado esto a que Google no ha introducido cambios que invaliden lo desarrollado en el entorno previo, se concluye que no está justificado de momento migrar a la plataforma de desarrollo actual. Será, sin embargo, un elemento a tener en cuenta si se decide avanzar en el desarrollo una vez concluida la presentación del trabajo.

[108] https://www.draw.io/
[109] http://tools.android.com/tech-docs/new-build-system/user-guide
[110] https://www.genymotion.com/

BUENAS PRÁCTICAS

Los programadores que formen parte del proyecto deberán en lo posible seguir buenas prácticas en todos los aspectos comprendidos en la implementación de la aplicación.

Se describen a continuación algunas de las más importantes.

COMENTARIOS

Agregar comentarios en el código en aquellos lugares en los que se realicen operaciones complejas. Se recomienda ser consciente y considerado con la cantidad de comentarios y no sobrepasarse con esto[111].

Hacer uso de JavaDoc en las clases y métodos. Esto permitirá luego generar automáticamente la documentación de la aplicación y API's. Los IDE's en uso auto-completan el template de JavaDoc si se comienza a comentar sobre un método o clase con /** y se presiona ENTER[112].

DESARROLLO EN ANDROID

Más allá de la incorporación de comentarios, se recomienda para el desarrollo de aplicaciones en la plataforma Android un sinnúmero de prácticas que mejoran la calidad del código –a través de mejorar su legibilidad o la performance de su ejecución- y/o contribuyen a una mejor experiencia de usuario. En Internet se desarrolla extensivamente el tema, pero algunos artículos son especialmente recomendables.

- Google Android – Mejores prácticas[113]
- Google Android – Entrenamiento[114]
- Siendo épico: Mejores prácticas para el desarrollo en Android[115]
- Guía de mejores prácticas para Android Java[116]

---

111 http://javarevisited.blogspot.com.ar/2011/08/code-comments-java-best-practices.html, http://www.hongkiat.com/blog/source-code-comment-styling-tips/, https://dzone.com/articles/5-best-practices-commenting.
112 En StackOverflow https://goo.gl/Lk91ag y en el sitio de Jetbrains https://goo.gl/nJsEY2.
113 http://developer.android.com/guide/practices/index.html
114 http://developer.android.com/training/index.html
115 http://www.slideshare.net/retomeier/being-epic-best-practices-for-building-android-apps
116 http://forum.xda-developers.com/showthread.php?t=2635275

REPOSITORIO PÚBLICO

Desde el sitio https://github.com/fiubaar se tiene acceso a los repositorios del proyecto. Este cuenta con un repositorio para la aplicación Cliente, otro para la aplicación Servidor, uno adicional que contiene la documentación y un cuarto destinado a albergar el código que hace a la página web del proyecto.

El código de cada aplicación está, de este modo, al alcance de quien quiera experimentar con él. Gracias a estos repositorios y la presente documentación, no será difícil compilar los distintos componentes del proyecto.

REPOSITORIO PRIVADO

A modo de comentario, creemos necesario mencionar que, durante la mayor parte del proyecto, y aún en este momento, trabajamos con un repositorio privado en BitBucket (https://bitbucket.org/fiubaar). En este repositorio, de todos los "*workflows*" (modos o flujos de trabajo) posibles con los que se trabaja comúnmente en git, elegimos una variante de "gitflow"[117], que es, por cierto, uno de los más conocidos. Si bien "gitflow" presenta un esquema de trabajo interesante, pasible de ser aplicado a cualquier desarrollo, como dijimos, no se hace uso de este workflow en un sentido estricto. El modo de trabajo por el cual optamos, utilizó un branch "master" default inicial, pero creando una rama (*branch*) nueva por cada sprint. A cada rama se le dio el nombre de "sprintN", donde N es el número de sprint.

No se hizo un "merge" de estos branches, ya que estuvo previsto desde un inicio que cada sprint tuviera un conjunto de entregables y una versión intermedia de la aplicación.

Solo al final del proyecto, una vez contando con la versión final completa de la aplicación, pasamos a tener en el branch "master", la release final de todo lo necesario.

Para más información sobre cómo utilizar GIT en general se puede recurrir a los tutoriales disponibles en Internet, entre los cuales son especialmente recomendables aquellos publicados por la empresa Atlassian[118].

---

[117] https://www.atlassian.com/git/workflows#!workflow-gitflow
[118] Buenos ejemplos son https://www.atlassian.com/git/tutorial, https://confluence.atlassian.com/display/BITBUCKET/Using+Git+branches