FIUBAAR

Realidad Aumentada en ANDROID

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

2017

Esteban Guillardoy (P.81597) eguillardoy@gmail.com & Alejandro E. Aguilar (P.73935) alezek@gmail.com

Facultad de Ingeniería de la Universidad de Buenos Aires

17/02/2017

Tabla de contenido

[Introducción 5](#_Toc477619863)

[Objetivos 5](#_Toc477619864)

[Motivación 5](#_Toc477619865)

[Metodología 6](#_Toc477619866)

[Especificación de Requerimientos 6](#_Toc477619867)

[Administración del Proyecto 8](#_Toc477619868)

[Calendario 8](#_Toc477619869)

[Replanificación N°1 - Al inicio del Sprint N°3, 20/Octubre/2014. 8](#_Toc477619870)

[Replanificación N°2 - Al inicio del Sprint N°4, 26/Enero/2015. 8](#_Toc477619871)

[Replanificación N°3 - Al inicio del Sprint N°5, 26/SEPTIMBRE/2016. 8](#_Toc477619872)

[Sprint N°1 (137HH)- Inicio: 04/08/2014 Finalización: 07/09/2014 8](#_Toc477619873)

[Sprint N°2 (144HH) - Inicio: 08/09/2014 Finalización: 19/10/2014 9](#_Toc477619874)

[Sprint N°3 (156HH) - Inicio: 15/12/2014 Finalización: 25/01/2015 9](#_Toc477619875)

[Sprint N°4 (152HH) - Inicio: 03/08/2015 Finalización: 13/09/2015 10](#_Toc477619876)

[Sprint N°5 (163HH) - Inicio: 26/09/2016 Finalización: 06/11/2016 10](#_Toc477619877)

[Sprint N°6 (151HH) - Inicio: 07/11/2016 Finalización: 18/12/2016 10](#_Toc477619878)

[Sprint N°7 (154HH) - Inicio: 02/01/2017 Finalización: 12/02/2017 11](#_Toc477619879)

[Burndown Charts por Sprint 11](#_Toc477619880)

[Arquitectura de la Aplicación 14](#_Toc477619881)

[Arquitectura de Android 14](#_Toc477619882)

[El Patrón Model-View-Presenter 15](#_Toc477619883)

[Performance 15](#_Toc477619884)

[Evolución de la Arquitectura 17](#_Toc477619885)

[Arquitectura 1 - Inicial 17](#_Toc477619886)

[Arquitectura 2 18](#_Toc477619887)

[Arquitectura 3 19](#_Toc477619888)

[Arquitectura 4 - Final 21](#_Toc477619889)

[Diagrama de paquetes 23](#_Toc477619890)

[Diagramas de Clases 26](#_Toc477619891)

[Clase mainApplication 26](#_Toc477619892)

[Clase mainActivity 27](#_Toc477619893)

[Clase mainController 28](#_Toc477619894)

[Clase objectAR 29](#_Toc477619895)

[Clase marker 30](#_Toc477619896)

[Clase markerDetector 31](#_Toc477619897)

[Clase nativeHandTrackingDetector 32](#_Toc477619898)

[Clase handDebugView 33](#_Toc477619899)

[Clase vuforiaFragment 34](#_Toc477619900)

[Clase vuforiaRenderer 35](#_Toc477619901)

[Clases del paquete Engine3D: Base3DFragment y modelFragment 36](#_Toc477619902)

[Clases del paquete rest: RestClient y download3DModelAsyncTask 37](#_Toc477619903)

[Diagramas de Secuencia 38](#_Toc477619904)

[Secuencia: Escaneo de un marcador 38](#_Toc477619905)

[Secuencia: Detección de Mano 39](#_Toc477619906)

[Diagrama de Despliegue 40](#_Toc477619907)

[Diseño de la Interfaz de Usuario 41](#_Toc477619908)

[Modelo de datos 42](#_Toc477619909)

[Investigaciones durante el desarrollo 44](#_Toc477619910)

[Investigación de Markers 44](#_Toc477619911)

[Estado del arte 44](#_Toc477619912)

[Requerimientos FIUBAAR 44](#_Toc477619913)

[Elección del marker núcleo 45](#_Toc477619914)

[Tracking del marker 45](#_Toc477619915)

[Marker seleccionado 46](#_Toc477619916)

[Documentación adicional útil 46](#_Toc477619917)

[Investigación Detección de Markers 49](#_Toc477619918)

[Implementaciones analizadas 49](#_Toc477619919)

[Documentación útil 50](#_Toc477619920)

[Testing de implementaciones 50](#_Toc477619921)

[Implementación seleccionada 54](#_Toc477619922)

[Investigación Detección de código QR 56](#_Toc477619923)

[Implementaciones encontradas 56](#_Toc477619924)

[Documentación útil 56](#_Toc477619925)

[Testing de implementaciones 57](#_Toc477619926)

[Implementación seleccionada 57](#_Toc477619927)

[Investigación Engine 3D Android 58](#_Toc477619928)

[Implementaciones encontradas 58](#_Toc477619929)

[Documentación útil 58](#_Toc477619930)

[Implementación seleccionada 59](#_Toc477619931)

[Investigación Clientes REST Android 59](#_Toc477619932)

[Implementaciones encontradas 59](#_Toc477619933)

[Documentación útil 60](#_Toc477619934)

[Testing de implementaciones 60](#_Toc477619935)

[Implementación seleccionada 60](#_Toc477619936)

[Investigación Hand & Finger Tracking 61](#_Toc477619937)

[Implementaciones encontradas 61](#_Toc477619938)

[Testing de implementaciones 62](#_Toc477619939)

[Implementación seleccionada 62](#_Toc477619940)

[Investigación sobre detección de gestos 63](#_Toc477619941)

[Ideas de Implementación 66](#_Toc477619942)

[Links 67](#_Toc477619943)

[Implementación seleccionada 69](#_Toc477619944)

[Conclusiones del Proyecto 69](#_Toc477619945)

[Apéndice I - Armado del Entorno de Desarrollo 71](#_Toc477619946)

[Herramientas requeridas 71](#_Toc477619947)

[Configuración 71](#_Toc477619948)

[Definición de variables de entorno 72](#_Toc477619949)

[Dispositivos Android 73](#_Toc477619950)

[Herramientas adicionales 73](#_Toc477619951)

[Notas Adicionales 74](#_Toc477619952)

[Buenas Prácticas 75](#_Toc477619953)

[Comentarios 75](#_Toc477619954)

[Desarrollo en Android 75](#_Toc477619955)

[Repositorio público 75](#_Toc477619956)

[Repositorio privado 76](#_Toc477619957)

[Referencias 76](#_Toc477619958)

[Apéndice II - Bibliografía 77](#_Toc477619959)

# 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 como “Aplicación cliente” y es la que se ejecuta en el dispositivo móvil con sistema operativo Android. La segunda, se denomina “Aplicación 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/SEPTIMBRE/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 aún 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: 12/02/2017

* 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: Pendiente** [T.Estimado: 6 semanas, T.Real: x semanas]

Cantidad Total de Horas Hombre: 1057

## Burndown Charts por Sprint

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.

# 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]](#footnote-1) [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]](#footnote-2) [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]](#footnote-3). [3]. La misma posee como capa base un sistema operativo Linux y posee un componente principal relacionado a las aplicaciones llamado Dalvik VM (Maquina virtual). Cada aplicación es ejecutada en un proceso y el mismo posee una instancia de una DVM[[4]](#footnote-4) [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 (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, AOT está presente a partir de la versión 4.4 de Android, aunque no por defecto sino que debe ser habilitado[[5]](#footnote-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]](#footnote-6) [5], es posible implementarlo de forma muy coherente, tal como puede apreciarse en diversos ejemplos[[7]](#footnote-7) [6] [7] [8]. Para ser más estrictos, en el caso del presente proyecto, el patrón que se utilizó en realidad es Model-View-Presenter (MVP).[[8]](#footnote-8) [9]. 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.

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]](#footnote-9) [10] [11].

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]](#footnote-10) [12] [13] [14] [15] [16] [17] [18]. 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.

## Evolución de la Arquitectura

Al finalizar el desarrollo del proyecto, hicimos 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* decidimos hacer en su lugar, en caso de requerir modificación y *por qué* tomamos 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).

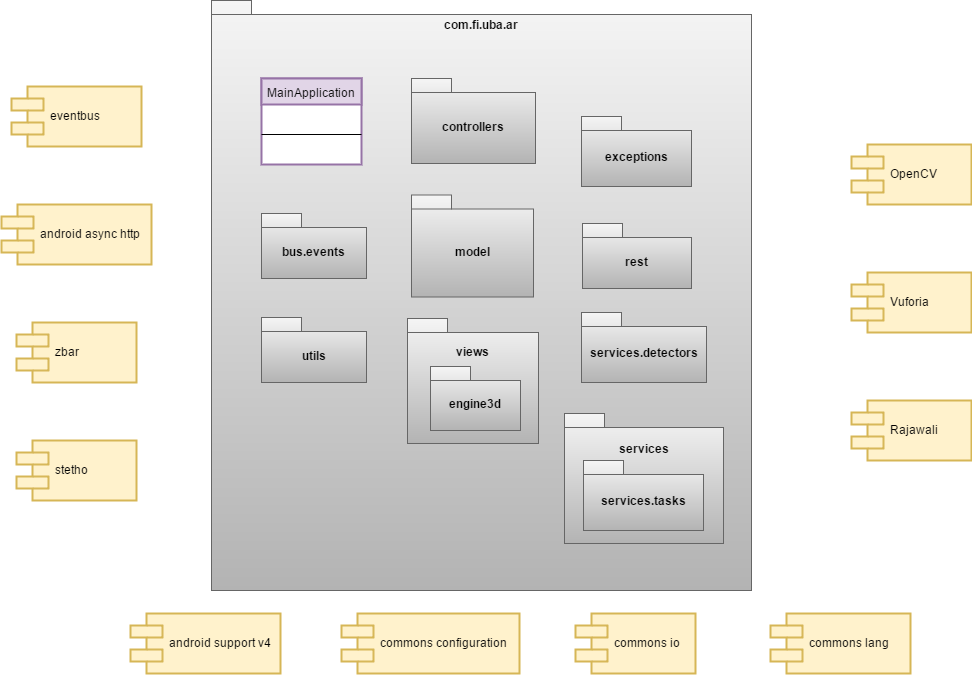
Gracias a que Android ya posee APIs para manejar ciertos gestos, se logra simplificar la lógica que inicialmente se pensaba implementar, delegando directamente en las mismas el reconocimiento de los gestos y permitiendo además una integración más estandarizada para la plataforma en base a métodos “listeners” 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.

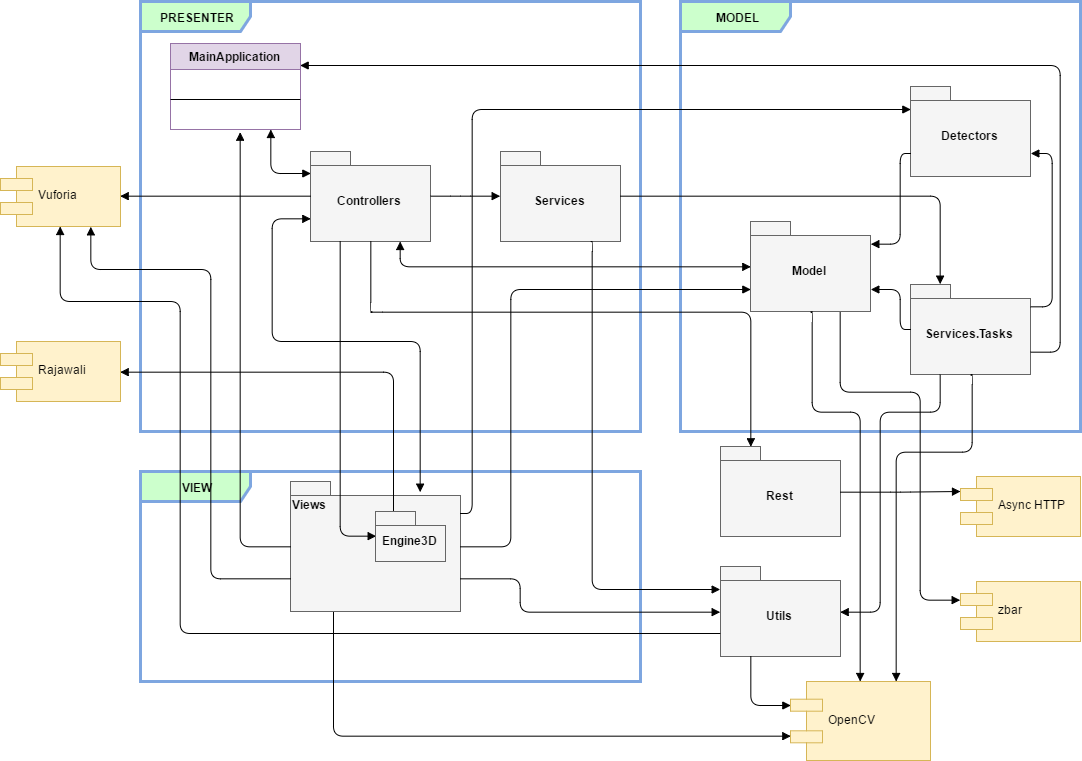
# 

## Diagrama de paquetes

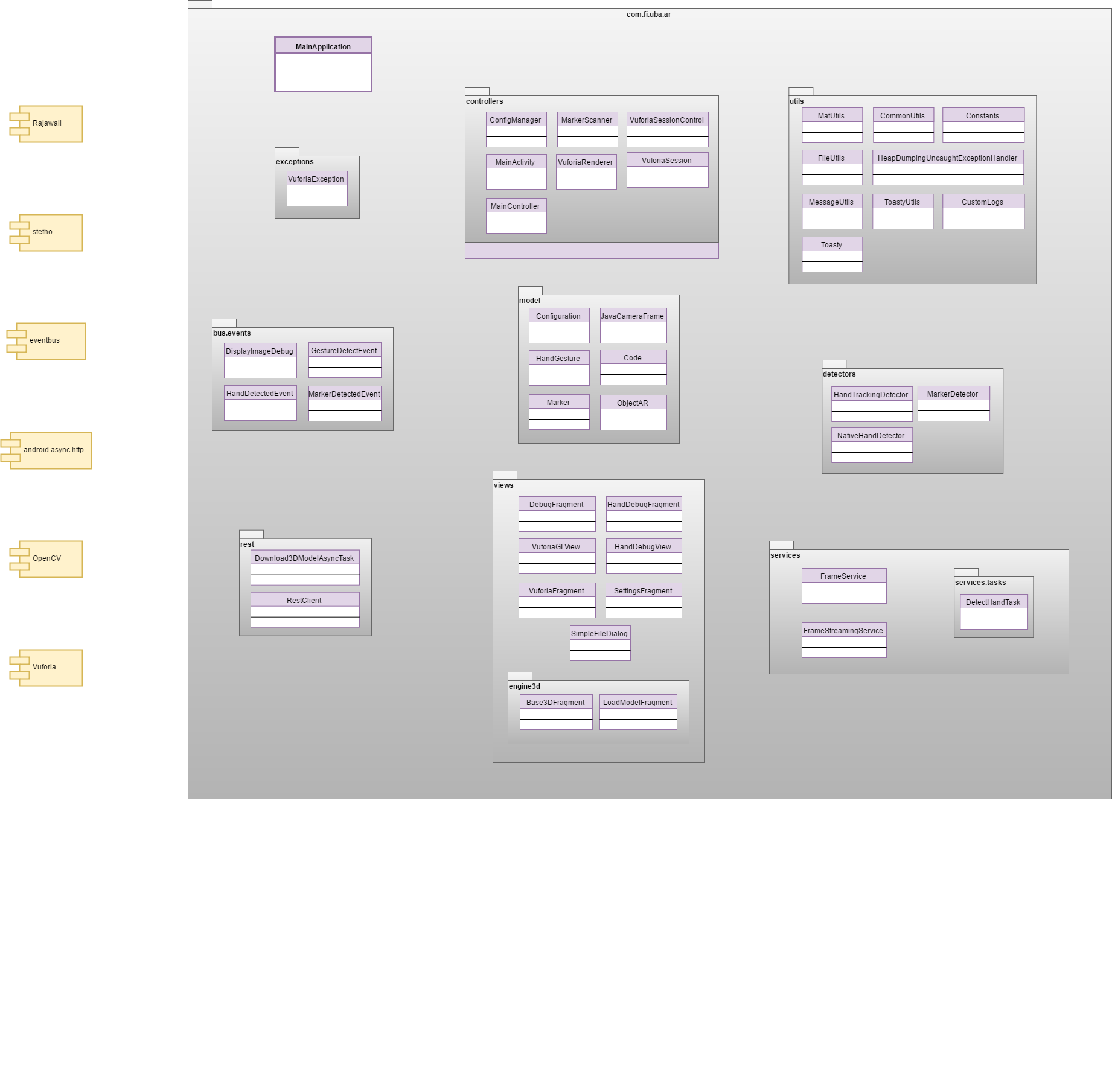
El código de la aplicación cliente se distribuye en los siguientes paquetes y librerías:



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.



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

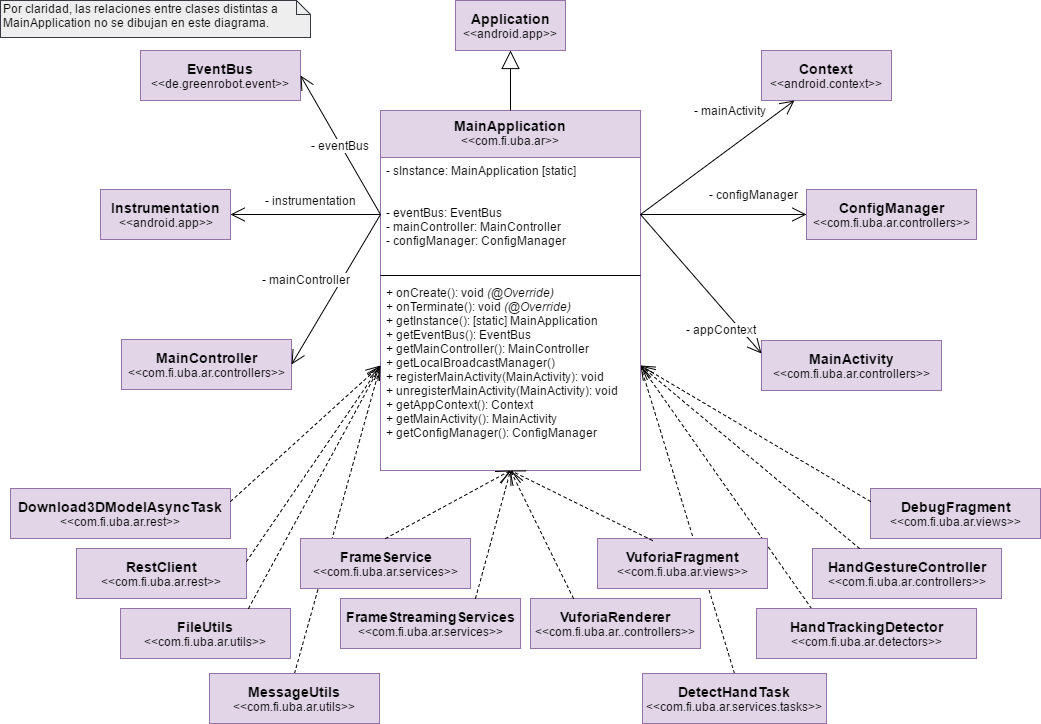


# 

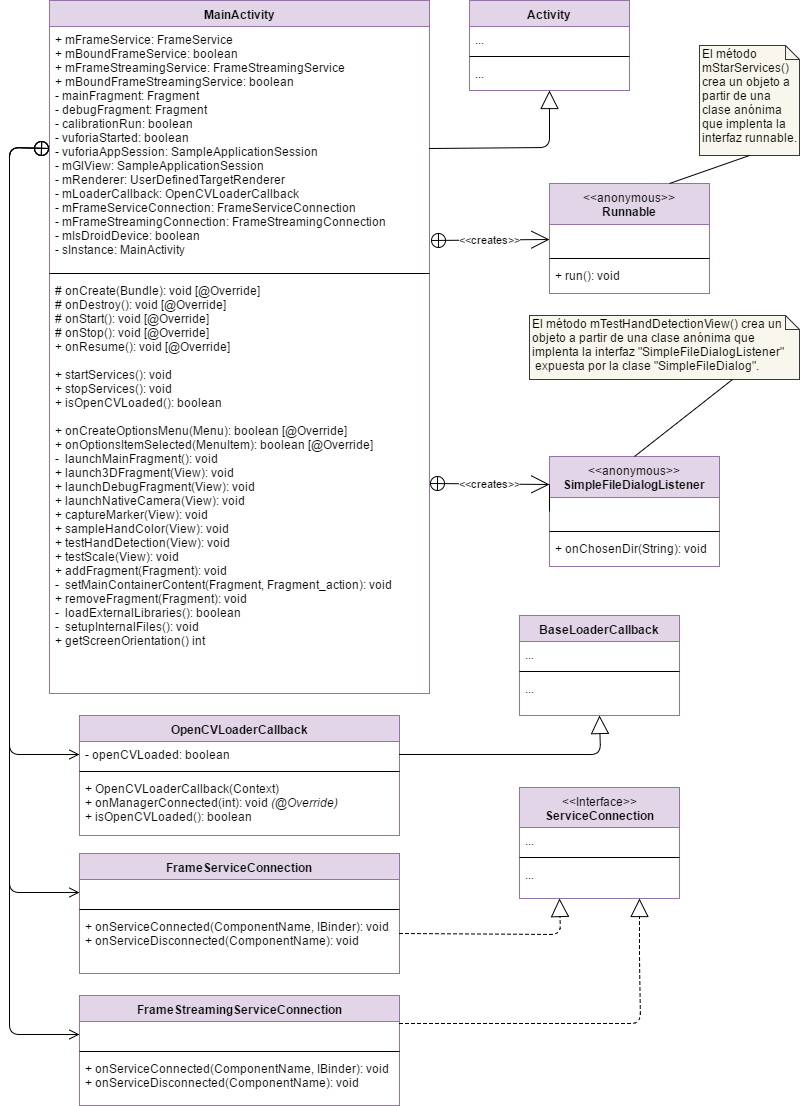
## Diagramas de Clases

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.

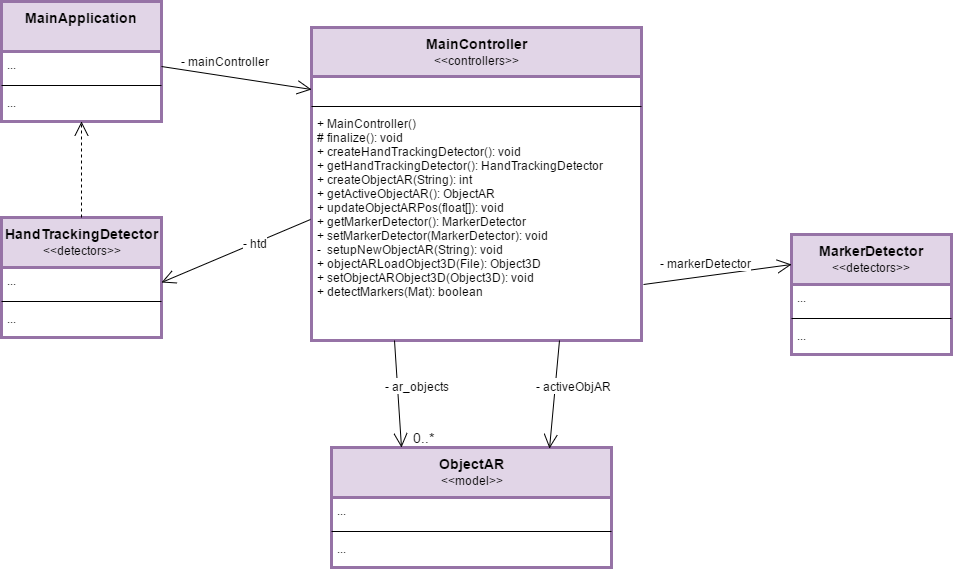
### Clase mainApplication



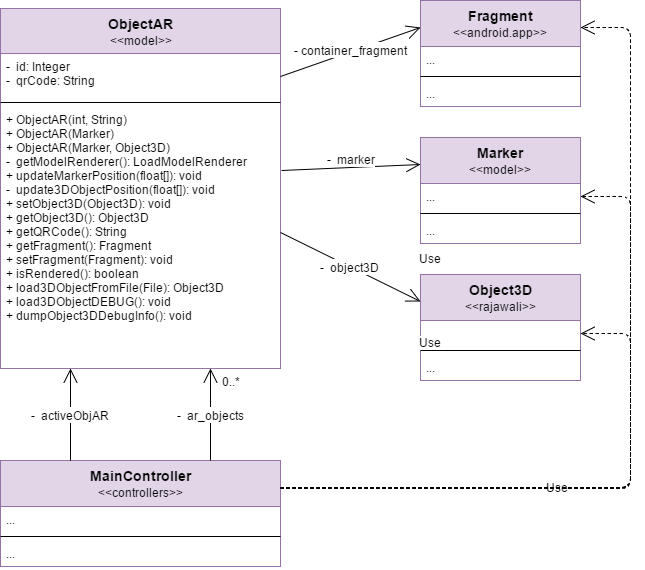
### Clase mainActivity



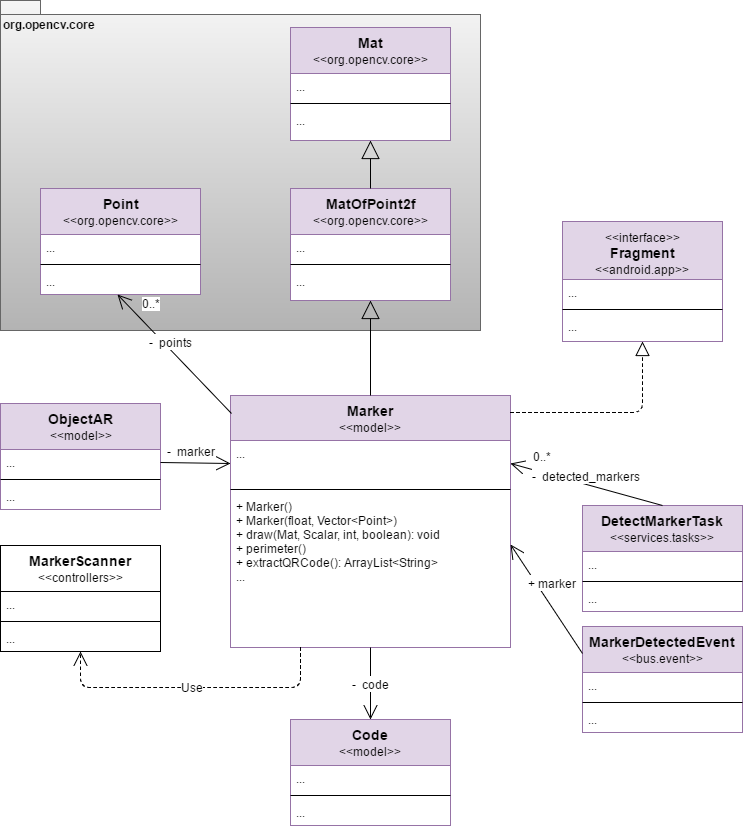
### Clase mainController



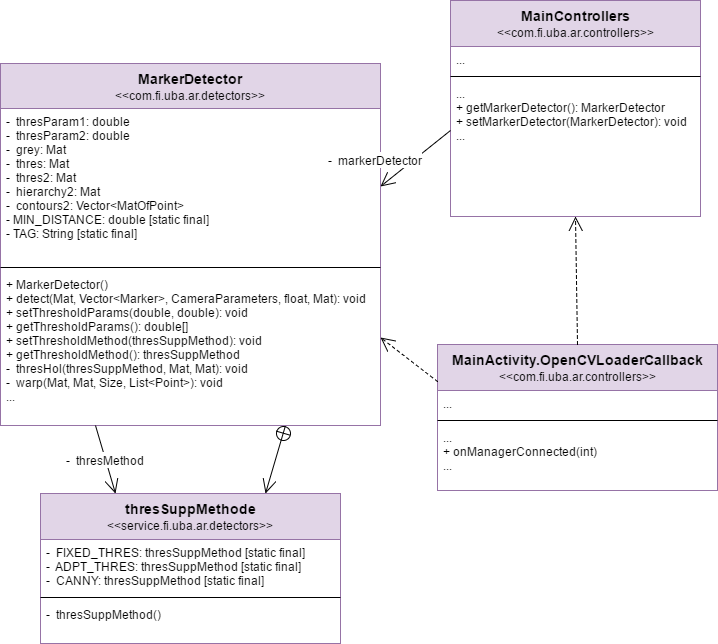
### Clase objectAR



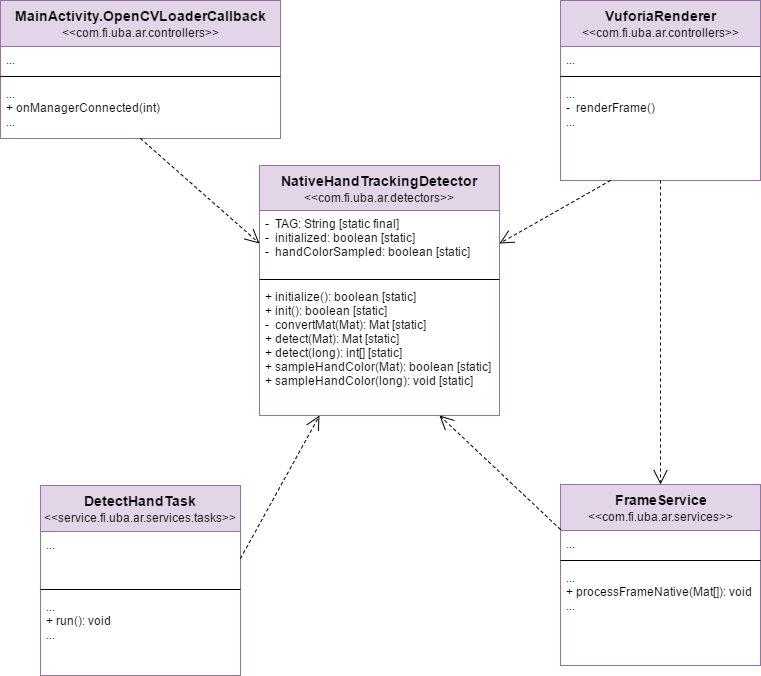
### Clase marker



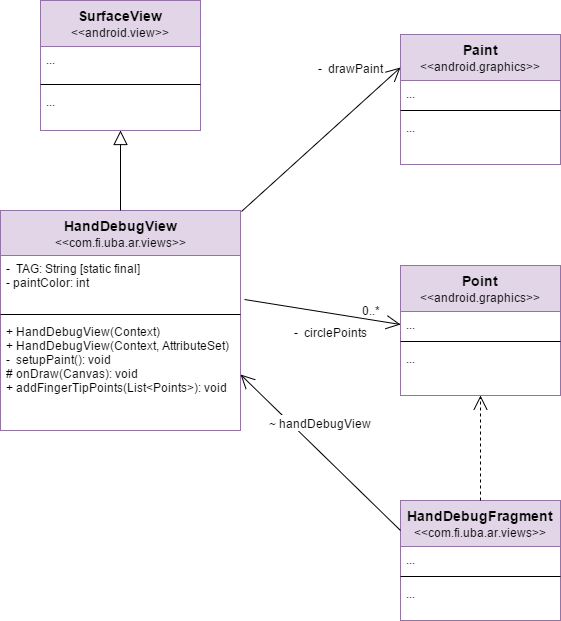
### Clase markerDetector



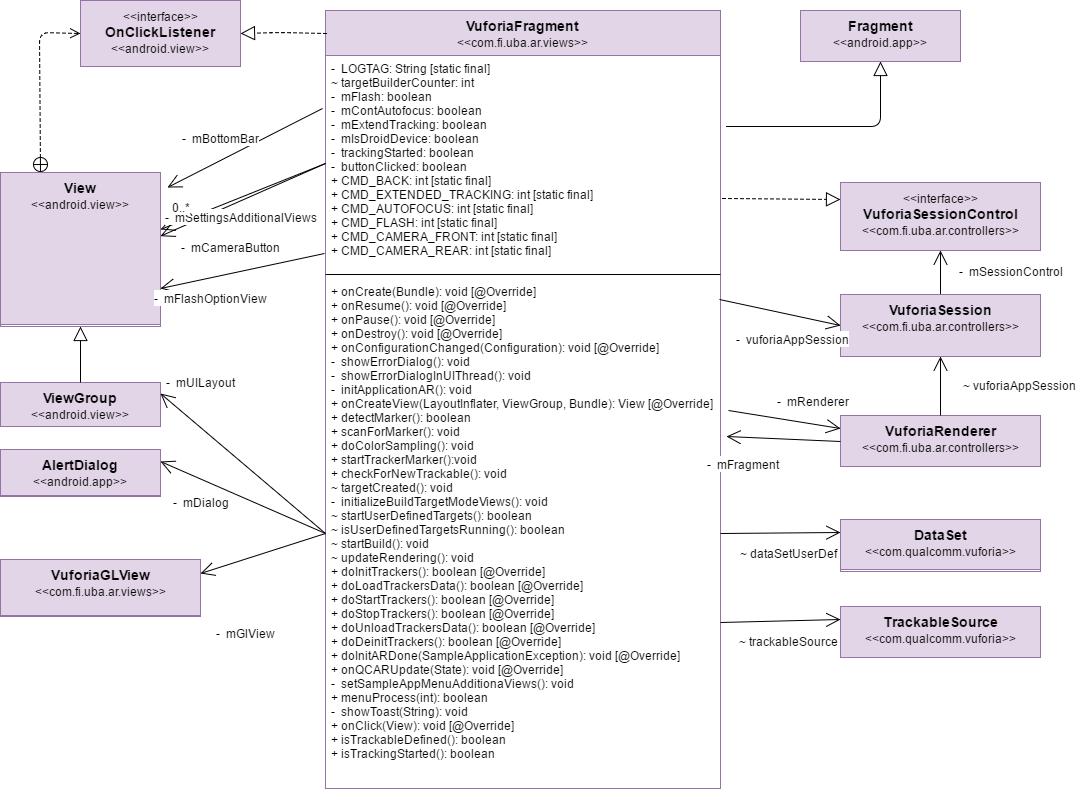
### Clase nativeHandTrackingDetector



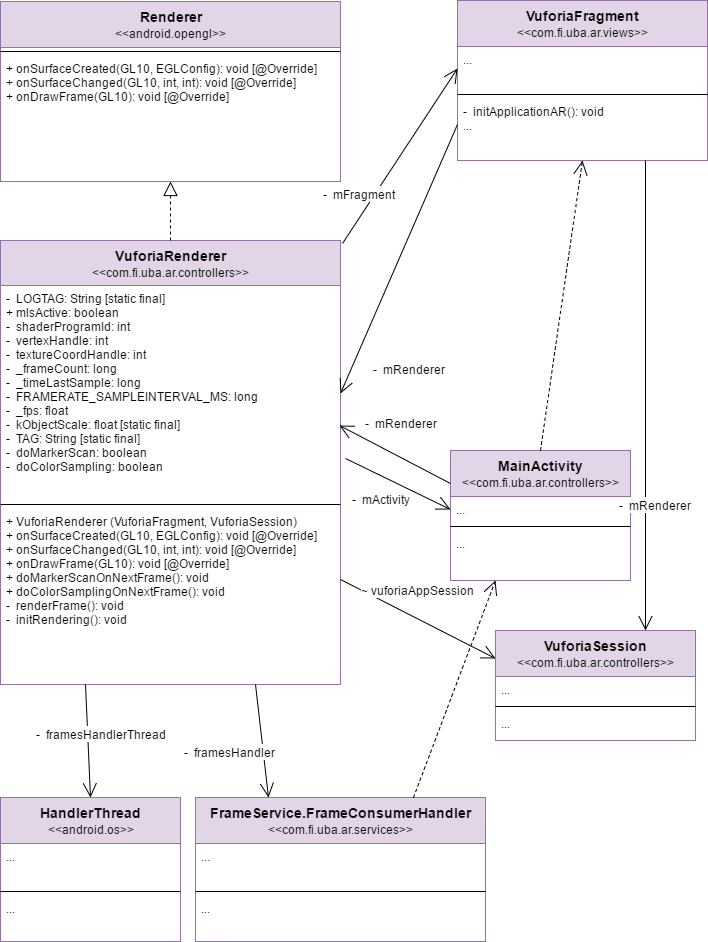
### Clase handDebugView



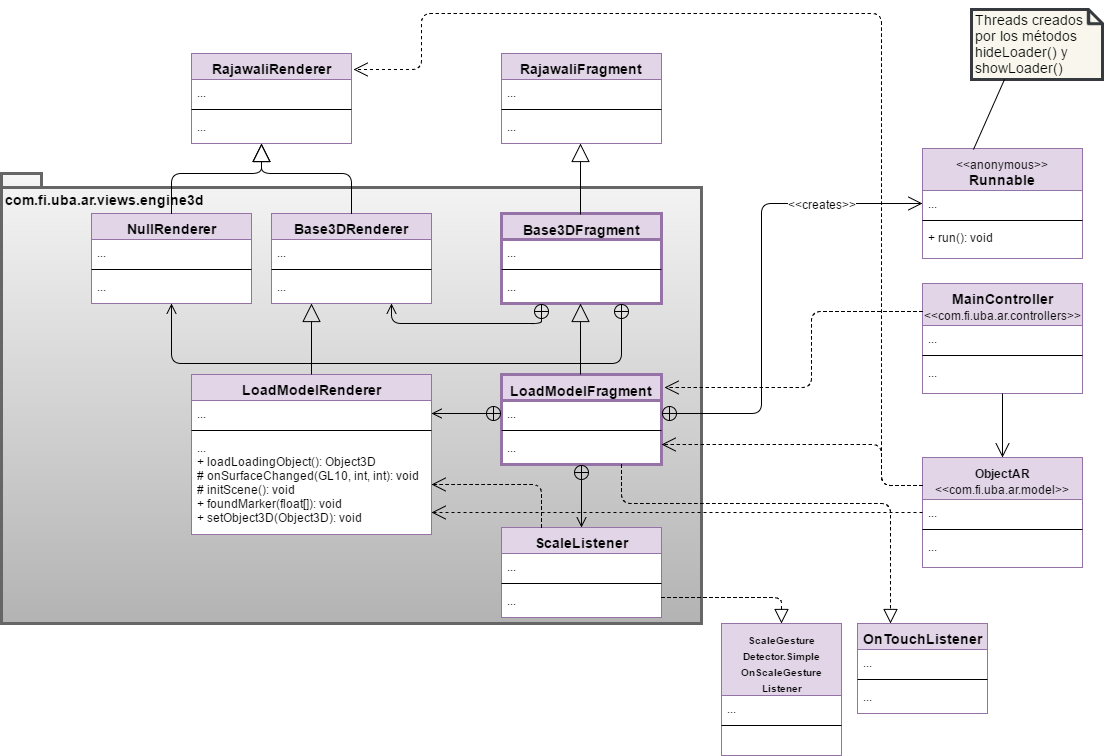
### Clase vuforiaFragment



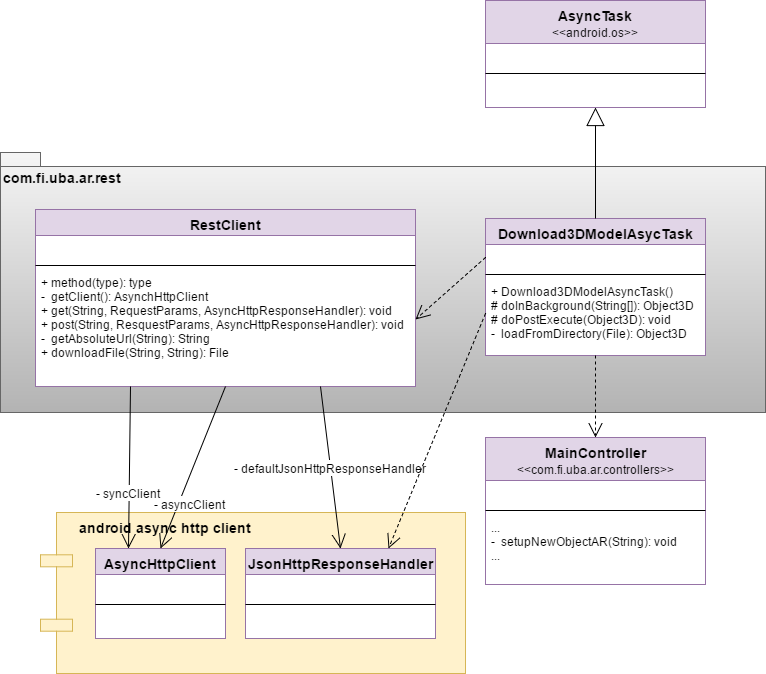
### Clase vuforiaRenderer



### Clases del paquete Engine3D: Base3DFragment y modelFragment



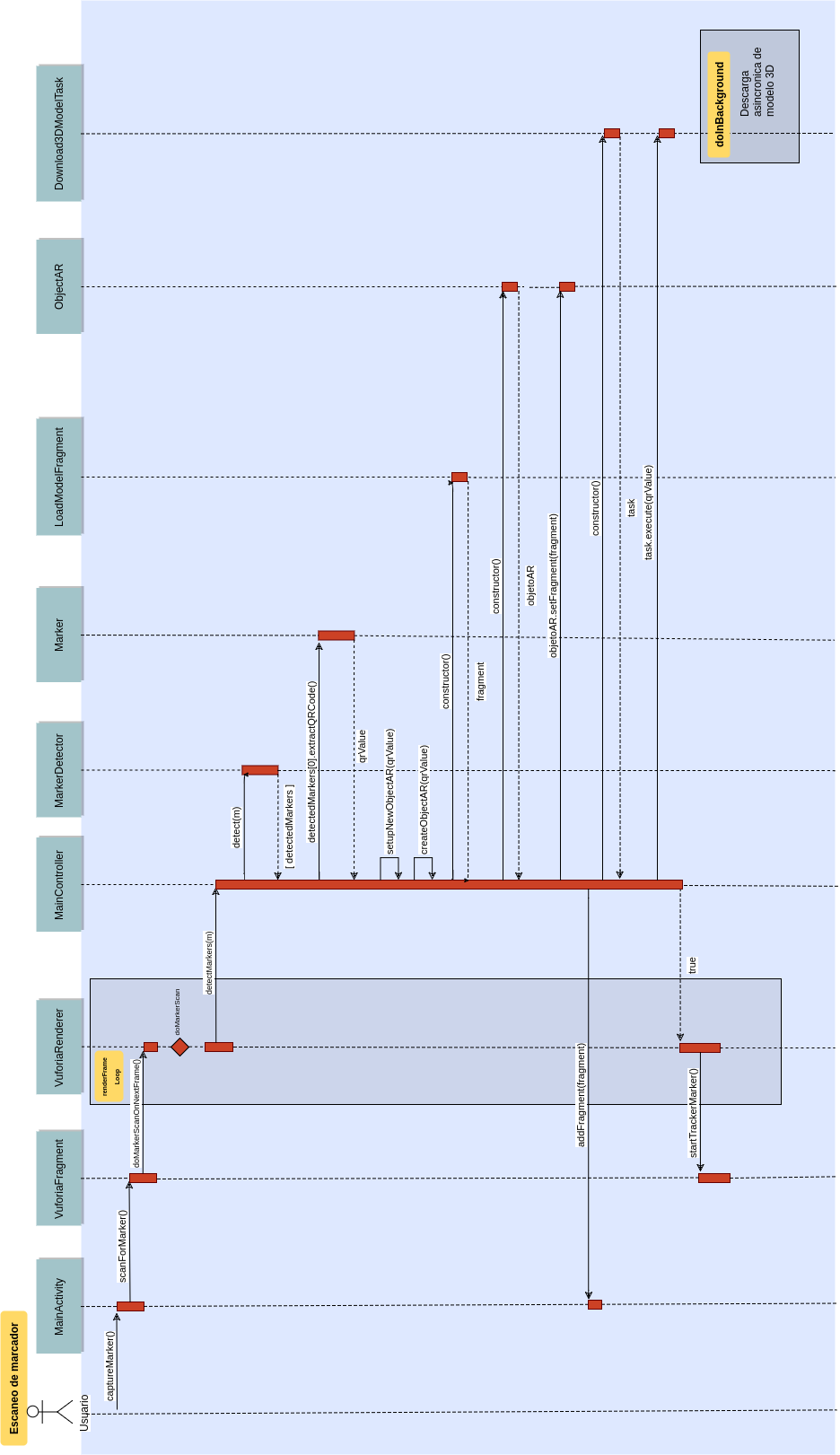
### Clases del paquete rest: RestClient y download3DModelAsyncTask



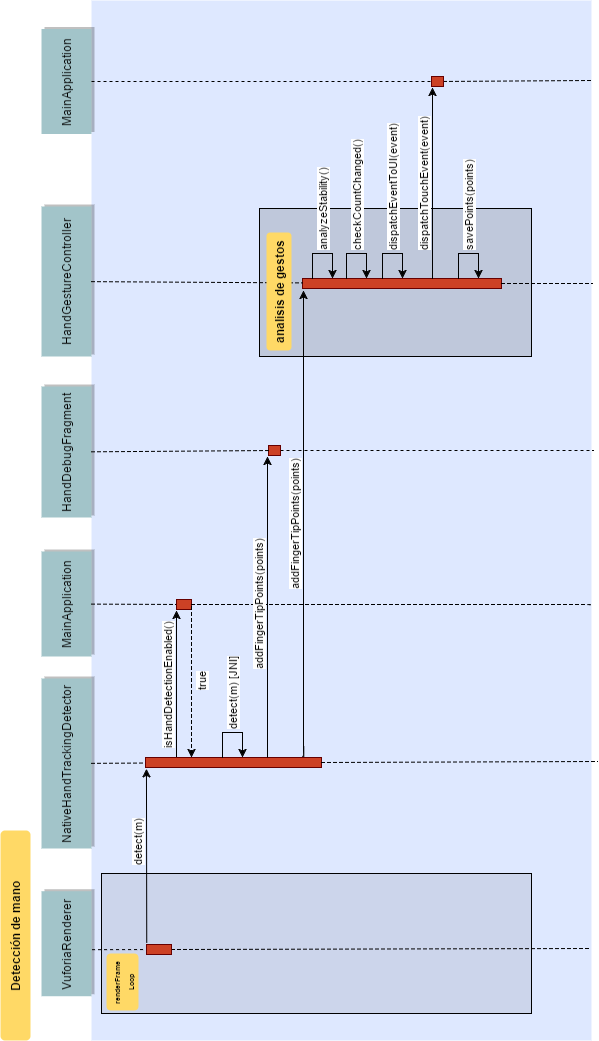
Es necesario señalar que estos diagramas representan a la Aplicación Cliente de FIUBAAR. Esta es la que representa el mayor esfuerzo de desarrollo, y por esta razón se muestra con todo detalle. Para entender la Aplicación FIUBAAR véase el diagrama de despliegue, en su sección específica.

## Diagramas de Secuencia

### Secuencia: Escaneo de un marcador

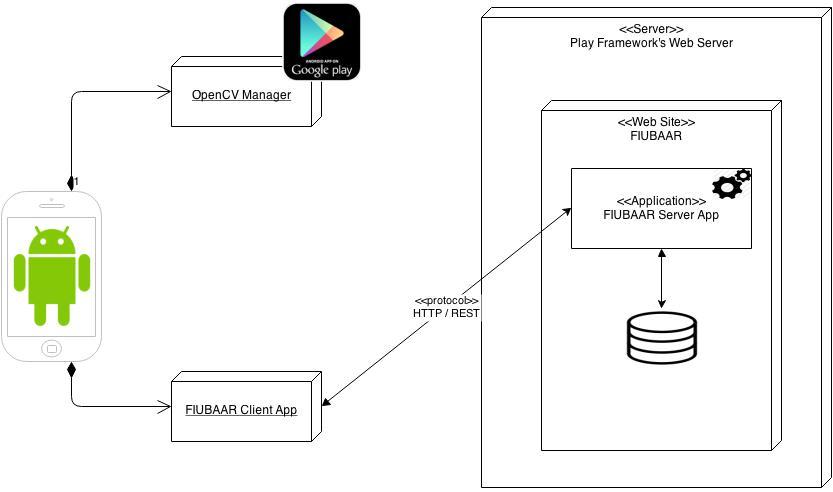


### Secuencia: Detección de Mano



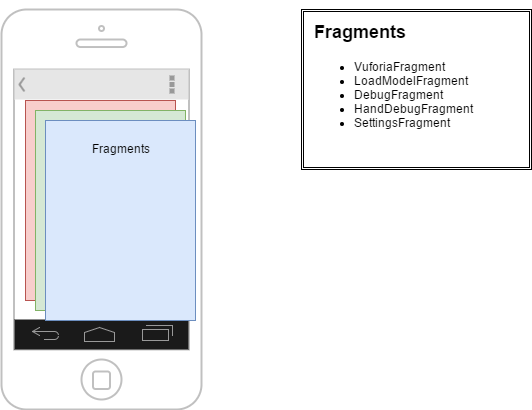
## Diagrama de Despliegue

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. Nos valdremos del Framework Play para implementar esta aplicación.

## Diseño de la Interfaz de Usuario

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.

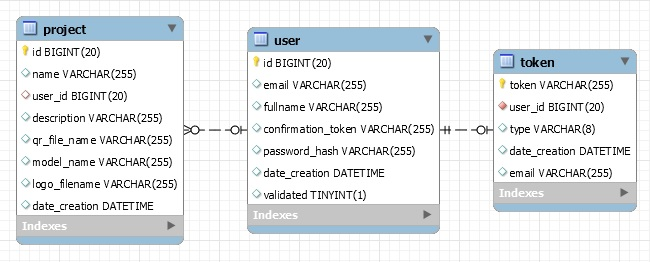


## Modelo de datos

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*).



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | ***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 | ***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 |

# 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 y fácilmente distinguible 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*), que no hacen uso de un marker, sino que realizan un seguimiento de cualquier tipo de imagen, sin necesidad de contar con características particulares o con 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

<http://www.artoolworks.com/support/library/Creating_and_training_new_ARToolKit_markers>

Existen herramientas que permiten generar nuevos markers con los patrones que uno desee

<http://flash.tarotaro.org/blog/2008/12/14/artoolkit-marker-generator-online-released/>

<http://www.arined.org/?page_id=6>

### Requerimientos FIUBAAR

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

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 (<http://www.zappar.com/>) y utiliza un marker que llaman Zapcode (<https://zapcode.it/>)

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 <https://zapcode.it/how-zapcodes-work/> y presentan una comparación con los códigos QR, explicando porque los zapcodes son -a su criterio- mejores <https://zapcode.it/zapcodes-vs-qr/>

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

<http://en.wikipedia.org/wiki/Barcode>

<http://en.wikipedia.org/wiki/Mobile_tagging>

### 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.

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

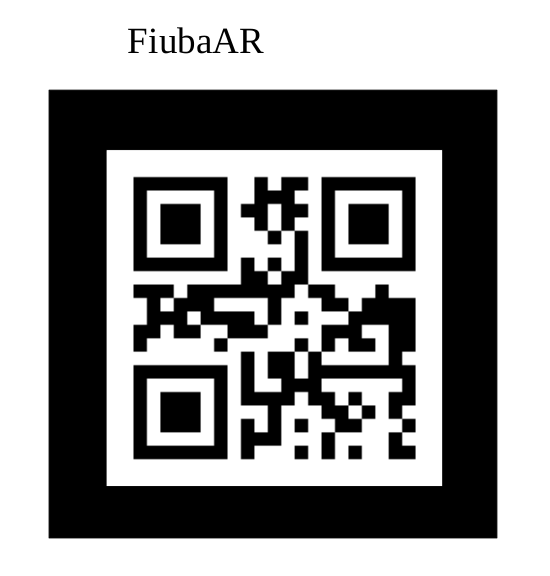
### Marker seleccionado

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:



### Documentación adicional útil

<http://qrlive.bluegouda.com/>

<https://github.com/zxing/zxing> (librería que reconoce varios tipos de código y posee una implementación para Android)

<http://dbis.eprints.uni-ulm.de/986/1/BA_Hartmann_13.pdf> tesis, cuyo código está disponible (<https://github.com/xamino/bachelor_framework_git>), en donde se implementa detección de marker que si bien no es QR, se implementa el reconocimiento.

<http://en.wikipedia.org/wiki/Fiduciary_marker>

Comparación de AR con marker y sin marker. Menciona otros formatos además de QR

<http://researchguides.dartmouth.edu/content.php?pid=227212&sid=1891183>

<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://huaizhiyazi.wordpress.com/2014/04/15/can-you-give-a-list-of-popular-android-libraries-that-are-commonly-used-to-build-applications/>

<http://dsynflo.blogspot.com.ar/2010/03/2d-barcodes-form-data-matrix-to-qr-code.html>

Se puede generar dinámicamente una imagen QR con Google charts con

[http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=|1&chl=TEXTO\_ACA](http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=%7C1&chl=TEXTO_ACA)

<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/>

<https://github.com/dm77/ZBarScanner>

## 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>

Esto es una librería en C++ que parece presentar la posibilidad de realizar una compilación para plataforma Android.

**AndroidARMarkerDetection:** Android port of Aruco

**Sitio:** <https://github.com/dszafir/AndroidARMarkerDetection>

Parece ser 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 se encuentra en

<https://github.com/xamino/bachelor_framework_git>

Marker Detection for Augmented Reality Applications

**Sitio:** <http://www.infi.nl/blog/view/id/56/Marker_Detection_for_Augmented_Reality_Applications>

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.

<http://iplimage.com/blog/cv-img-tec-black-white-marker-detection/>

Este artículo presenta la idea básica detrás de un algoritmo elemental para realizar la detección de markers con forma cuadrada.

<http://qtandopencv.blogspot.com.ar/2013/10/perspective-correction-for.html>

El artículo 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.

<http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects/>

Este artículo también explica cómo realizar la corrección de perspectiva, con una clara explicación del los pasos del algoritmo utilizado.

### Testing de implementaciones

#### Aruco

Compila directamente con todos los valores por defecto de proyecto descargado desde <http://sourceforge.net/projects/aruco/files/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.

Inicialmente, no logramos compilar el código para la plataforma Android y pero se estimó que quizás solo fuera cuestión de modificar archivos CMake.

Esta librería parece ser una muy buena opción estable para detectar y realizar tracking de markers, 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 <https://github.com/taka-no-me/android-cmake> ya que vemos algunos errores como esto <https://code.google.com/p/android-cmake/issues/detail?id=15>

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  # -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 excepto 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 markers definidos previamente para la aplicación FIUBAAR.

Encontramos una página que realiza una evaluación técnica de la librería

<http://iplimage.com/blog/aruco-technical-evaluation/>

#### 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 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 prácticamente, 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

Quizás estos threads ayuden a acomodar el código para no tener problemas con los asserts que causan el crash.

<http://answers.opencv.org/question/3023/solvepnp-assertion-failed/>

<https://github.com/kylemcdonald/ofxFaceTracker/issues/15>

#### Implementation of a Java Framework for Marker Based Detection in Augmented Reality

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.

#### Marker Detection for Augmented Reality Applications

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 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 (https://developer.vuforia.com/downloads/samples) 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 (<http://developer.vuforia.com/library/articles/FAQ/Watermark> ), el requerimiento de una clave para la aplicación generada en el portal de desarrollo (<http://developer.vuforia.com/library/articles/Training/Vuforia-License-Manager> ) 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 opensource.

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 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

<http://dsynflo.blogspot.com.ar/2010/06/libdecodeqr-libdecodeqr-is-cc-library.html>

<https://www.openhub.net/p/zxing/rss_articles?page=5>

<http://stackoverflow.com/questions/17973641/how-to-decode-data-using-zxing-c>

<http://wiki.ssrrsummerschool.org/doku.php?id=robocup2012:qrcode-cppexample>

<http://www.pixcl.com/QR_Code_Decoder.htm>

<http://www.pixcl.com/libdecodeqr_instability.htm>

<https://conra.dk/2013/01/06/a-quicker-qr-code-scanner.html>

<http://stackoverflow.com/questions/18638319/zbar-memory-leak-on-ios>

<http://developer.nokia.com/community/wiki/Qr_Decoder_in_Qt>

### Testing de implementaciones

<http://www.blackdogfoundry.com/blog/zbar-bar-code-qr-code-reader-android/>

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 liberados).

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, pero cual se adoptaría dependía de para cuál pudiera resolverse el problema de memoria que ambas presentaban, tal vez no por sí mismas sino por inclusión dentro del código de la aplicación en desarrollo.

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.

<https://github.com/MasDennis/Rajawali/wiki>

Integración de Rajawali con Vuforia SDK <https://github.com/MasDennis/RajawaliVuforia>

Integración de jPCT con Vuforia SDK <http://www.jpct.net/wiki/index.php/Integrating_JPCT-AE_with_Vuforia>

<https://github.com/sidneibjunior/vuforia-jpct>

jMonkeyEngine Android Integration

<http://hub.jmonkeyengine.org/wiki/doku.php/jme3:android>

<http://hub.jmonkeyengine.org/wiki/doku.php/jme3:eclipse_jme3_android_jnindk>

### Implementación seleccionada

De todas las implementaciones encontradas, se encontró que Rajawali es estable, compiló en el primer intento sin necesidad de introducir modificaciones en el código, es ampliamente utilizada (con numerosas menciones a través de distintos sitios) y es parte de muchos otros proyectos. Además, 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, que nos permitirá solo enfocarnos en la implementación de la lógica de consumo de los datos obtenidos.

### Implementaciones encontradas

**Nombre**: Android Asynchronous Http Client

**Sitio:** <http://loopj.com/android-async-http/>

**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.

<https://www.youtube.com/watch?v=xHXn3Kg2IQE>

<http://developer.android.com/training/volley/index.html>

<http://spring.io/guides/gs/consuming-rest-android/>

<http://pivotallabs.com/using-retrofit-rest-client-android-applications/>

<https://raw.githubusercontent.com/stephanenicolas/robospice/master/gfx/RoboSpice-InfoGraphics.png>

### 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 (http://www.appbrain.com/stats/libraries/details/loopj\_asynchttpclient/android-asynchronous-http-client).

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 pero 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 encontradas.

Spring es un framework 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.

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 ya que se realizan tareas intensivas constantemente y es necesario poder ejecutar en paralelo diferentes acciones. Además, habiendo sido esta librería elegida para importantes desarrollos como los casos de Pinterest, Instagram, MercadoLibre, etc. aporta mayor peso aún al momento de la elección.

## Investigación Hand & Finger Tracking

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/blog/2014/03/27/color-based-hand-gesture-recognition-on-android/>

Esta implementación parece ser la mejor alternativa: compila y ejecuta. El problema es que es muy lenta, a menos que se la configure con una resolución bien chica, de otro modo no se ve para nada como el video de demo, y hay una demora gigante en el video.

Está basada en la anterior (Implementación N° 01) ya que lo menciona en el post y cita el paper, pero esta portado todo a java y no lo hace con la implementación nativa en JNI. JNI está solo siendo usado para la librería de entrenamiento “SVM”.

Se estima que esta implementación podría ganar velocidad si usara código nativo completamente.

**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>

Desarrollo de características interesantes realizado por un estudiante de Japón.

**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> (no compiló)

**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>

### 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 tiempo de respuesta.

## Investigación sobre detección de gestos

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.- <https://eaglesky.github.io/blog/2014/03/27/color-based-hand-gesture-recognition-on-android/> (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: Deben 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:

<https://developer.android.com/training/gestures/detector.html>

<https://developer.android.com/training/gestures/scale.html>

<https://developer.android.com/training/gestures/multi.html>

<https://developer.android.com/reference/android/view/GestureDetector.html>

<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.

### Links

<https://depts.washington.edu/madlab/proj/dollar/index.html>

<https://github.com/asweigart/moosegesture>

<https://stackoverflow.com/questions/5867059/android-how-to-create-a-motionevent>

<https://stackoverflow.com/questions/4396059/how-to-simulate-a-touch-event-in-android>

<https://developer.android.com/training/gestures/scale.html>

<https://stackoverflow.com/questions/5250290/why-doesnt-this-motionevent-simulation-work>

<https://stackoverflow.com/questions/5240287/how-to-send-synthesized-motionevent-through-the-system>

[https://groups.google.com/forum/#!searchin/android-platform/simulate$20motionevent|sort:relevance/android-platform/e1mL3YT2uU8/LdkCesv4rxUJ](https://groups.google.com/forum/#!searchin/android-platform/simulate$20motionevent%7Csort:relevance/android-platform/e1mL3YT2uU8/LdkCesv4rxUJ)

<https://github.com/RobotiumTech/robotium/blob/b69dcf740baabec48c91999e523377faef79682e/robotium-solo/src/main/java/com/robotium/solo/Solo.java>

<https://www.softteco.com/blog/touch-hold-swipe-release-gesture-simulation-in-android-unit-test/>

<http://alvinalexander.com/java/jwarehouse/android/test-runner/src/android/test/TouchUtils.java.shtml>

<http://blogs.steeplesoft.com/posts/2013/simulating-swipes-in-your-android-tests.html>

<https://nandhanthiravia.blogspot.com.ar/2014/01/inject-multi-touch-event.html>

<http://www.pocketmagic.net/injecting-events-programatically-on-android/>

<http://www.vogella.com/tutorials/AndroidTouch/article.html>  < artículo destacado.

<https://stackoverflow.com/questions/29573026/dispatchtouchevent-in-fragment-in-android>

<https://github.com/Almeros/android-gesture-detectors>

<https://github.com/pwittchen/gesture>

<https://github.com/nisrulz/sensey>

<https://github.com/codepath/android_guides/wiki/Gestures-and-Touch-Events>

<https://medium.com/@ali.muzaffar/android-detecting-a-pinch-gesture-64a0a0ed4b41>

<https://blahti.wordpress.com/2011/01/17/moving-views-part-2/>

<https://stackoverflow.com/questions/11523423/how-to-generate-zoom-pinch-gesture-for-testing-for-android>

<https://stackoverflow.com/questions/3637044/generating-multitouch-motionevents-for-testing>

<https://stackoverflow.com/questions/39582808/android-motionevent-pointer-index-confusion>

### Implementación seleccionada

Para mantener la estructura del documento y como adicional a todo el análisis anteriormente expuesto, se destaca la implementación seleccionada es la primera de las listadas: una implementación nativa que muestra un mejor desempeño que su alternativa.

# 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** (version 7.x o superior) [[11]](#footnote-11)
* **Android ADT Bundle** (contiene Android SDK)[[12]](#footnote-12)
* **Android Native Development Kit** (NDK)[[13]](#footnote-13)
* **OpenCV for Android** - (android-sdk)[[14]](#footnote-14)
* **Activator [[15]](#footnote-15)**

## Configuración

Para la correcta configuración se deben seguir los siguientes pasos:

Proceder a instalar en primer lugar Java JDK.

Luego se debe descomprimir el archivo de Android ADT Bundle siguiendo los pasos descriptos en la guía suministrada por Google Android [19].

Descomprimir Android NDK

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 en nuestros proyectos: <http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html>

<http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html> [9/Marzo/2017]

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.

Rajawali: 3DEngine. <https://github.com/MasDennis/Rajawali>

<https://github.com/Rajawali/Rajawali> [9/Marzo/2017]

**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 (

<http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_dev_intro.html#android-dev-intro> )( <http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/android_dev_intro.html#android-dev-intro> [9/Marzo/2017])

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 [20].

En Linux se puede ejecutar en una terminal

export OPENCV\_HOME="<path>/OpenCV-2.4.9-android-sdk"

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 [21] (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](mailto: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 [22]. Para Windows se recomienda usar msysgit (<http://msysgit.github.io/>), actualmente llamado “git for Windows” [23].
* Aplicación para diagramas UML: Descargar Modelio desde su sitio <http://www.modelio.org/>. Esta herramienta está basada en Eclipse, así que simplemente se descomprime y se ejecuta. Otra opción consiste en recurrir a una herramienta web como <https://www.draw.io/>
* 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 <http://tools.android.com/tech-docs/new-build-system/user-guide>
* GenyMotion (<https://cloud.genymotion.com/page/doc/>): 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.

## 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.

## 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. [[9]](http://javarevisited.blogspot.com.ar/2011/08/code-comments-java-best-practices.html)[[10]](http://www.hongkiat.com/blog/source-code-comment-styling-tips/)[[11]](http://java.dzone.com/articles/5-best-practices-commenting)

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. [[10]](http://stackoverflow.com/questions/1777175/how-can-i-generate-javadoc-comments-in-eclipse?rq=1)[[11]](http://www.jetbrains.com/idea/features/javadoc.html#link6)

### Desarrollo en Android

<http://developer.android.com/guide/practices/index.html>

<http://developer.android.com/training/index.html> (contiene buenas prácticas para algunos cuantos aspectos del desarrollo para Android)

<http://www.slideshare.net/retomeier/being-epic-best-practices-for-building-android-apps>

<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” (<https://www.atlassian.com/git/workflows#!workflow-gitflow>), 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 siguientes tutoriales:

<https://www.atlassian.com/git/tutorial>

<https://confluence.atlassian.com/display/BITBUCKET/Using+Git+branches>

## Referencias

[1] <https://developer.android.com/about/dashboards/index.html>

# Apéndice II - Bibliografía

Se lista a continuación la bibliografía consultada, en el orden de aparición en el documento.

|  |  |
| --- | --- |
| [1] | Google Android, «JNI Tips,» [En línea]. Available: https://developer.android.com/training/articles/perf-jni.html. [Último acceso: 9 Marzo 2017]. |
| [2] | Google Android, «Aspectos fundamentales de la aplicación,» [En línea]. Available: https://developer.android.com/guide/components/fundamentals.html. [Último acceso: 9 Marzo 2017]. |
| [3] | Tutorials Point, «Android Architecture,» [En línea]. Available: http://www.tutorialspoint.com/android/android\_architecture.htm. [Último acceso: 9 Marzo 2017]. |
| [4] | Google Android, «Procesos y subprocesos,» [En línea]. Available: https://developer.android.com/guide/components/processes-and-threads.html. [Último acceso: 9 Marzo 2017]. |
| [5] | Stack Overflow, «MVC Pattern in Android ?,» [En línea]. Available: http://stackoverflow.com/questions/2925054/mvc-pattern-in-android. [Último acceso: 9 Marzo 2017]. |
| [6] | J. Musselwhite, «Android Architecture: Part 10, The Activity Revisited,» [En línea]. Available: http://www.therealjoshua.com/2012/07/android-architecture-part-10-the-activity-revisited/. [Último acceso: 9 Marzo 2017]. |
| [7] | MindTheRobot - Android Dev Insight, «Android Architecture: Message-based MVC,» [En línea]. Available: http://mindtherobot.com/blog/675/android-architecture-message-based-mvc/. [Último acceso: 9 Marzo 2017]. |
| [8] | K. Sokolova, M. Lemercier y L. Garcia, «Android Passive MVC,» [En línea]. Available: http://www.thinkmind.org/download.php?articleid=patterns\_2013\_1\_20\_70039. [Último acceso: 9 Marzo 2017]. |
| [9] | Stack Overflow, «Difference between asp.net MVC and MVP? Are they both same?,» [En línea]. Available: http://stackoverflow.com/questions/19996963/difference-between-asp-net-mvc-and-mvp-are-they-both-same. [Último acceso: 9 Marzo 2017]. |
| [10] | Google Android, «Servicios,» [En línea]. Available: https://developer.android.com/guide/components/services.html. [Último acceso: 9 Marzo 2017]. |
| [11] | Google Android, «Class Service,» [En línea]. Available: https://developer.android.com/reference/android/app/Service.html. [Último acceso: 9 Marzo 2017]. |
| [12] | Stack Overflow, «Android Service: Process vs. Not,» [En línea]. Available: http://stackoverflow.com/questions/4083756/android-service-process-vs-not. [Último acceso: 9 Marzo 2017]. |
| [13] | Stack Overflow, «How to have Android Service communicate with Activity,» [En línea]. Available: http://stackoverflow.com/questions/2463175/how-to-have-android-service-communicate-with-activity. [Último acceso: 9 Marzo 2017]. |
| [14] | Stack Overflow, «Example: Communication between Activity and Service using Messaging,» [En línea]. Available: http://stackoverflow.com/questions/4300291/example-communication-between-activity-and-service-using-messaging. [Último acceso: 9 Marzo 2017]. |
| [15] | Android Developers Blog, «Multitasking the Android Way,» [En línea]. Available: https://android-developers.googleblog.com/2010/04/multitasking-android-way.html. [Último acceso: 9 Marzo 2017]. |
| [16] | Mind The Robot - Android Dev Insight, «Android Architecture Tutorial: Developing an App with a Background Service (using IPC),» [En línea]. Available: http://mindtherobot.com/blog/37/android-architecture-tutorial-developing-an-app-with-a-background-service-using-ipc/. [Último acceso: 9 Marzo 2017]. |
| [17] | Techotopia, «Android Remote Bound Services – A Worked Example,» [En línea]. Available: http://www.techotopia.com/index.php/Android\_Remote\_Bound\_Services\_%E2%80%93\_A\_Worked\_Example. [Último acceso: 9 Marzo 2017]. |
| [18] | Stack Overflow, «Service vs IntentService,» [En línea]. Available: http://stackoverflow.com/questions/15524280/service-vs-intentservice. [Último acceso: 9 Marzo 2017]. |
| [19] | Google Android, «Installing the Eclipse ADT Bundle,» [En línea]. Available: http://web.archive.org/web/20140828071826/http://developer.android.com/sdk/installing/index.html?pkg=adt. [Último acceso: 9 Marzo 2017]. |
| [20] | Computer Hope, «How to set the path and environment variables in Windows,» [En línea]. Available: http://www.computerhope.com/issues/ch000549.htm. [Último acceso: 9 Marzo 2017]. |
| [21] | Google Android, «Using Hardware Devices,» [En línea]. Available: http://web.archive.org/web/20141103185158/https://developer.android.com/tools/device.html. [Último acceso: 9 Marzo 2017]. |
| [22] | Eclipse, «EGit,» [En línea]. Available: http://www.eclipse.org/egit/. [Último acceso: 9 Marzo 2017]. |
| [23] | git for windows, «git for windows,» [En línea]. Available: https://git-for-windows.github.io/. [Último acceso: 9 Marzo 2017]. |

1. Como referencia visitar <http://developer.android.com/training/articles/perf-jni.html>. [↑](#footnote-ref-1)
2. Consultar <https://developer.android.com/guide/components/fundamentals.html>. [↑](#footnote-ref-2)
3. Para mayor detalle, <http://www.tutorialspoint.com/android/android_architecture.htm>. [↑](#footnote-ref-3)
4. Consultar <https://developer.android.com/guide/components/processes-and-threads.html>. [↑](#footnote-ref-4)
5. Para habilitar AOT deben seguirse los siguientes pasos: Settings 🡪 Developer Options 🡪 Select Runtime 🡪 Use ART. [↑](#footnote-ref-5)
6. <http://stackoverflow.com/questions/2925054/mvc-pattern-in-android>. [↑](#footnote-ref-6)
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>. [↑](#footnote-ref-7)
8. Consultar <http://stackoverflow.com/questions/19996963/difference-between-asp-net-mvc-and-mvp-are-they-both-same>. [↑](#footnote-ref-8)
9. Consultar <https://developer.android.com/guide/components/services.html>, <https://developer.android.com/reference/android/app/Service.html>. [↑](#footnote-ref-9)
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>. [↑](#footnote-ref-10)
11. 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> Ultimo acceso 9/Marzo/2017 [↑](#footnote-ref-11)
12. 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> Ultimo acceso 9/Marzo/2017 [↑](#footnote-ref-12)
13. 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> Ultimo acceso 9/Marzo/2017 [↑](#footnote-ref-13)
14. Disponible en http://sourceforge.net/projects/opencvlibrary/files/opencv-android/ [9/Marzo/2017 – Versión 3.2.0] Ultimo acceso 9/Marzo/2017 [↑](#footnote-ref-14)
15. https://www.typesafe.com/activator/download

    https://www.lightbend.com/activator/download [9/Marzo/2017] [↑](#footnote-ref-15)