BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
Facultad de Ciencias de la Computación

Aplicación móvil para mostrar sitios turísticos de la ciudad de Puebla empleando realidad aumentada y geolocalización.

TESIS

Para obtener el grado de

Ingeniero en Ciencias de la Computación

Presenta
Jonathan García Rosas

ASESOR
M.C. Rafael de la Rosa Flores

Mayo de 2014

Índice

Índice de figuras

Introducción

El uso de Smartphones entre la población cada día va en aumento. Se estima que al día se activan cerca de un millón de dispositivos tan solo con el sistema operativo Android. En México, de acuerdo con estudios realizados por la firma de consultoría The Competitive Intelligence Unit (CIU), para el año 2015, 7 de cada 10 mexicanos tendrán un Smartphone [13]. En los últimos años, la limitante en cuanto a calidad y precio que determinaba la adquisición de un Smartphone se ha visto reducida gracias al abaratamiento de los costos de dispositivos móviles con una mejora de prestaciones respecto a años anteriores, lo que implica una creciente demanda de servicios que los usuarios pueden consumir, y que ha dado paso al desarrollo profesional de aplicaciones como una fuente de ingresos para desarrolladores que se consolida en las plataformas de venta, como App Store y Market Place, en las cuales, tan solo en el año 2012, dejaron ganancias por cerca de $30,000,000 de dólares[20].

Las aplicaciones móviles que se ofertan actualmente en las tiendas online están catalogadas en diversas categorías que van desde el entretenimiento, productividad, medicina, turismo y estilos de vida, las cuales aprovechan los recursos disponibles de un Smartphone para comunicarse por Internet, solicitar información a través de servicios web, enviar mensajes, aprovechar los servicios de navegación satelital, reproducir contenidos multimedia como música y videos, ejecutar videojuegos, entre otros usos. Algunas aplicaciones realizan tareas de procesamiento complejo que hace años solo podían realizar computadoras de escritorio, y que ahora los Smartphones pueden ejecutar gracias a sus componente internos como procesadores, memorias de mayor capacidad y la ayuda de sensores como el acelerómetro, el giroscopio o chips GPS, factores que hacen que un Smartphone sea cada vez más útil para tareas que exigen un mayor rendimiento y que, en algunos años, se espera que sean estos los que sustituyan el modelo tradicional de trabajo por computadoras de escritorio. Existen ciertas aplicaciones que necesitan un mayor poder de procesamiento para ejecutar tareas de manera fluida. Para este trabajo de tesis, se plantea el desarrollo de una aplicación móvil que use realidad aumentada para mostrar sitios turísticos de la ciudad de Puebla, además de usar servicios de localización para ubicarlos dichos sitios en un mapa.

La realidad aumentada es una técnica usada para combinar el mundo real con elementos virtuales en un dispositivo donde actúa primordialmente una cámara. Además de estar ligada a dispositivos especiales que simulan objetos virtuales, la realidad aumentada se puede ver en aplicaciones de computadora y, gracias a la proliferación de dispositivos móviles, se puede ver también en aplicaciones que hacen uso de sensores del dispositivo para dar una mayor muestra de convencimiento a los usuarios que hacen uso de ella. Para el desarrollo de aplicaciones con realidad aumentada existen problemas diversos que varían de complejidad de acuerdo a las funciones que la aplicación utilice. La mayor parte de aplicaciones de realidad aumentada en móviles están enfocadas al uso de marcadores artificiales creados por desarrolladores o los mismos usuarios, por lo cual, estas aplicaciones solo detectan aquellos marcadores predefinidos. Un problema común que se presenta es que el uso de marcadores en exteriores consiste en un método invasivo al colocarlos en el lugar deseado, esto para que el usuario lo enfoque y despliegue la información que el marcador contenga.

Este trabajo de tesis está orientado al desarrollo de una aplicación para dispositivos móviles con el sistema operativo Android, cuya finalidad principal es usar realidad aumentada y geolocalización para mostrar sitios de interés en la ciudad de Puebla, para ello se emplean diversas herramientas de desarrollo de Software, comenzando por una metodología de desarrollo ágil como lo es Scrum, la

cual es usada con prácticas de modelos iterativos e incrementales para desarrollar incrementos totalmente operativos por intervalos de tiempo cortos denominados Sprints. Se usan entornos de desarrollo de Software libres para producir la aplicación móvil, simuladores para pruebas así como algunas herramientas privativas como el caso de las librerías de realidad aumentada empleadas para este proyecto. Además, se contempla probar el producto final en un entorno operativo y en dispositivos móviles de diversas gamas y características de hardware.

La estructura de este documento se enumera a continuación:

Capítulo 1. Marco teórico. En este capítulo se analizan todos los recursos necesarios para desarrollar el proyecto tomando en cuenta las tecnologías que existen actualmente y que se aprovechan para este trabajo de tesis.

Capítulo 2. Estado del arte. Aquí se mencionan los sistemas que actualmente están en el mercado o que fueron desarrollados para fines similares a los de este trabajo.

Capítulo 3. Análisis. Se hace un análisis a detalle de los procesos necesarios para que la aplicación de realidad aumentada se desempeñe de forma correcta.

Capítulo 4. Diseño del Sistema. En este capítulo se describen todos los bloques y se diseñan las clases de cada bloque mediante el uso de capas. Por cada clase se da una breve descripción de su funcionamiento.

Capítulo 5. Implementación del sistema. Aquí se describe la implementación de cada uno de los bloques del sistema y se describen a detalle cada una de las clases desarrolladas, además de describir la interfaz con el usuario y la interacción del sistema de acuerdo a sus capas en cada bloque.

Capítulo 6. Instalación y Pruebas del Sistema. En este capítulo se explican los requerimientos indispensables para que el sistema funcione y se realizan diversas pruebas en dispositivos reales en algunos lugares de la capital de Puebla.

Conclusiones. Se presentan las conclusiones finales.

1. Marco teórico

En la actualidad los dispositivos móviles cuentas con características que permiten realizar actividades semejantes a una computadora y ofrecen conectividad con internet. Los Smartphones o teléfonos inteligentes son usados en áreas que van desde el entretenimiento hasta la investigación, ofreciendo la posibilidad de desarrollar aplicaciones que se ejecuten en estos dispositivos. Gran parte de las aplicaciones ofrecidas por las tiendas online están desarrolladas para el entretenimiento, siendo tan solo una pequeña porción las que se ofrecen con realidad aumentada. A continuación se describen los conceptos más usados en aplicaciones móviles y las tecnologías que existen actualmente para el desarrollo de realidad aumentada, además de explicar las tecnologías en cuanto a conectividad para geolocalización y servicios que se ofrecen en Internet.

1.1.1 Sistemas operativos móviles

Los sistemas operativos móviles más usados hoy en día son Android, iOs y Windows Phone, que abarcan una cuota de mercado importante.

Android. El sistema operativo Android fue desarrollado por la empresa Android Inc. Está basado en Linux y fue pensado para operar en celulares con pantallas táctiles. La empresa fue adquirida en 2005 por Google, presentando el primer móvil con Android en el año 2008. El sistema operativo es usado en computadoras personales, televisiones, tabletas etc. Una ventaja que ofrecen con respecto a sus competidores (iOS, Windows Phone) es el acceso a su código fuente por parte de desarrolladores.

IOS. El sistema operativo iOS fue desarrollado por Apple originalmente para el iPhone (iPhone OS) y siendo usado posteriormente en otros dispositivos de la firma (iPod, iPad y el Apple TV). La interfaz de usuario está basada en el concepto de manipulación directa usando gestos multitáctiles. IOS deriva directamente de Mac OS X, el cual está basado en Darwin BSD. Cuenta con un kit de desarrollo de Software para uso exclusivo en el entorno de desarrollo Xcode bajo sistemas OS X.

Windows Phone. Windows Phone es un sistema operativo móvil creado por Microsoft que es la versión posterior al sistema Windows Mobile. A diferencia de Windows Mobile que estaba orientado al mercado de los negocios, Windows Phone está más enfocado al mercado empresarial y de consumo de aplicaciones. Compite directamente con los sistemas operativos iOs de Apple y Android de Google. Uno de los esquemas innovadores de este sistema operativo es la interfaz metro, en la que se cambia radicalmente la tendencia de los sistemas operativos móviles de usar íconos por el uso de listas de mosaicos dinámicos, los cuales muestran información útil y características personalizadas por el usuario.

1.1.2 Sensores

Actualmente los dispositivos móviles de última generación cuentan con múltiples sensores que realizan numerosas tareas, tanto del sistema como para uso en aplicaciones. A continuación se listan los principales sensores de un Smartphone:

Acelerómetro. Es un dispositivo transductor que detecta el movimiento o el giro, es decir, es capaz de responder con una señal eléctrica ante una perturbación inducida por la aplicación de una fuerza

o la gravedad. Un acelerómetro ocupa tres ejes para medir un espacio tridimensional. En la figura 1.1 se observan los ejes del acelerómetro en un dispositivo móvil.

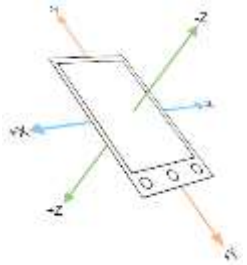

Figura 1.1 Ejes x, y y z del acelerómetro.

Giroscopio. El giróscopo o giroscopio (Figura 1.2) es un dispositivo mecánico que sirve para medir, mantener o cambiar la orientación en el espacio de algún aparato o vehículo [1]. Permite que un teléfono inteligente mida y mantenga la orientación. Los sensores giroscópicos pueden monitorear y controlar posiciones del dispositivo como la orientación, la dirección, el movimiento angular y la rotación. Cuando se aplica a un teléfono inteligente, un sensor giroscópico comúnmente lleva a cabo funciones de reconocimiento de gestos. Además, los giroscopios en los teléfonos inteligentes ayudan a determinar la posición y orientación del teléfono.

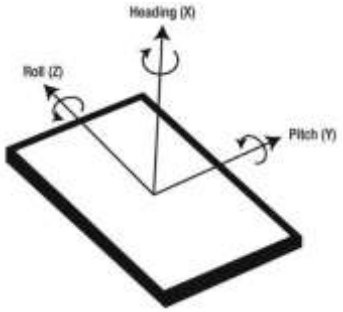

Figura 1.2 Ejes del dispositivo.

Sensor infrarrojo. Sirve para verificar la proximidad de un objeto desde el teléfono. Generalmente es de poco alcance y es utilizado en gran parte para verificar el objeto superior a la pantalla del móvil mientras se realice una llamada.

Magnetómetro. Mide la magnitud magnética de los objetos alrededor del dispositivo.

Sensor de luz. Se utiliza para medir la luz en el ambiente para ajustar el brillo de la pantalla entre otras funciones.

Pantalla táctil. Es una pantalla que mediante un toque directo sobre su superficie permite la entrada de datos y órdenes al dispositivo, y a su vez muestra los resultados introducidos previamente; actuando como periférico de entrada y periférico de salida de datos, así como emulador de datos interinos erróneos al no tocarse efectivamente. Este contacto también se puede realizar por medio de un lápiz óptico u otras herramientas similares.

GPS. El chip GPS incorporado en los dispositivos móviles permite determinar el punto geográfico

exacto del móvil o la persona en el planeta mediante la conexión a satélites del Sistema de posicionamiento global. Algunas variantes como AGPS (GPS Asistido) realizan accesos a redes de datos para mejorar la ubicación. Además, ciertos dispositivos se conectan a la redes GLONASS (el servicio ruso de posicionamiento). El punto geográfico es definido por la latitud y la longitud [1] (Figuras 1.3 y 1.4 respectivamente)

Figura 1.3 Representación de las latitudes

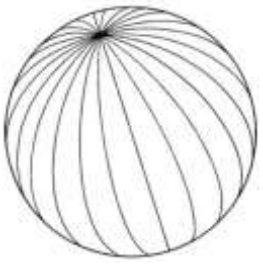

Figura 1.4 Representación de las longitudes

Cámara. Está incorporada en los teléfonos desde principios del presente siglo. Aunque los sensores se limitan al rendimiento con poca luz, en los últimos años se incorporan cámaras más potentes para la obtención de mejores imágenes. Las cámaras de los teléfonos inteligentes se utilizan como dispositivos de entrada en aplicaciones de numerosos proyectos de investigación y comerciales. Un ejemplo de éxito comercial es el uso de los códigos QR (Figura 1.5).

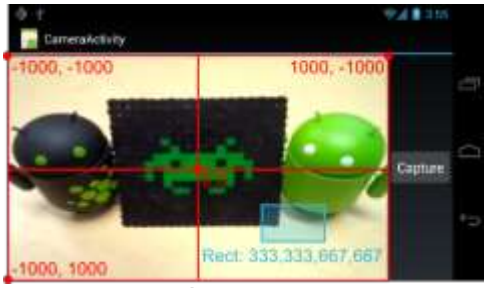

Figura 1.5 Acceso a la cámara de un dispositivo Android.

1.1.3 Conectividad

La conectividad es un tema importante en los últimos años pues los intercambios de información desde el Smartphone con servidores son más comunes y se realizan de forma inalámbrica usando comúnmente las redes de comunicación de proveedores de telefonía celular así como con redes Wi-Fi.

Wi-Fi. Es un mecanismo de conexión inalámbrica entre dispositivos electrónicos. Cumple con los estándares 802.11 relacionados a redes inalámbricas de área local. Los dispositivos móviles actuales cuentan con esta característica que usan para realizar conexiones a Internet a través de puntos de acceso como routers, switches, etc.

Bluetooth. Facilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de los 2,4 GHz. Es una especificación para redes de área

personal y fue diseñado para dispositivos de bajo consumo energético que requieran corto alcance de emisión. Es muy popular para transferir archivos entre dos dispositivos móviles en un radio cercano que alcanza hasta los 20 metros de distancia.

NFC. Siglas en español de Comunicación de Campo Cercano, es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos. Está basado en ISO 14443 y Felica. NFC se comunica mediante inducción en un campo magnético, en donde dos antenas de espiral son colocadas dentro de sus respectivos campos cercanos. Trabaja en la banda de los 13,56 MHz, esto hace que no se aplique ninguna restricción y no requiera ninguna licencia para su uso.

3G. Corresponde a la tercera generación de comunicación de telefonía móvil. Proporcionan la posibilidad de transferir voz, datos etc. Esta tecnología permite velocidades de descarga de alrededor de 3 Mbits/s. Su evolución es conocida como 3-5G la cual usa HSDPA (High Speed Download Packet Acces) que permite velocidades de descarga de hasta 14,4 Mbits/s. Generalmente este servicio es adquirido por medio de contratos de plan de datos ofrecidos a diversos dispositivos desde computadoras hasta rastreadores vehiculares.

4G LTE. Es la cuarta generación de comunicación de telefonía móvil. Está basada completamente en el protocolo IP y puede dar una velocidad hasta 100 Mbits/s de descarga.

1.2 Plataformas de desarrollo de aplicaciones móviles

IOS SDK. Es un paquete de herramientas para desarrolladores proporcionado por la empresa Apple (Figura 1.6). El iOS SDK se descarga exclusivamente desde el IDE Xcode, el cual es exclusivo para sistemas operativos OS X y requiere de un registro en la comunidad de desarrolladores de Apple. Con este Kit de desarrollo de software se programan aplicaciones para el iPhone, iPod y iPad en sus distintas versiones respectivas. Las aplicaciones se ofrecen en la tienda App Store, aunque requieren de una exhaustiva revisión para su publicación.

Figura 1.6 Entorno de desarrollo para iOS

Android SDK. El kit de desarrollo para Android incluye un conjunto de herramientas de desarrollo.

Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales (Figura 1.7). Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Max OS X 10.4.9 o posterior, y Windows XP o posterior. La programación de aplicaciones se hace habitualmente en Java y se utilizan entornos de desarrollo como Eclipse o App Inventor. La plataforma Android es una de las preferidas por los desarrolladores tomando parte del 67% de estos a nivel mundial. Las aplicaciones creadas pueden ser ofertadas en la tienda online de Google: Play Store.

Figura 1.7 Entorno de desarrollo para Android.

Windows Phone SDK. Microsoft lanzó un SDK para la programación de aplicaciones en Visual Studio (Figura 1.8) y que pueden ser desarrollados en dos importantes plataformas:

- Silverlight para Windows Phone: Permite el desarrollo de aplicaciones basadas en XAML. Hereda directamente características de .NET Framework, código administrado y el CLR. Sus características principales son el soporte para lectura y escritura de ficheros, manipulación de XML y manejo de gráficos.
- Microsoft XNA Framework es una implementación nativa de .NET Compact Framework que incluye un conjunto de bibliotecas para el desarrollo de juegos.

Las aplicaciones de Windows Phone pueden comercializarse en el Market Place de Microsoft y existen cerca de 100,000 disponibles para su descarga.

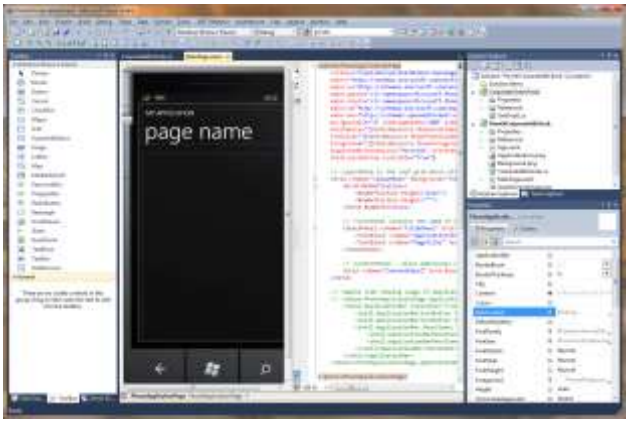

Figura 1.8 Entorno de desarrollo para Windows Phone

1.3 Geolocalización, navegación y mapas

1.3.1 Sistemas de geolocalización

También llamada georreferenciación, la geolocalización implica el posicionamiento que define la localización de un objeto en un sistema de coordenadas determinado. Este proceso es generalmente empleado por los sistemas de información geográfica, un conjunto organizado de hardware y software, más datos geográficos, que se encuentra diseñado especialmente para capturar, almacenar, manipular y analizar en todas sus posibles formas la información geográfica referenciada, con la clara misión de resolver problemas de gestión y planificación[8].

Sistema de posicionamiento global (GPS). El Sistema de posicionamiento global fue implementado por el departamento de defensa de los Estados Unidos de América en 1993 comenzando como un proyecto para actividades militares y abierto al público en el año 2000. El funcionamiento básico de un GPS se basa en el principio matemático de triangulación. Para calcular la posición actual es necesario que el GPS determine con exactitud la distancia a la que se encuentra de los satélites. La red de satélites está compuesta por 24 satélites dispersos en 6 órbitas polares, cada uno recorre 2 veces la tierra en 24 horas, garantizando que existan al menos 4 satélites dispuestos para la consulta de posicionamiento (Figura 1.9). El sistema GPS permite rastrear en tiempo real la ubicación de una persona, animal o vehículo siempre que cuente con un dispositivo que emita las señales para permitir su ubicación.

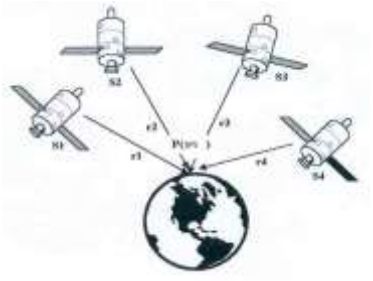

Figura 1.9 Triangulación por GPS desde 4 satélites.

GLONASS. Es el sistema global de navegación por satélite desarrollado por la Unión Soviética, siendo administrado hoy por la Federación Rusa. Cuenta con una constelación de 31 satélites situados en tres planos orbitales con 8 satélites cada uno. El sistema comenzó a ser funcional en el año de 1996 con fines militares pero a partir de 2007 fue abierto al público en general. Goza de una precisión cercana al metro de proximidad pero solo limitada a actividades bélicas y no a cualquier usuario. Es la contraparte del sistema estadounidense GPS y el sistema GALILEO europeo.

A-GPS. El GPS asistido (Assisted Global Positioning System) es usado en teléfonos y dispositivos móviles. Resuelve inconvenientes del GPS con el uso de receptores de referencia. Se conecta a servidores de localización que facilita datos de interés al dispositivo móvil, tales como la lista de efemérides de los satélites que estén en el campo visible. En algunos casos puede devolver valores de altitud o elevaciones del terreno para completar una localización de tres dimensiones (Figura 1.10).

Figura 1.10 Sistema de posicionamiento A-GPS

Geolocalización con JavaScript. La API de geolocalización de JavaScript permite que un navegador web use distintos métodos como el GPS, la dirección IP, triangulación de antenas de redes móviles o la localización del WI-FI que se use. El navegador web utiliza estos recursos para determinar la latitud y la longitud exacta en la que se encuentre el usuario. La precisión varía de acuerdo al dispositivo que se esté usando por lo que el resultado es aproximado [7] (Figura 1.11).

Figura 1.11 Geolocalización con JavaSript aplicada en el API de Google Maps

1.3.2 Servicios de navegación y localización web.

Google Maps API. Google Maps fue desarrollado originalmente por Lars y Jens Rasmussen, co-fundadores de Where 2 Technologies, empresa dedicada a soluciones de mapeo. La empresa fue adquirida por Google en 2004. Antes de que hubiera una API pública, algunos desarrolladores descubrieron la manera de hackear Google Maps para incorporar los mapas en sus propios sitios web. Esto llevó a Google a la conclusión de que había una necesidad de una API pública, y en junio de 2005 fue lanzado públicamente. Google Maps es HTML, CSS y JavaScript trabajando juntos. Los mapas son imágenes que se cargan en el fondo de una pantalla a través de peticiones ejecutadas con tecnología AJAX, y se insertan en un DIV de una página HTML. El API consiste de archivos JavaScript que contienen las clases, métodos y propiedades que se usan para el comportamiento de los mapas.

Open Street Maps. También conocido como OSM, es un proyecto colaborativo para crear mapas libres y editables. Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles, ortografías y otras fuentes libres. Esta cartografía, tanto las imágenes

creadas como los datos vectoriales almacenados en su base de datos, se distribuye bajo licencia abierta Open Database License (ODbL).

1.4 Realidad aumentada y modelado de objetos en 3D

1.4.1 Introducción a la realidad aumentada

La realidad aumentada de investigación (RA) explora la aplicación de imágenes generadas por ordenador en tiempo real a secuencias de vídeo como una forma de ampliar el mundo real. El término es usado para definir una visión directa o indirecta de un entorno físico del mundo real, los cuales se combinan con elementos generados por ordenador para dar la apariencia de una realidad junta o mixta.

1.4.1.1 Definiciones

Una de las definiciones de realidad aumentada fue dada por Ronald Azuma en 1997. La definición de Azuma dice que la realidad aumentada [4]:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- Está registrada en 3D.

Además Paul Milgram y Fumio Kishino definen en 1994 la realidad de Milgram-Virtuality Continuum (Figura 1.12) como un continuo que abarca desde el entorno real a un entorno virtual puro. Entre medio existe la Realidad Aumentada (más cerca del entorno real) y Virtualidad Aumentada (está más cerca del entorno virtual) [9].

Figura 1.12 Modelo "Continuo-Virtual" de Milgram

Los complementos virtuales usados en realidad aumentada suelen ser modelos creados por ordenador y objetos 3D o 2D, muchos de los cuales suelen ser textos, íconos o cualquier imagen que amplíe la información de la imagen real. El término de realidad aumentada se ha extendido enormemente debido al interés por esta tecnología por el público en general.

La realidad aumentada se ha aplicado en numerosas áreas, tales como arquitectura, entretenimiento (Figura 1.13), educación, arte, medicina y comunidades virtuales.

Figura 1.13 Ejemplo de RA empleado en juegos (Tomado del juego Droid Shooting [15])

1.4.1.2 Funcionamiento

Cuatro etapas principales son esenciales para llevar a cabo la realidad aumentada y se muestran en la figura 1.14.

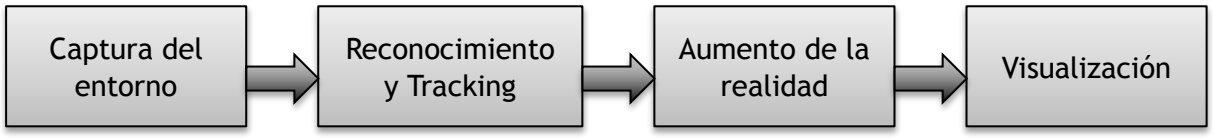

Figura 1.14 Etapas del proceso de realidad aumentada.

En primer lugar se realiza la captura del entorno con la cámara. Después se realiza el proceso de reconocimiento para establecer el seguimiento del punto de vista (ViewPoint Tracking), con el fin de condicionar los objetos virtuales para su visualización. En el tercer paso se eligen y renderizan los objetos virtuales que serán colocados en la visualización. La última etapa superpone la capa virtual creada sobre la real.

1.4.1.3 ViewPoint Tracking

Existen 2 métodos para solucionar el problema de ViewPoint Tracking, las cuales se diferencian de acuerdo a las necesidades de precisión exigidas. Por ejemplo, para aplicaciones de medicina o cirugía el posicionamiento del objeto en el espacio correcto es esencial.

Localización espacial usando geoposicionamiento. De acuerdo a la posición de un dispositivo GPS, se calcula la posición de los objetos virtuales que se añaden en la captura del entorno. La precisión se aumenta con ayuda de sensores como el acelerómetro o la brújula (Figura 1.15). No es recomendable para aplicaciones que exijan precisión.

Figura 1.15 Realidad aumentada con geoposicionamiento.

Reconocimiento espacial usando visión artificial. Es un método que se realiza a nivel de software y es más complejo que el anterior. Se divide en dos métodos:

- **Marcadores físicos.** Se conoce también como Marker tracking y se basa en la detección de "marcadores". Los ejemplos más usados son los códigos QR (Figura 1.16).

Figura 1.16 Ejemplo de código que representa el texto "Android".

- **Sin marcadores.** Esta técnica es muy compleja debido al coste computacional requerido es muy alto. El software o aplicación debe ser capaz de reconocer objetos que componen la escena real a partir de reconocimiento de patrones como líneas, formas y fisionomía. Actualmente se limita al reconocimiento de ciertos objetos como caras, superficies u objetos de forma concreta. Los objetos identificados se denominan "targets" [2].

1.4.2 Realidad Aumentada y Smartphones

Una de las técnicas de visualización para mostrar realidad aumentada es el uso de un display (generalmente un Smartphone), en la que todas las soluciones utilizadas hasta la fecha por los diferentes dispositivos de mano emplean técnicas de superposición sobre el video con la información gráfica. Inicialmente los dispositivos de mano empleaban sensores de seguimiento tales como brújulas digitales y GPS que añadían marcadores al video. Más tarde el uso de sistemas, como ARToolKit [3], nos permitía añadir información digital a las secuencias de video en tiempo real. Hoy en día los sistemas de visión como SLAM o PTAM son empleados para el seguimiento. El display de

mano promete ser el próximo éxito comercial de las tecnologías de Realidad Aumentada. Sus dos principales ventajas son el carácter portátil de los dispositivos de mano y la posibilidad de ser aplicada en los teléfonos con cámara.

1.4.3 Herramientas para la implementación de realidad aumentada

ARToolKitPlus. ARToolKitPlus es una variante de ARToolKit [3], optimizada para el desarrollo de aplicaciones móviles. Fue desarrollada por la Universidad de Graz en 2007. Inicialmente era de código cerrado por lo que no está muy documentada. La librería implementa módulos para el cálculo de la orientación y posición de la cámara relativa a los marcadores en tiempo real. Debido a su elevada demanda más tarde se liberó su código. Sin embargo su poca documentación la hace poco recomendable para el uso por parte de desarrolladores poco experimentados. Además el proyecto ha sido abandonado y la librería ha sido reemplazada por StudierStube Tracker y Studierstube ES desarrollado por la misma universidad.

Layar. Es un navegador de realidad aumentada, desarrollado para Android e iOS. Tiene licencia privativa por lo que no dispone de acceso al código fuente. El funcionamiento del software se basa en el geoposicionamiento y no en el reconocimiento de marcas.

Vuforia. Es una plataforma de desarrollo de aplicaciones de RA para Android e iOS desarrollada por el departamento de I+D de la empresa Qualcomm en Austria. Esta plataforma fue publicada en 2010. Una de las principales ventajas de esta plataforma es que se basa en el reconocimiento de marcas naturales, incluyendo objetos 3D, y que existe una extensión para Unity 3D que permite crear escena virtuales con animaciones y muy completas. (Figura 1.17)

La plataforma se presenta en código abierto aunque su uso con Unity 3D requiere de la adquisición de la licencia de esta.

Figura 1.17 Aplicación "Sesame Street" que usa tecnología Vuforia de Qualcomm

Una aplicación de Realidad Aumentada basada en Vuforia, consta de una serie de componentes esenciales. A continuación se describirán términos referentes a la arquitectura de clases básicas de Vuforia, muchos de estos componentes se tratan de *singletons*; cuya traducción literal sería "hijo único", y que consiste en una clase diseñada para tener sólo una instancia (o un número limitado de ellas). Conociendo esto, y por comodidad a la hora de emplear el lenguaje, se llamará instancia a un *singleton*.

- **Cámara** (camera, *singleton*): La instancia de la cámara se encarga de que cada fotograma capturado por la cámara digital se pase de forma eficiente al tracker. El desarrollador indica a la instancia de la cámara, con los métodos oportunos, cuando debe comenzar y detenerse la captura de fotogramas. Cada fotograma es enviado automáticamente a un dispositivo dependiente del formato de imagen y dimensiones.
- **Convertidor de imágenes** (Image Converter, *singleton*): La instancia del conversor de formato de pixel realiza la conversión entre el formato con el que trabaja la cámara (por ejemplo, YUV12) a un formato adecuado para el renderizado en OpenGL ES (por ejemplo, RGB565) y para el seguimiento (por ejemplo, luminancia). Esta conversión implica un submuestreo para tener la imagen capturada por la cámara en diferentes resoluciones disponible en la pila de fotogramas convertidos.
- **Tracker** (*singleton*): El tracker contiene los algoritmos de visión computacional para detectar y seguir (detect & track) los objetos en los fotogramas capturados por la cámara. Basado en la imagen tomada por cámara, diferentes algoritmos se ocupan de detectar nuevas imágenes de referencia (Targets) o marcadores (Markers). Los resultados se almacenan en un objeto de estado que es utilizado por el procesador de vídeo de fondo y al que puede accederse desde el código de la aplicación. El tracker puede cargar múltiples conjuntos de datos (datasets), pero sólo uno puede estar activo a la vez.
- **Procesador de vídeo de** fondo (Video Background Rendered, *singleton*): La instancia del procesador de vídeo de fondo procesa la imagen capturada por la cámara que se encuentra almacenada en el objeto de estado. El rendimiento del renderizado del vídeo de fondo está optimizado para dispositivos específicos.
- **Código de la aplicación** (Application Code):' El desarrollador se encarga de inicializar todos los componentes anteriores y llevar a cabo tres pasos fundamentales en el código de la aplicación. Por cada fotograma procesado, el objeto de estado se actualiza y se llama al método de procesamiento de la aplicación. El desarrollador debe:
    - Consultar el objeto de estado para los Targets y/o Markers nuevos que puedan aparecer en escena o si se actualiza su estado.
    - Actualizar la lógica de la aplicación con nuevos datos de entrada.
    - Renderizar la capa de Realidad Aumentada.
- **Recursos de imágenes de referencia** (Target Resources): Los Target Resources se crean mediante el Target Management System, que se encuentra disponible en el sitio de Vuforia. El dataset descargado contiene un fichero de configuración XML que permite al desarrollador configurar ciertas características de los trackables y un fichero binario que contiene la base de datos de los trackables. Estos elementos (XML y binario) son compilados por la aplicación a desarrollar en el paquete de instalación de la aplicación y los usa el SDK Vuforia en tiempo de ejecución.

En la figura 1.18 se observa la arquitectura de una aplicación con el SDK de Vuforia

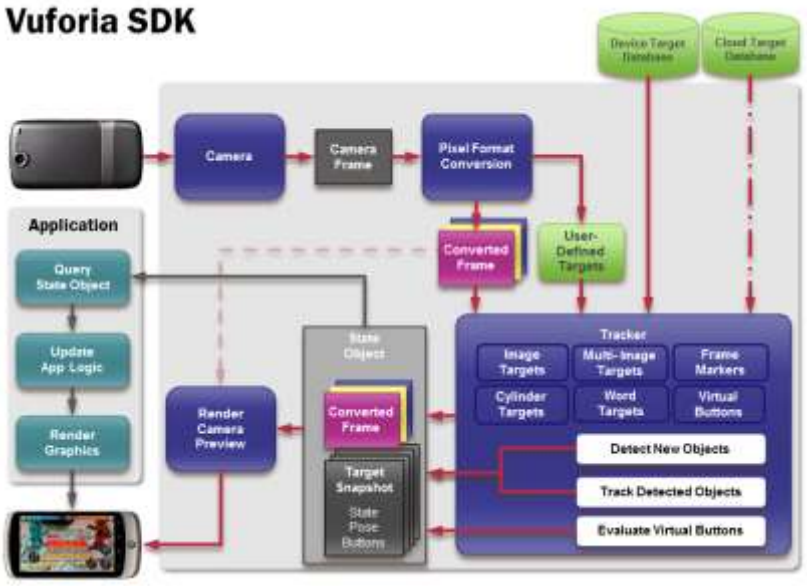

Figura 1.18. Arquitectura de una aplicación Vuforia [19].

1.4.4 Objetos 3D

El término gráficos 3D por computadora se refiere a trabajos de arte gráfico que son creados con ayuda de computadoras y programas especiales. Un gráfico 3D se origina mediante unos procesos de cálculos matemáticos sobre entidades geométricas tridimensionales producidas en un ordenador y cuyo propósito es conseguir una proyección visual en dos dimensiones para ser mostrada en una pantalla [11]. (Figura 1.19)

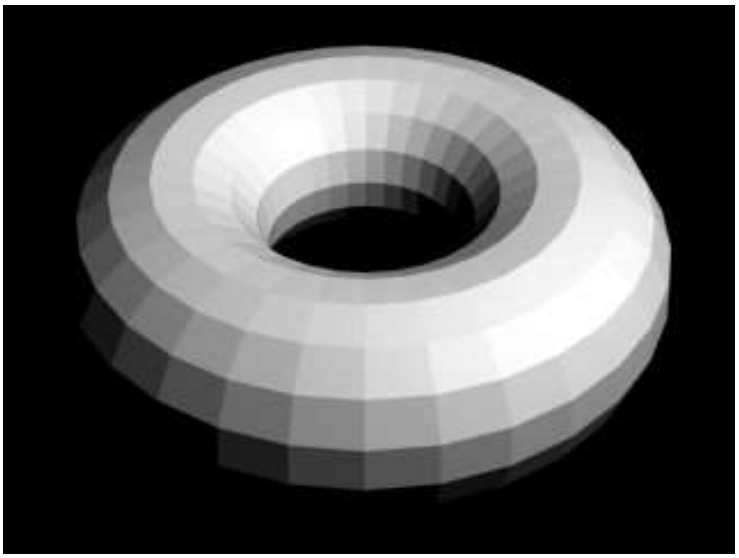

Figura 1.19 Representación tridimensional compuesta por 512 vértices y 1024 polígonos [17].

Los pasos para crear elementos gráficos tridimensionales se muestran a continuación:

Modelado. Consiste en dar forma a objetos individuales que luego serán usados en la escena. Existen diversos tipos de geometría para modelar con NURBSS [10] y modelo poligonal o subdivisión de superficies.

Iluminación. Creación de luces de diversos tipos puntuales, direccionales en área o volumen, con distinto color o propiedades. Esto es la clave de una animación.

Gran parte de la iluminación en 3D requiere del entendimiento físico de la luz en la realidad, este entendimiento puede ir desde lo más básico en el tema como por ejemplo el concepto de iluminación global hasta comportamientos complejos y extraños de la luz como la dispersión en superficies y subsuperficies.

Iluminación global. Se lleva a cabo para simular la iluminación de la vida real. Cada superficie circundante al objeto producirá una intensidad de iluminación distinta desde distintas direcciones y con distintas tonalidades de color sobre el objeto dependiendo del color de cada superficie y a su vez la luz también rebotará en el objeto de atención.

Animación. Los objetos se pueden animar en cuanto a:

- Transformaciones básicas en los tres ejes (XYZ), rotación, escala y traslación.
- Forma:
  - Mediante esqueletos: a los objetos se les puede asignar un esqueleto, una estructura central con la capacidad de afectar la forma y movimientos de ese objeto. Esto ayuda al proceso de animación, en el cual el movimiento del esqueleto automáticamente afectará las porciones correspondientes del modelo.
  - Mediante deformadores: ya sean cajas de deformación (*lattices*) o cualquier deformador que produzca, por ejemplo, una deformación sinusoidal.
  - Dinámicas: para simulaciones de ropa, pelo, dinámicas rígidas de objeto.

Renderizado. (*Render* en inglés) es un término usado para referirse al proceso de generar una imagen o vídeo mediante el cálculo de iluminación GI partiendo de un modelo en 3D. Este término técnico es utilizado por los animadores o productores audiovisuales (CG) y en programas de diseño en 3D como por ejemplo 3DMax, Maya, Blender, etc.

1.4.5 Software de modelado 3D

Blender. Este programa está dedicado especialmente al modelado, animación y creación de gráficos tridimensionales.

El programa fue inicialmente distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, GNU/Linux, Solaris, FreeBSD, e IRIX.

Unity 3D. Es un motor de videojuegos multiplataforma creado por Unity Technologies. Está disponible para plataformas Windows y Mac OS X, y permite crear juegos para Windows, OS X, Linux, XBOX, PlayStation, Wii, iPad, iPhone y Android. Cuenta con un plugin para juegos en navegadores (Figura 1.20).

Unity Technologies fue fundada en 2004 por David Helgason (CEO), Nicholas Francis (CCO), y Joachim Ante (CTO) en Copenhague, Dinamarca después de su primer juego, GooBall, que no obtuvo éxito. El éxito de Unity ha llegado en parte debido al enfoque en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles. En 2008, con el auge del iPhone, Unity fue uno de los primeros desarrolladores de motores en empezar a apoyar a la plataforma en su totalidad. Unity está siendo utilizado por el 53,1% de los desarrolladores, con cientos de juegos lanzados en dispositivos Android y iOS.

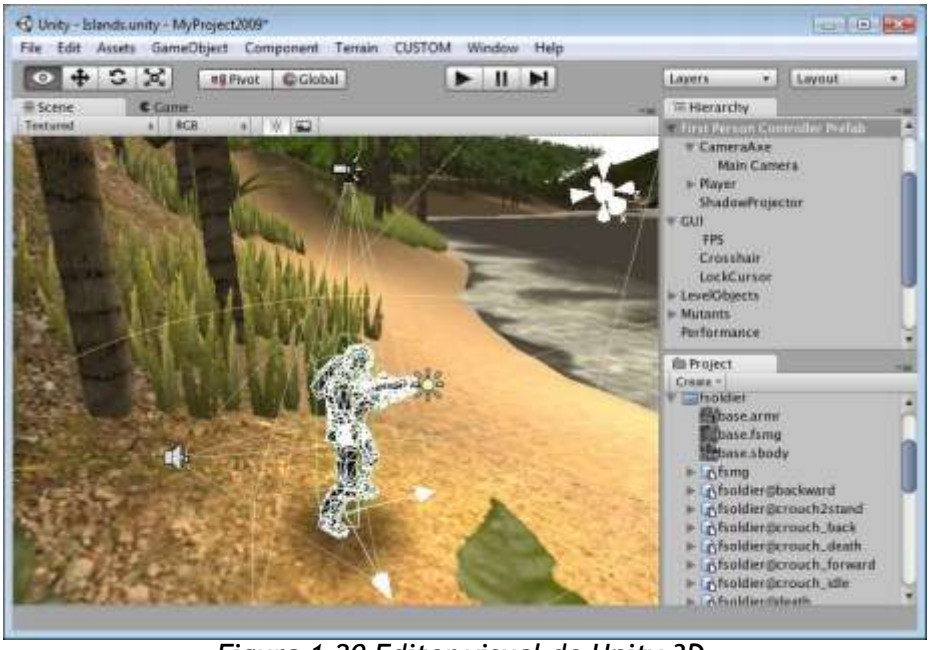

Figura 1.20 Editor visual de Unity 3D.

1.5 Servicios Web

Un servicio web es un conjunto de protocolos que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes ejecutadas en distintas plataformas se pueden comunicar entre ellas utilizando servicios web para intercambiar datos en redes de computadoras como Internet. Las organizaciones OASIS yW3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares [12].

Los servicios web proporcionan mecanismos de comunicación estándares entre diferentes

aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

1.5.1 Protocolos

JSON. Acrónimo de JavaScript Object Notation es un formato ligero para el intercambio de datos. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, etc., que atienden a millones de usuarios) cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente. Desde Diciembre de 2005, Yahoo comenzó a dar soporte opcional de JSON en algunos de sus servicios web.

SOAP. (Simple Objects Access Protocol) Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por David Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

REST. REpresentational State Transfer, es un estilo arquitectónico propuesto por Roy Fielding en 2000 para construir aplicaciones distribuidas inspirado en las características de la web. En la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

1.5.2 Frameworks

A continuación se listan los frameworks más populares para servicios web [14]:

.NET. Es un framework de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Permite crear servicios Web personalizados o utilizar servicios de aplicación integrados y llamar a estos servicios desde cualquier aplicación cliente. Su modelo de mensajería es un cliente/servidor e implementa los protocolos SOAP y WSDL.

Apache AXIS 2. Es un motor para servicios web. Es un rediseño total y una re implementación completa de la ampliamente difundida pila SOAP "Apache Axis". Existen implementaciones de Axis2 en Java y en C. Axis2 no solo provee la capacidad de agregar servicios web a las aplicaciones web, sino que además puede funcionar como servidor autónomo.

1.6 Patrones de arquitectura

Los patrones arquitectónicos, o patrones de arquitectura, son patrones de diseño de software que ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones

sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

1.6.1 Modelo vista controlador

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario [6]. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento [16].

1.6.2 MVC Android

El principal objetivo de usar el modelo vista controlador para desarrollar aplicaciones en Android es que se puede separar los datos de una aplicación, la interfaz del usuario y la lógica de negocio en tres distintos componentes que se relacionan para que la aplicación funcione. En la figura 1.21 se observa el modelo vista controlador de Android.

Figura 1.21 MVC Android.

1.6.3 MVC3 .NET

La implementación de Microsoft ASP.NET MVC proporciona una alternativa al modelo de formularios Web Forms de ASP.NET para crear aplicaciones web.ASP.NET MVC en un marco de presentación de poca complejidad y fácil de testear que, como las aplicaciones basadas en formularios Web Forms, se integra con características de ASP.NET como son las páginas maestras y la autenticación basada en pertenencias. En el contexto de ASP.NET el modelo vista controlador se puede interpretar de la siguiente manera:

- Toda la lógica de negocio y el acceso a datos es el Modelo (en muchos casos el modelo puede estar en uno o varios ensamblados referenciados).
- Las vistas contiene, básicamente, el código que se envía al navegador, es decir, el código HTML (y código de servidor asociado, siempre y cuando este código haga cosas de presentación, no de lógica de negocio).
- Los controladores reciben las peticiones del navegador y en base a esas, deciden que vista debe enviarse de vuelta al navegador y con qué datos.

La ventaja que primero salta a la vista de ASP.NET es la facilidad con la que se generan URL semánticas, es decir URL que tengan la forma http://servidor/ver/productos/cafeteras en lugar de http://servidor/productos/ver.aspx?code=cafeteras. Las URLs semánticas se indexan mejor en los buscadores y son una práctica SEO habitual. No es que en webforms no se puedan hacer, es que en ASP.NET MVC vienen de serie [18].

2. Estado del arte

Existe una gran variedad de sistemas orientados a la promoción de sitios turísticos así como aplicaciones móviles que usan realidad aumentada para mostrar dichos sitios. En este capítulo se analizan las aplicaciones desarrolladas por Universidades, dependencias gubernamentales y empresas privadas en el sector de promoción turística y vías de uso en realidad aumentada [1].

2.1 Aplicaciones móviles que usan realidad aumentada

Layar: es un navegador de realidad aumentada, desarrollado para Android y iOS. Tiene licencia privativa por lo que no dispone de acceso al código fuente. El funcionamiento del software se basa en el geoposicionamiento y no en el reconocimiento de marcas.

Está basado en un sistema de capas que funcionan sobre el navegador de realidad y que pueden ser mostradas o no a elección del usuario. El desarrollador implementa estas capas, en 2D o 3D, para añadir información aumentada a la imagen real. El sistema se compone de la aplicación cliente que se ejecuta en el dispositivo, un servidor central que provee los datos y un servidor privado para que el desarrollador gestione esos datos y los envíe al servidor central para finalmente visualizarlos en la aplicación. Las capas definidas por el usuario pueden ser puestas a disposición de la comunidad de manera centralizada. La etapa de renderizado de objetos 3D está optimizada para su uso en dispositivos móviles.

UNAM 360: Es una aplicación que permite "descubrir" el campus de la UNAM en Ciudad de México con el uso de realidad aumentada y los mapas de Google. Fue desarrollada por alumnos de la Facultad de Ingeniería de la UNAM pertenecientes al departamento de UNAM Mobile.

Museos df: Aplicación que permite conocer los museos más importantes de la Ciudad de México. No permite el uso de realidad aumentada, pero, además de la localización de museos, despliega información acerca de horarios, precios y eventos. Fue desarrollado por el departamento de UNAM Mobile.

2.2 Aplicaciones de turismo

Tecnología al servicio de la promoción turística en Querétaro

Rutas de Querétaro es un proyecto que ha tenido 2 años de gestación para los mercados internacionales y que se ha presentado en diversos eventos nacionales y regionales. Consiste en un libro online y una aplicación disponibles para la plataforma iOs. El costo total de la plataforma fue de tres millones de pesos incluyendo 225 prestadores de servicios turísticos. El punto destacado de este proyecto es poner en valor los atractivos y productos turísticos del estado de Querétaro, enmarcando cuatro rutas principales que pueden ser consultadas desde la aplicación [22].

2.3 Aplicaciones de navegación

Google Maps: El servicio de mapas online de Google fue lanzado el 8 de febrero de 2005 para computadoras personales que son esenciales en la vida diaria de un gran sector de la población, el cual mediante coordenadas permite ubicar puntos de referencia en todo el mundo. Integra imágenes por satélite, mapas tomados de agencias gubernamentales y ofrece servicios alternos como el clima, transporte público, relieves y tráfico en tiempo real en algunos países. Este sistema está desarrollado en JavaScript y XML. Google Maps tiene a su disposición un API para los desarrolladores que deseen manipularla.

Apple Maps: el sistema de mapas de la compañía Apple se encuentra disponible desde el 19 de Septiembre de 2012 exclusivamente para plataformas con el sistema operativo iOS. Fue lanzado para competir con el sistema de mapas de Google. Uno de sus principales inconvenientes es la baja calidad en sus mapas de tres dimensiones así como la falta de información en muchos lugares, direcciones faltantes o confundidas.

2.4 Aplicaciones de escritorio

Puebla desde el aire: Es una aplicación web de escritorio realizada por la Benemérita Universidad Autónoma de Puebla la cual muestra las rutas turísticas del centro histórico de la ciudad de Puebla. Además de mostrar los sitios más representativos del primer cuadro de la ciudad, muestra las rutas que un viajero puede tomar para conocer lugares sin perder detalles de los lugares por los que pasa mientras dura el recorrido. Su principal debilidad de este sistema es la navegación así como los colores usados para mostrar las rutas [23].

3. Análisis

3.1 Planteamiento

La finalidad de este proyecto de tesis es realizar una aplicación para dispositivos móviles de la línea Smartphone, específicamente con sistemas operativos Android, con la cual se podrá ubicar los sitios turísticos del centro histórico de la ciudad de Puebla, considerado patrimonio de la humanidad por la UNESCO, con geolocalización en un mapa, además de mostrar la información de cada sitio por realidad aumentada. Para sitios representativos de la ciudad el sistema será capaz de desplegar objetos en tercera dimensión con solo enfocar hacia la fachada del edificio. Se planea que la aplicación tenga conexiones con un servidor para consultar los sitios, además de contar con un sistema de navegación en tres dimensiones.

3.2 Objetivos generales y específicos

Objetivos Generales

Desarrollar una aplicación para dispositivos móviles empleando realidad aumentada y geolocalización para la ubicación de lugares turísticos en la ciudad de Puebla.

Objetivos específicos:

• Desarrollar una aplicación móvil para la plataforma Android que incluya realidad aumentada a partir de marcadores naturales y que actuará como una aplicación cliente. Mostrará en un mapa la posición de los lugares turísticos de la ciudad y se auxiliará del GPS para mostrar la ubicación actual.

• Examinar las principales características como ubicación, historia, etc. de los museos, iglesias, monumentos y sitios de esparcimiento en Puebla para situarlos en un mapa que será visualizado en la aplicación móvil.

• Crear un servicio web para gestionar la comunicación con la aplicación cliente e intercambiar datos, con la finalidad de obtener los marcadores de los sitios públicos y enviarlos a la aplicación cliente.

• Diseñar una base de datos para el almacenamiento de marcadores y programar sus respectivas consultas.

3.3 Descripción

La aplicación Puebla RA debe actuar como un cliente que solicita consultas a un servicio web implementado y al servicio de Google Maps. Esta aplicación debe solicitar la ubicación actual usando el GPS del dispositivo o conectándose a servicios de geoposicionamiento asistido para después solicitar al servidor la lista de marcadores cercanos de sitios turísticos. Debe tener la

opción de ubicar cada sitio por realidad aumentada mostrando información de cada lugar al enfocar el móvil a la posición del sitio, además de que en ciertos lugares representativos se deben desplegar animaciones en 3D que estén en contexto al lugar que se está enfocando.

3.4 Requerimientos de entrada del sistema

El sistema requiere como entrada la ubicación del móvil con tres parámetros: latitud y longitud, la cual es devuelta por el GPS, GLONASS o A-GPS, y un radio de alcance en kilómetros. Después de obtener la ubicación obtendrá la lista de lugares o marcadores cercanos (de acuerdo al radio deseado) desde el servidor en formato JSON. Un ejemplo de petición hacia el servidor sería "*19.00504,-98.204773,2*", donde los parámetros corresponden a la latitud, longitud y el radio de alcance respectivamente. Los marcadores recibidos se muestran en tiempo real en la pantalla del dispositivo tomando en cuenta su ubicación geográfica y son almacenados en caché mientras la aplicación esté activa.

3.4.1 Formatos del Navegador RA

3.4.1.1 Formato de envío

La ubicación del sitio será enviada desde el dispositivo móvil al servicio web REST de la siguiente forma:

```
http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/latitud,longitud,radio
```

Donde latitud y longitud corresponden a la ubicación actual del usuario, y radio corresponde a la distancia en kilómetros a la redonda a la que se quiere obtener marcadores.

Ejemplo de envío:

```
http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/19.00504,-98.204773,2
```

3.4.1.2 Formato de recepción

El servidor devuelve una lista de sitios turísticos (denominada geoCTI) calculados de acuerdo al radio que se envió como parámetro. Cada elemento de la lista tiene los siguientes parámetros:

- *id*
- *id_categoria*
- *summary*
- *tittle*
- *elevation*
- *lng*
- *lat*
- *url_imagen*

Donde **id** es el identificador del marcador, i**d_categoria** es el identificador para la categoría de dicho marcador, **summary** tiene un pequeño resumen del marcador, **tittle** contiene el nombre del marcador, **elevation** tiene la altitud a la que se encuentra geográficamente el marcador, **lng** es la

longitud de acuerdo a la posición geográfica del marcador, **lat** es la latitud de acuerdo a la posición geográfica del marcador y **url_imagen** contiene un vínculo de una imagen del sitio que almacena el marcador.

Un ejemplo del marcador devuelto por el servicio web es el siguiente:

```
"Id":2,"id_categoria":1,"summary":"Fuente de la facultad de ciencias de la
computación","tittle":"Fuente FCC","elevation":2100,"lng":-
98.20451,"lat":19.005008,"url_imagen":"http://cs.buap.mx"
```

3.4.2 Formato de muestreo

3.4.2.1 Navegador RA

Los marcadores recibidos desde el servidor son deserializados y cada uno es asignado a un objeto llamado **Marcador**, que posteriormente serán mostrados en la pantalla del dispositivo. El formato de asignación del objeto Marcador es el siguiente:

```
Marcador(nombre, latitud, longitud, altitud, color, categoria, summary,
url_imagen);
```

Donde **nombre** es el nombre del sitio, **latitud** y **longitud** y **altitud** son la ubicación geográfica, **categoría** es el identificador de la categoría del marcador, **summary** contiene un resumen del marcador y **url_imagen** contiene un vínculo a una imagen.

3.4.2.2 Mapa de Google Maps

Cuando accedemos al API de Google Maps para mostrar el mapa de ubicación, se pasan como parámetros los marcadores (objetos Marker), los cuales serán mostrados en el mapa de Google. El formato de los marcadores que se envían es el siguiente:

```
GoogleMaps(MENSAJE_SITIOS, (ArrayList<String>) msnSitios);
```

Donde MENSAJE_SITIOS es el identificador de la lista de marcadores a enviar al mapa y msnSitios contiene todos los marcadores que se dibujarán en el mapa. El parámetro msnSitios tiene el siguiente formato:

List msnSitios;
```
msnSitios.add (nombre,latitud,longitud);
```

msnSitios es una lista de marcadores la cual tiene tres elementos: **nombre**, que contiene el nombre del marcador, **latitud** y **longitud**, que contienen la ubicación geográfica.

3.4.2.3 Formatos del Visualizador de Realidad aumentada

El dispositivo móvil determina los sitios turísticos cercanos que mostrará en la pantalla para visualizarlos por realidad aumentada. El sistema será capaz de identificar solo los más cercanos y mostrarlos con visualizaciones en 2D con información del sitio. Para ciertos sitios designados se mostrara un objeto 3D, en el cual se enfoca la fachada de un edificio (La catedral de puebla, por ejemplo) y se visualiza en la pantalla el objeto. Es necesario un formato en XML que contiene el marcador natural o target necesario para ver los objetos en 3D.

3.4.2.4 Formato XML del marcador

```
<?xml version="1.0" encoding="UTF-8"?>
<QCARConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="qcar_config.xsd">
  <Tracking>
    <ImageTarget name="puerta2" size="200.000000 266.600006" />
    <ImageTarget name="puerta1" size="200.000000 266.600006" />
  </Tracking>
</QCARConfig>
```

El archivo XML con los marcadores contiene el objeto ImageTarget dentro de Tracking, el cual tiene dos parámetros: **name** y **size**.**name** es el nombre del archivo .dat que contiene la información binaria del marcador que usarán las librerías de Vuforia y **size** son las dimensiones de dicho marcador o target.

3.4.2.5 Formato .h

El objeto que será visualizado en la pantalla del dispositivo está almacenado en un archivo .h. Dicho archivo debe contener cuatro vectores necesarios para renderizar el objeto 3D, que son los siguientes:

vertices: Número de vértices de objeto.
faces: Número de caras que tendrá el objeto.
normals: Se define el vector de normales.
texture coords: Coordenadas para dibujar la textura del objeto.

3.5 Proceso de interacción

La interacción aproximada del sistema se muestra en la figura 3.1:

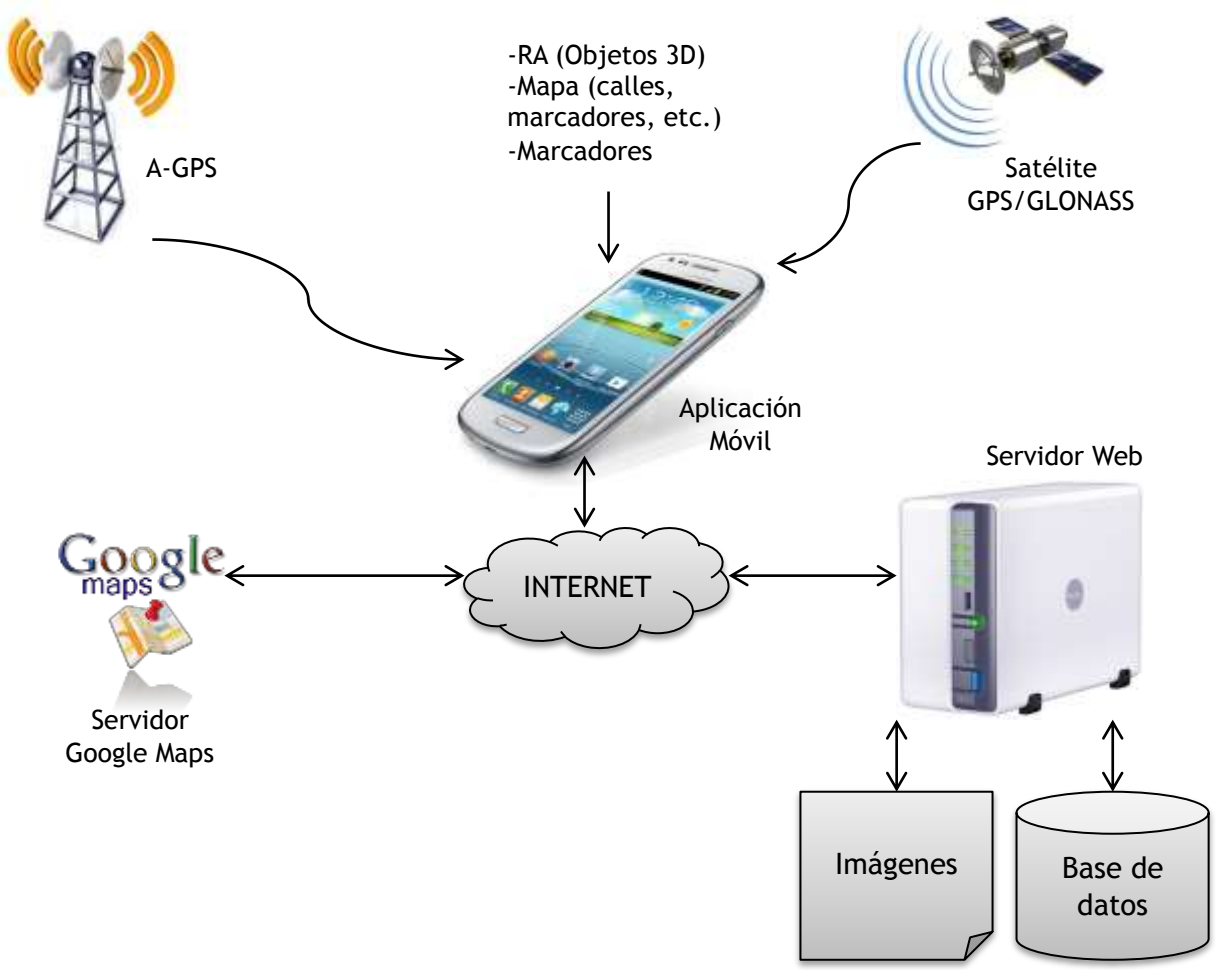

Figura 3.1 Interacción del sistema.

Al ejecutarse la aplicación se solicita la posición mediante GPS, GLONASS o A-GPS de acuerdo a las características del móvil, una vez obtenida, se envía al servidor web junto con el radio en kilómetros para verificar los puntos cercanos a la posición y devuelve los marcadores en formato JSON al cliente. Cuando el cliente recibe los datos correspondientes, se almacenan en el móvil y se muestran en pantalla auxiliados de la cámara del dispositivo, esto es, sobreponiendo la información sobre la pantalla e interactuando cada vez que el móvil cambia de dirección de enfoque. Se hace una solicitud al servidor de Google Maps para ubicar en un mapa los marcadores recibidos. La aplicación puede mostrar el mapa de forma normal o auxiliada con la cámara trasera del móvil mostrar los marcadores recibidos con realidad aumentada. Se despliegan información almacenada en el móvil de cada sitio y animaciones y objetos 3D en determinados lugares importantes.

4. Diseño del sistema

4.1 Diagrama por bloques

El sistema se compone de 5 bloques principales: **Presentación**, **Navegación RA**, **Mapas**, **Visualización RA** y, **Servicios web**. Cada bloque cuenta con distintas capas que ejecutan diversas funciones. Para el bloque de presentación, las capas se encargan de la comunicación con el servidor web y la interacción con el usuario. En el caso del bloque de Navegación RA, contiene las capas encargadas de la manipulación con el hardware del dispositivo y las operaciones que se realizan para mostrar la realidad aumentada. El bloque de Mapas es el encargado de mostrar los mapas. El bloque de visualización RA es el encargado de gestionar las librerías de Vuforia para mostrar la realidad aumentada con vistas en 3D. Por último, el bloque de Servicio Web contiene las capas encargadas de responder a peticiones por parte de los usuarios, así como interactuar con la base de datos de marcadores. En la figura 4.1 se observa el diagrama por bloques del sistema y la interacción entre bloques.

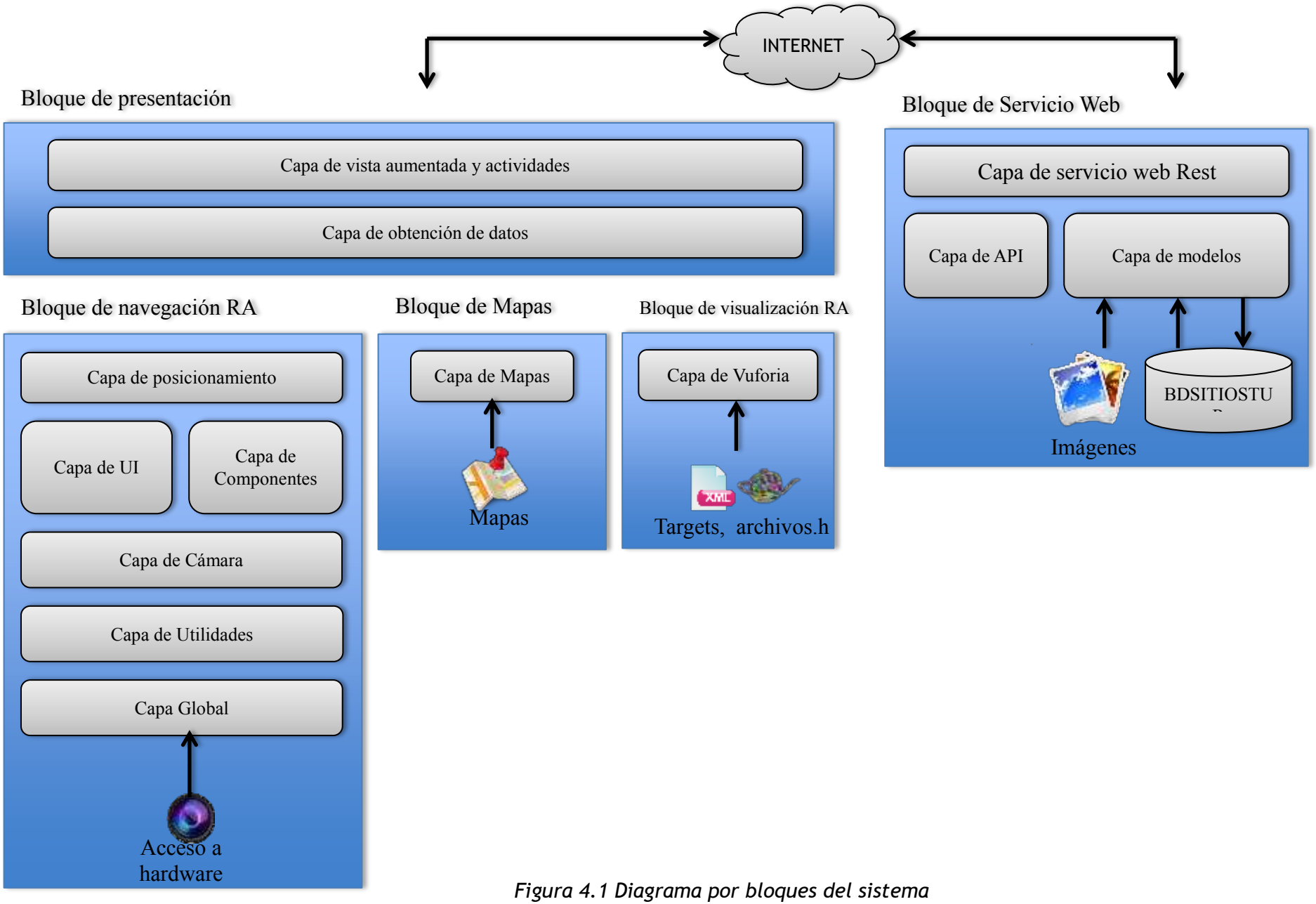

Figura 4.1 Diagrama por bloques del sistema

A continuación se describe cada bloque con sus respectivas capas.

Bloque de presentación

Este bloque es el encargado de la comunicación con el servidor web Rest así como la interacción entre el sistema y el usuario. Contiene las siguientes capas:

- Capa de vista aumentada y actividades: Esta capa será la encargada de mostrar todas las interfaces al usuario. Para la visualización del navegador de realidad aumentada debe tener presente el uso de la bloque de navegación RA. Para la visualización de objetos en 3D se debe comunicar con el bloque de visualización RA. La visualización de mapas se realiza en conjunto con el bloque de Mapas.

  La aplicación se conforma de 7 interfaces principales que se pueden observar en la figura 4.2

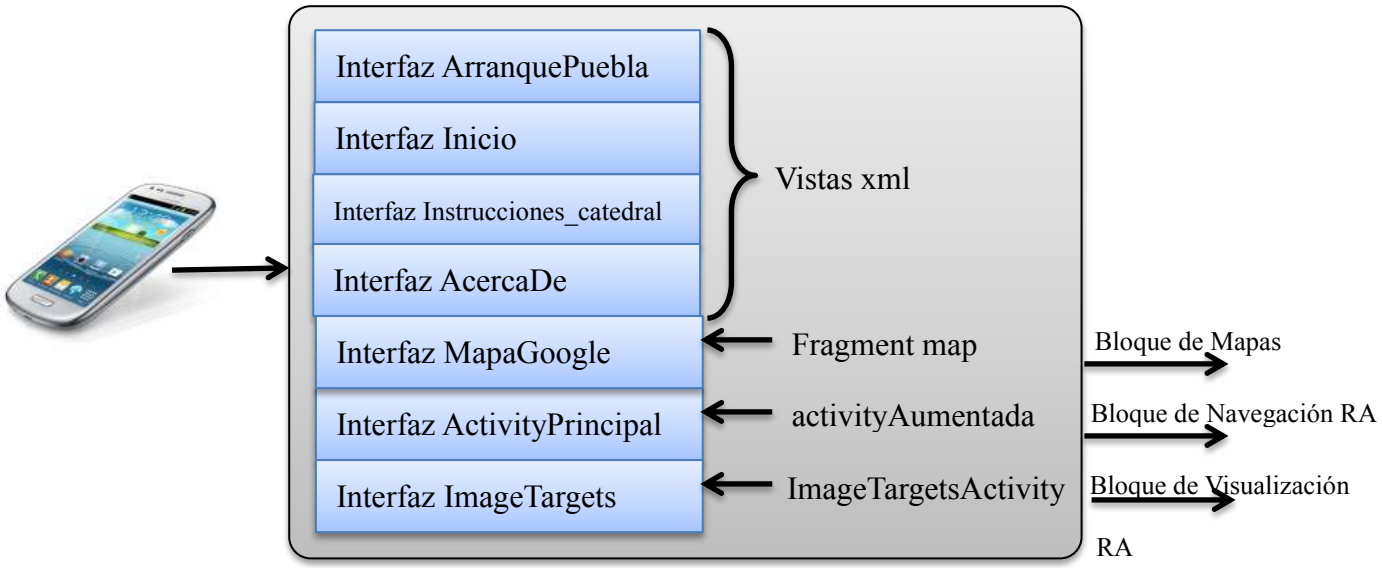

Figura 4.2 Capa de vista aumentada y actividades

- Capa de obtención de datos: Esta capa es la encargada de la comunicación con el servidor web para obtener los sitios turísticos.

Bloque de Navegación RA: Este bloque se encarga de ejecutar el motor de navegación de realidad aumentada.

- Capa de posicionamiento: Representa la ubicación física del dispositivo móvil y la disposición de los elementos en pantalla.

- Capa de UI: Es la encargada de dibujar los elementos como líneas, puntos, círculos, textos e imágenes.

- Capa de componentes: Gestiona los elementos que se muestran en pantalla, dichos elementos son: Marcadores, radar y barra de zoom.

- Capa de cámara: Obtiene el acceso a la cámara del dispositivo y contiene las clases para mostrar la imagen capturada en pantalla en tiempo real.

- Capa de utilidades: Contiene las clases que realizan las operaciones matemáticas necesarias para mostrar marcadores y calcular la inclinación del dispositivo.

- Capa global: Esta capa se encarga del almacenamiento general de los sitios mientras corre la aplicación.

Bloque de Mapas. Este bloque es el encargado de controlar la comunicación con el API de Google Maps para mostrar mapas.

- Capa de mapas: Esta clase muestra los mapas y los sitios turísticos como marcadores.

Bloque de visualización RA: Este bloque es el encargado de mostrar la realidad aumentada a partir de marcadores naturales usando las librerías de Vuforia.

- Capa de Vuforia: se encarga de renderizar y mostrar un objeto 3D a partir de una librería .h OpenGL, así como del procesamiento digital de una imagen en tiempo real.

Bloque de Servicio Web. Este bloque contiene todas las capas para el manejo de solicitudes en un servicio web. Cuenta con las siguientes capas:

- Capa de Servicio Web Rest: Esta capa será la encargada de recibir solicitudes desde el equipo móvil. Es capaz de devolver la información de los sitios cercanos a la posición desde donde se hizo la solicitud.

- Capa de modelo: Esta capa es contiene las operaciones y consultas que se realizan con la base de datos y de procesar la información que se reciba para devolverla en el formato correcto. Contiene todas las clases que serán serializadas y enviadas al cliente móvil.

- Capa de API: Esta capa redirige las acciones hacia el Controlador del servicio web a partir de la solicitud url.

4.2 Bloque de presentación

4.2.1 Diseño de la Capa de obtención de datos

La capa de obtención de datos tiene tres clases: FuenteDatos, que es una clase abstracta, FuenteDatosServidorRest que hereda de la clase anterior y ServidorRestDataSource, que hereda de la segunda clase. Estas clases se usan para comunicarse con el Servidor Web REST. A continuación se presenta cada clase:

Clase FuenteDatos

Esta clase abstracta solo obtiene la lista de marcadores mediante el método obtenerMarcadores.

Figura 4.3 Clase FuenteDatos

Clase FuenteDatosRed

Esta clase contiene los métodos para obtener la información desde el servidor web Rest.

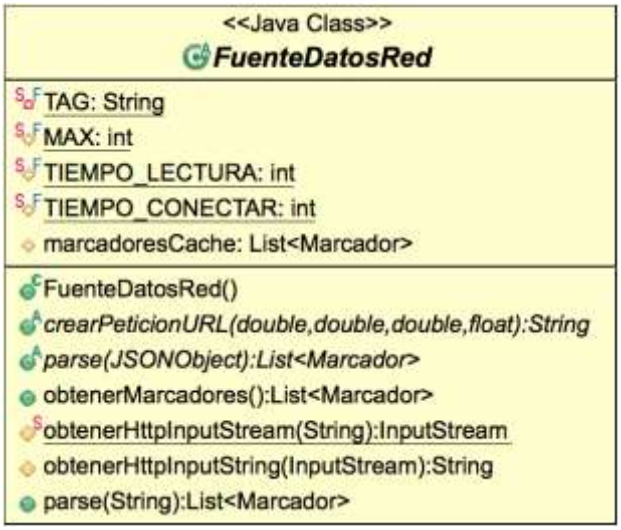

Figura 4.4 Clase FuenteDatosRed

Clase FuenteServidorRest

Esta clase se encarga de generar la solicitud hacia el servidor Rest así como de procesar la información recibida. Es heredada por la clase FuenteDatosRed.

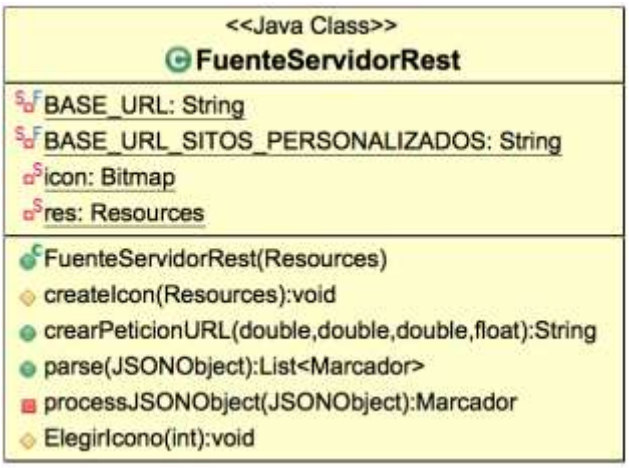

Figura 4.5 Clase FuenteServidorRest

4.2.2 Capa de Vista Aumentada y actividades

Esta capa se encarga de administrar todas las interfaces y actividades de la aplicación. Contiene todas las clases usadas para la interacción entre el usuario y el dispositivo móvil, encargadas de mostrar los elementos en pantalla así como controlar los eventos físicos que se apliquen al dispositivo móvil.

Clase activityAumentada

Es la clase que controla la posición de todos los controles de la vista aumentada (clase ViewAumentada)

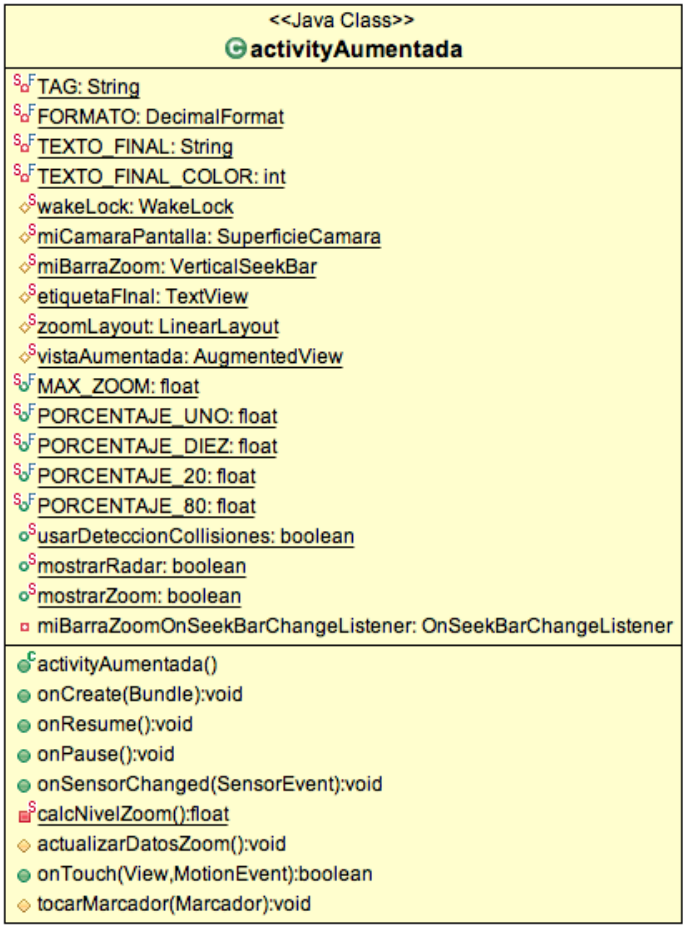

Figura 4.6 Clase activityAumentada

Clase activitySensores

Esta clase no tiene una interfaz, pero hereda de la clase Activity de Android. Su función es ser heredada por la clase activityAumentada (la cual es heredada por activityPrincipal). Se encarga de controlar los sensores acelerómetro, magnetómetro y giroscopio.

Figura 4.7 Clase activitySensores

Clase ViewAumentada

Esta clase es una extensión personalizada de la clase View de Android Framework usada para mostrar el radar, la barra de zoom que controla el radio y los marcadores que se muestran en la pantalla a través de la cámara.

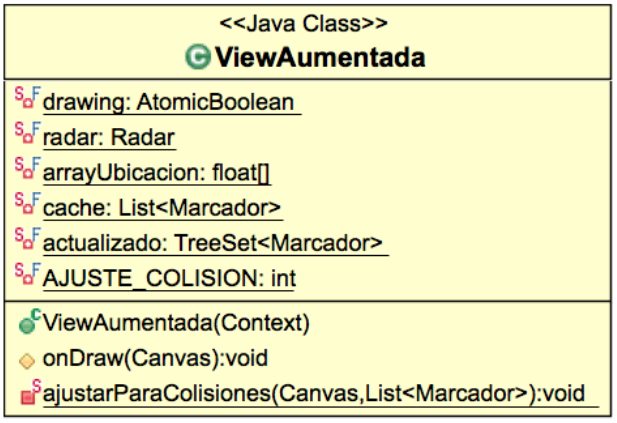

Figura 4.8 Clase ViewAumentada

Clase ActivityPrincipal

Esta clase es la encargada de lanzar la actividad principal del navegador RA, mantener actualizados

los marcadores cada vez que se actualice la barra de zoom y mostrarlos en el radar.

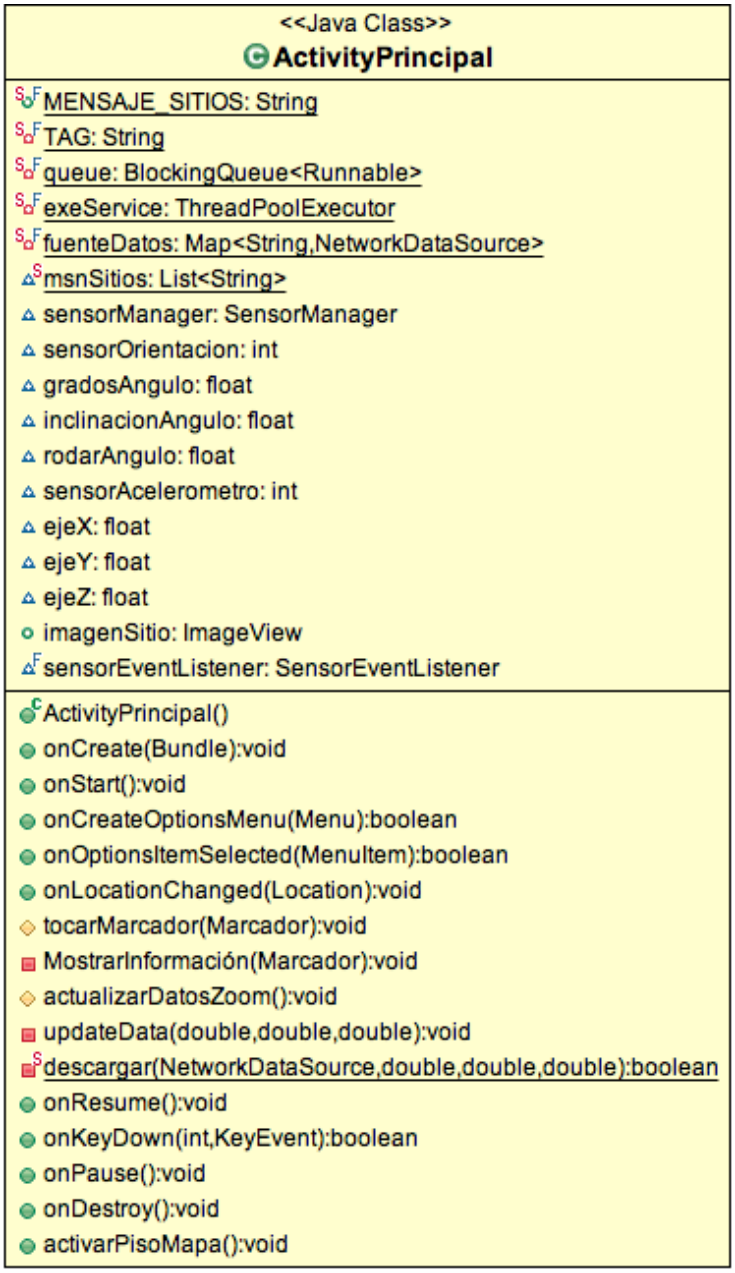

Figura 4.9 Clase ActivityPricipal

Clase ArranquePuebla

Esta clase muestra una interfaz de introducción cada vez que se inicia la aplicación en el dispositivo.

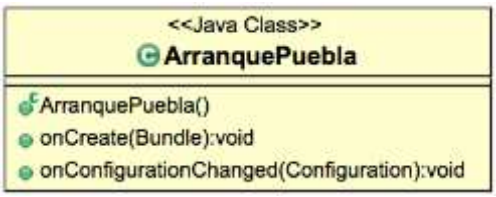

Figura 4.10 Clase ArranquePuebla

Clase Inicio

Esta clase será la encargada de mostrar el menú principal de la aplicación.

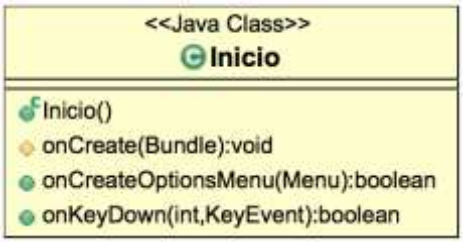

Figura 4.11 Clase Inicio

4.3 Bloque de Navegación RA

4.3.1 Capa de posicionamiento

En esta capa se muestran las clases para obtener la posición en el mundo del dispositivo móvil así como la posición que tendrán los objetos en la pantalla.

Clase UbicacionFisica

Esta clase representa la ubicación del dispositivo móvil en cualquier parte del mundo.

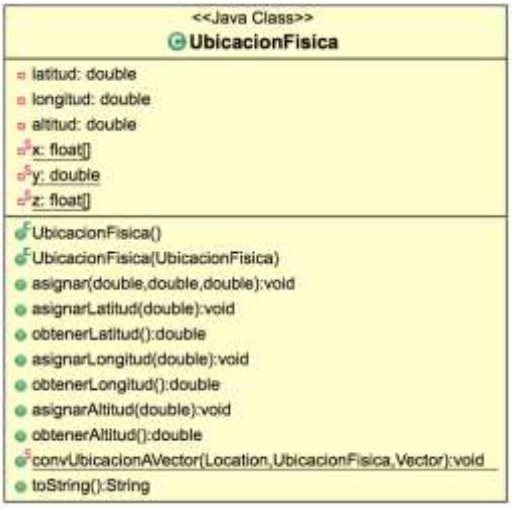

Figura 4.12 Clase UbicacionFisica

Clase PosicionEnPantalla

Esta clase se usa para posicionar en la pantalla del móvil los objetos tales como radar, barra de zoom y marcadores.

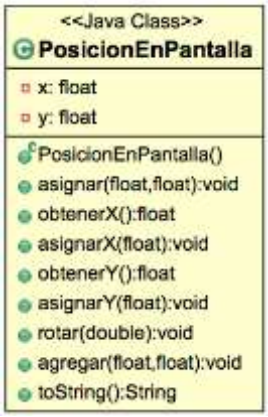

Figura 4.13 Clase PosicionEnPantalla

4.3.2 Capa UI

Esta capa contiene todas clases que se usan para dibujar en la pantalla del dispositivo los marcadores con los sitios turísticos, imágenes, líneas y puntos.

Clase DibujarObjeto

Es la clase base para todos los objetos que se dibujan (líneas, bitmaps, puntos) en la interfaz del navegador. Contiene métodos que serán sobrecargados por las demás clases Dibujar de la capa UI.

Figura 4.14 Clase DibujarObjeto

Clase DibujarCaja

Esta clase permite dibujar un contenedor para cada marcador.

Figura 4.15 Clase DibujarCaja

Clase DibujarTextoEnCaja

Esta clase se encarga de escribir el texto en la caja contenedor.

Figura 4.16 Clase DibujarTextoEnCaja

Clase DibujarCirculo

Esta clase sirve para dibujar un círculo para el radar que se muestra en pantalla.

Figura 4.17 Clase DibujarCirculo

Clase DibujarIcono

Esta clase sirve para dibujar el ícono del marcador.

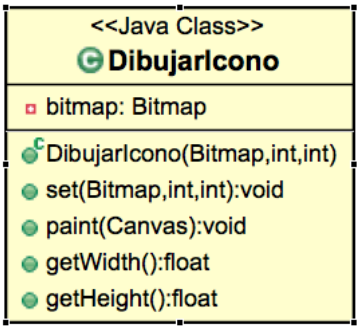

Figura 4.18 Clase DibujarIcono

Clase DibujarLinea

Esta clase es para dibujar líneas rectas. Es usada para dibujar las líneas del radar.

Figura 4.19 Clase DibujarLinea

Clase DibujarPunto

Esta clase es usada para dibujar un punto en el radar. Cada punto instanciado representa un marcador.

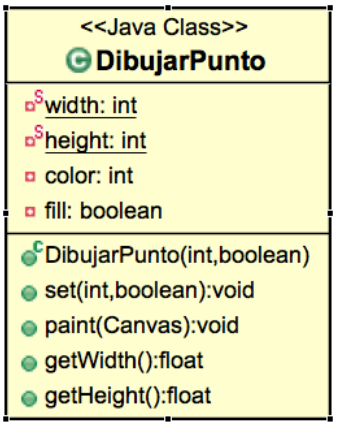

Figura 4.20 Clase DibujarPunto

Clase DibujarPosicion

Esta clase hereda de la clase DibujarObjeto y tiene la función de rotar y escalar todos los objetos que se muestran en pantalla

Figura 4.21 Clase DibujarPosicion

Clase DibujarPuntosRadar

Esta clase muestra la posición relativa de los marcadores dentro del radar.

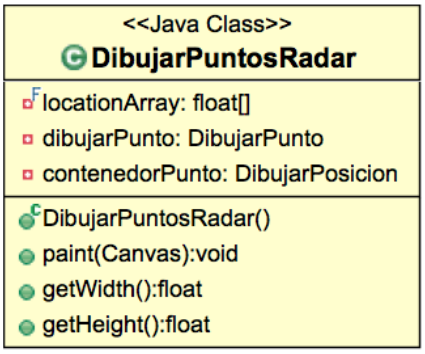

Figura 4.22 Clase DibujarPuntosRadar

Clase DibujarTexto

Es una clase que hereda de DibujarObjeto y escribe en pantalla el texto para el radar.

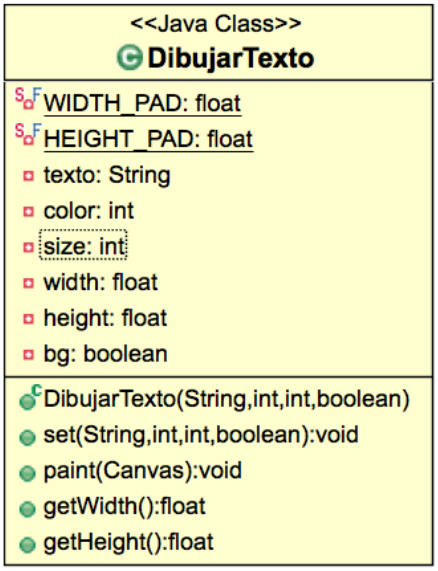

Figura 4.23 Clase DibujarTexto

Clase CargarImagenTask

Esta clase tiene la función de realizar un subproceso encargado de cargar la imagen desde el servidor mientras se muestra la información de un marcador.

Figura 4.24 Clase CargarImagenTask

4.3.3 Capa de utilidades

Esta capa contiene las clases para realizar las operaciones matemáticas sobre los vectores que se usan y cálculos de inclinación del dispositivo. Fueron adaptadas del Framework OpenSource Mixare Tools.

Clase Utilidades

Esta clase contiene el método obtenerAngulo que nosotros usamos cuando calculamos la inclinación del acimut (ángulo de inclinación con respecto al horizonte, también llamado rumbo).

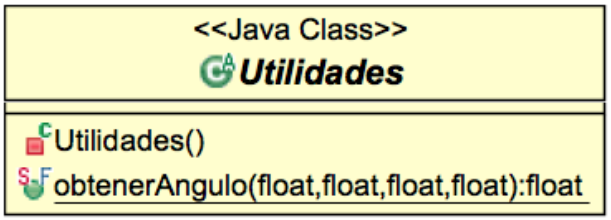

Figura 4.25 Clase Utilidades

Clase InclinacionAcimutCalc

Esta clase calcula el ángulo de inclinación y el acimut

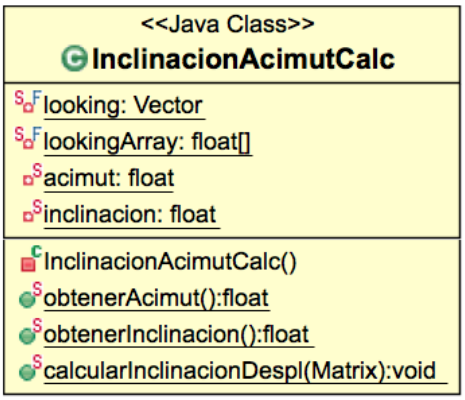

Figura 4.26 Clase InclinacionAcimutCalc

Clase vector

Esta clase se encarga de manejar las operaciones matemáticas detrás de los vectores. (Clase adaptada del framework Mixare tools mixare.org).

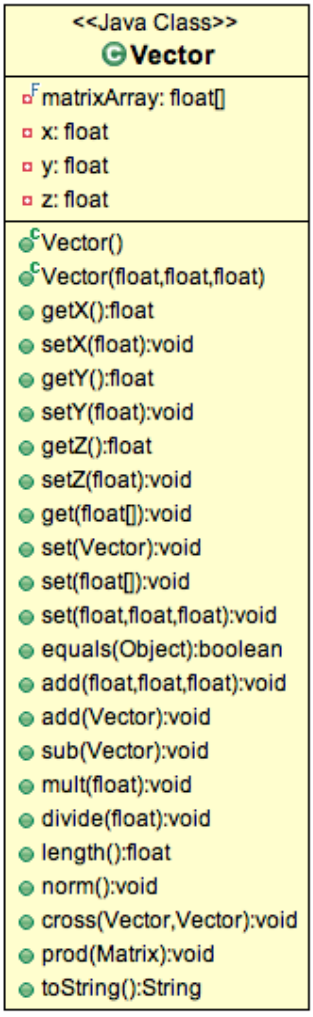

Figura 4.27 Clase Vector

Clase Matrix

Esta clase maneja las funciones de las matrices. (Clase adaptada del framework Mixare tools mixare.org).

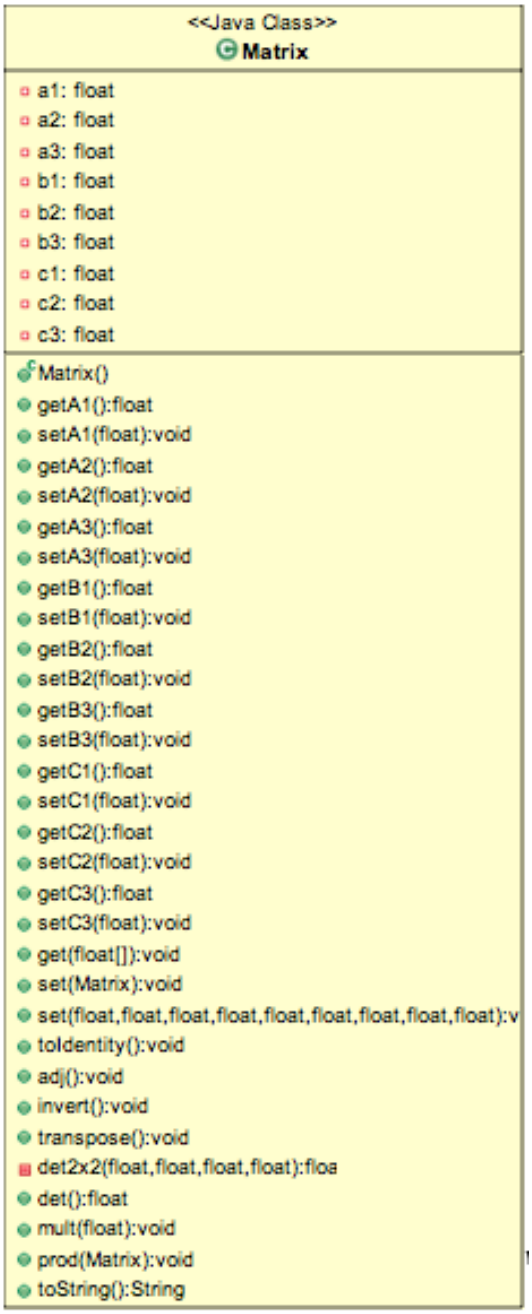

Figura 4.28 Clase Matrix

4.3.4 Capa de componentes

La capa de componente contiene las clases para formar los objetos que se muestran en pantalla.

Clase Radar

Esta clase es usada para mostrar el Radar junto con sus elementos (Puntos, líneas y texto).

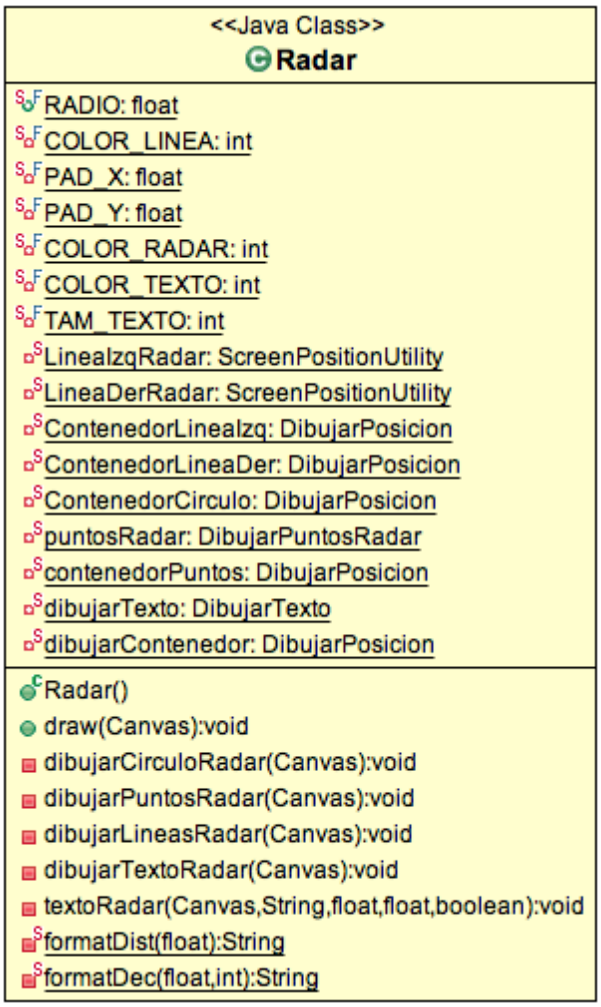

Figura 4.29 Clase Radar

Clase Marcador

Esta clase maneja la disposición y visualización de los marcadores en pantalla. Calcula que marcadores pueden ser mostrados en pantalla de acuerdo a la distancia a la que se encuentran y dibuja el ícono y el texto apropiado para cada marcador.

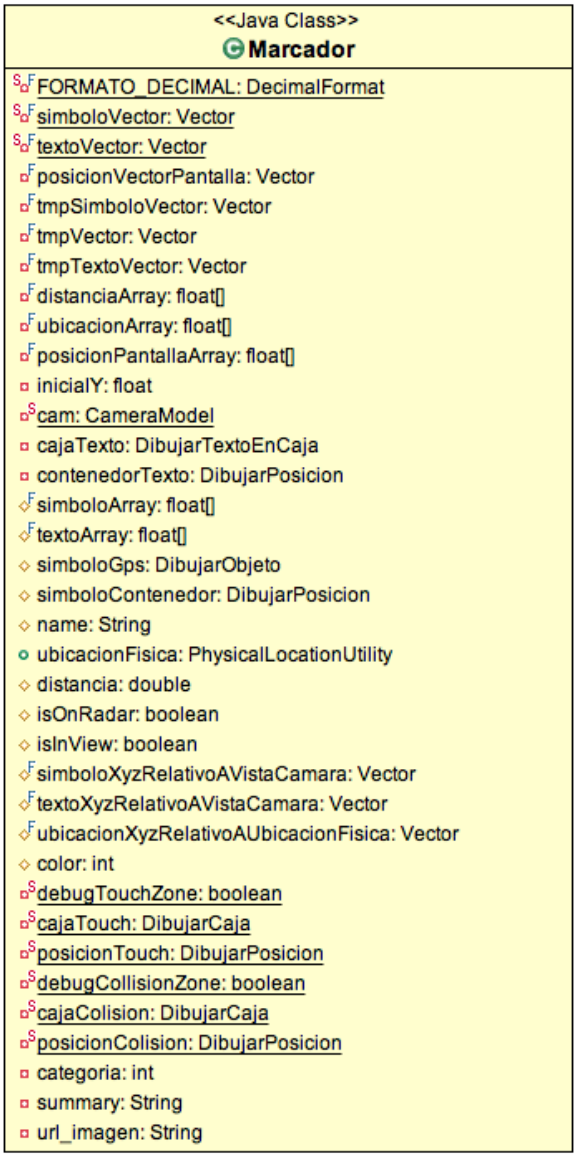

Figura 4.30 Clase Marcador (1)

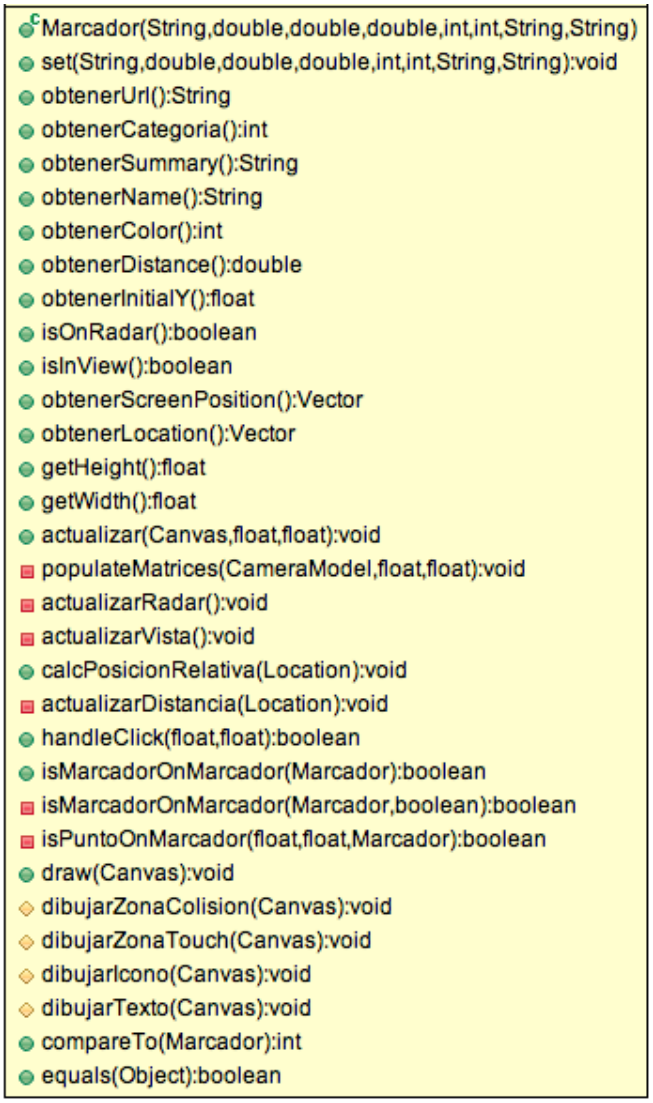

Figura 4.31 Clase Marcador (2)

Clase IconoMarcador

Esta clase dibuja el ícono del marcador. Hereda de la clase Marcador.

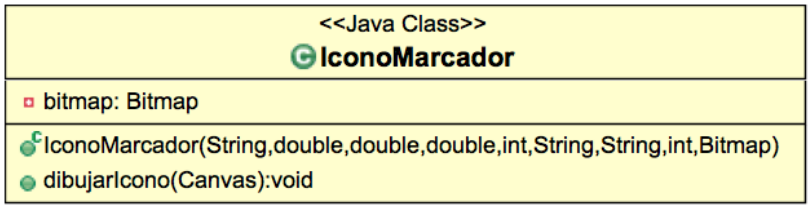

Figura 4.32 Clase IconoMarcador

Clase VerticalSeekBar

Esta clase es una extensión de la clase Android SeekBar. Sirve para mostrar una barra personalizada de forma vertical (Un objeto Android SeekBar por defecto se muestra horizontalmente).

Figura 4.33 Clase VerticalSeekBar

4.3.5 Capa de cámara

Esta capa contiene las clases que interactúan con la cámara y la pantalla del dispositivo.

Clase SuperficieCamara

Esta clase maneja la vista de la cámara que se muestra en la pantalla. Es una extensión de la clase Android SurfaceView.

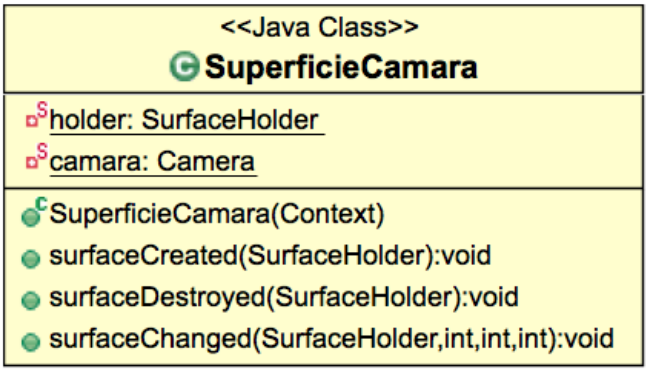

Figura 4.34 Clase SuperficieCamara

Clase CameraCompatibility

Esta clase habilita la actividad del navegador RA para mantener una compatibilidad con todas las versiones de Android y obtiene las limitaciones sobre versiones más antiguas del API de Android. (Clase adaptada del framework Mixare tools mixare.org).

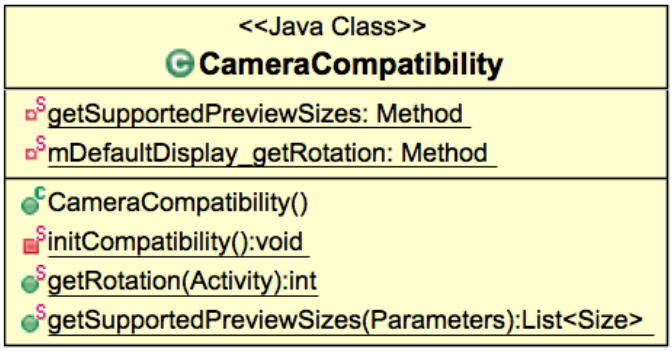

Figura 4.35 Clase CameraCompatibility

Clase CameraModel

Esta clase representa la cámara y la vista. (Clase adaptada del framework Mixare tools mixare.org).

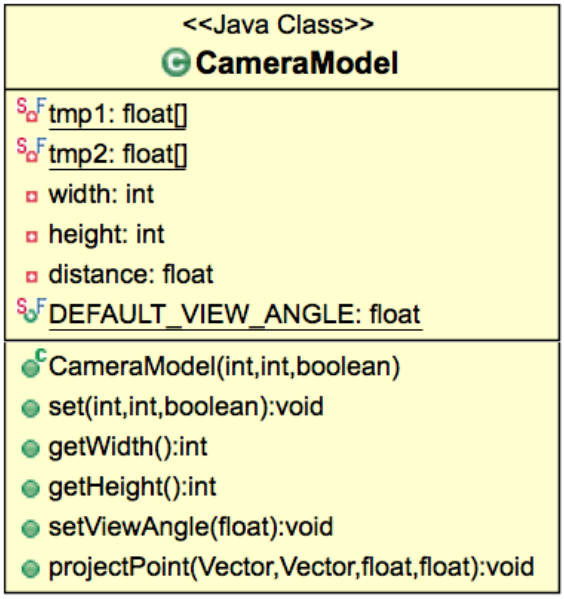

Figura 4.36 Clase CameraModel

4.3.6 Capa Global

Esta capa contiene las clases para almacenar toda la información con la que se interactúa mientras la aplicación está corriendo.

Clase DatosRA

Esta clase es el control global y el almacenamiento de la información del navegador RA. Almacena todos los datos de los marcadores mientras el navegador RA está corriendo.

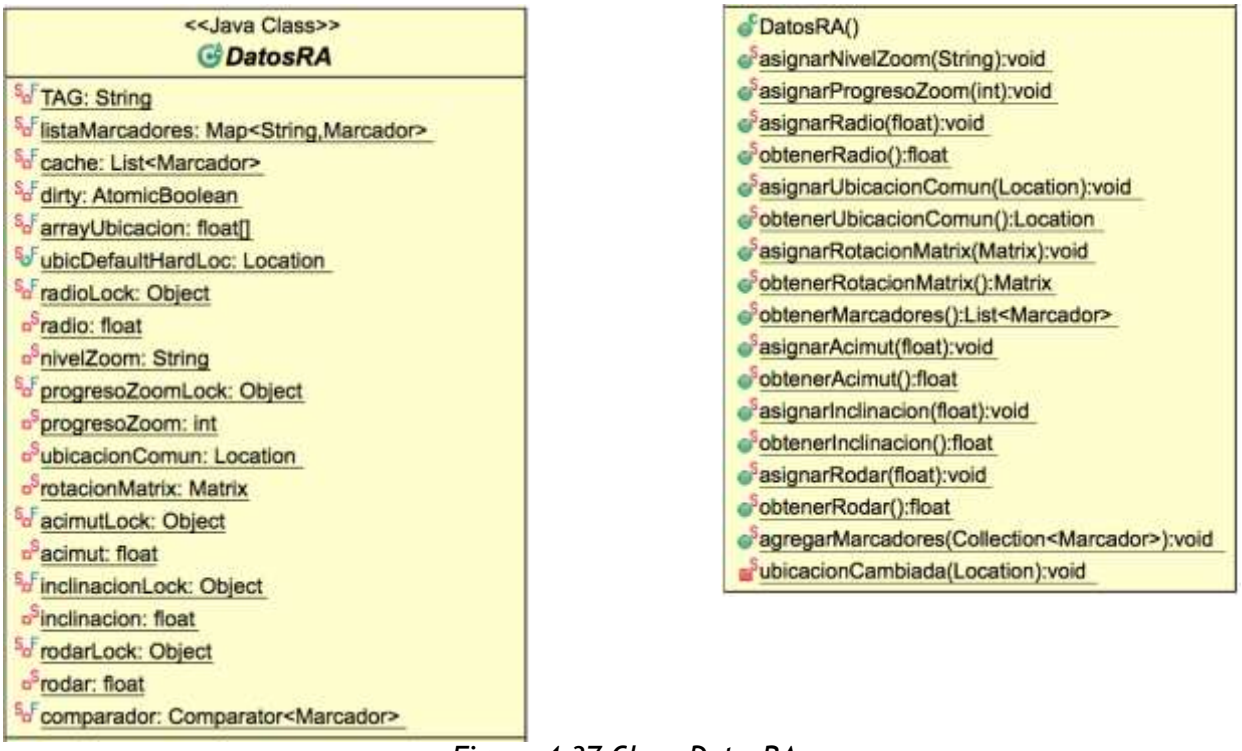

Figura 4.37 Clase DatosRA

4.4 Bloque de visualización RA

4.4.1 Capa Vuforia

Esta capa contiene todas las clases necesarias para mostrar objetos en tres dimensiones a partir del enfoque de la cámara a un marcador natural o target.

Clase InstruccionesCatedral

Esta clase muestra una actividad con las instrucciones de realidad aumentada para la catedral.

Figura 4.38 Clase InstruccionesCatedral

Clase DebugLog

Esta clase gestiona los mensajes de registro del sistema que se generan mientras está corriendo el visualizador de realidad aumentada.

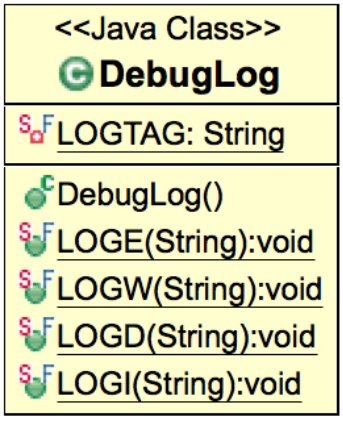

Figura 4.39 Clase DebugLog

Clase ImageTargets

Esta clase es la encargada del procesamiento del target (marcador Vuforia) para mostrar sobre este un objeto en 3D.

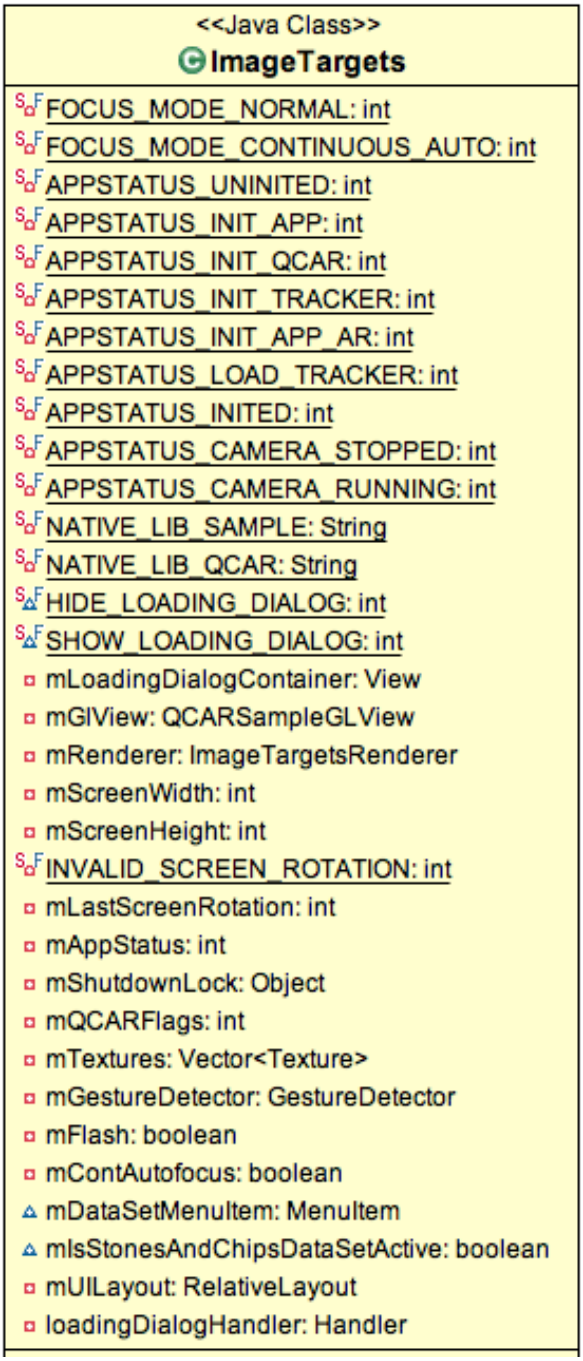

Figura 4.40 Clase ImageTargets (1)

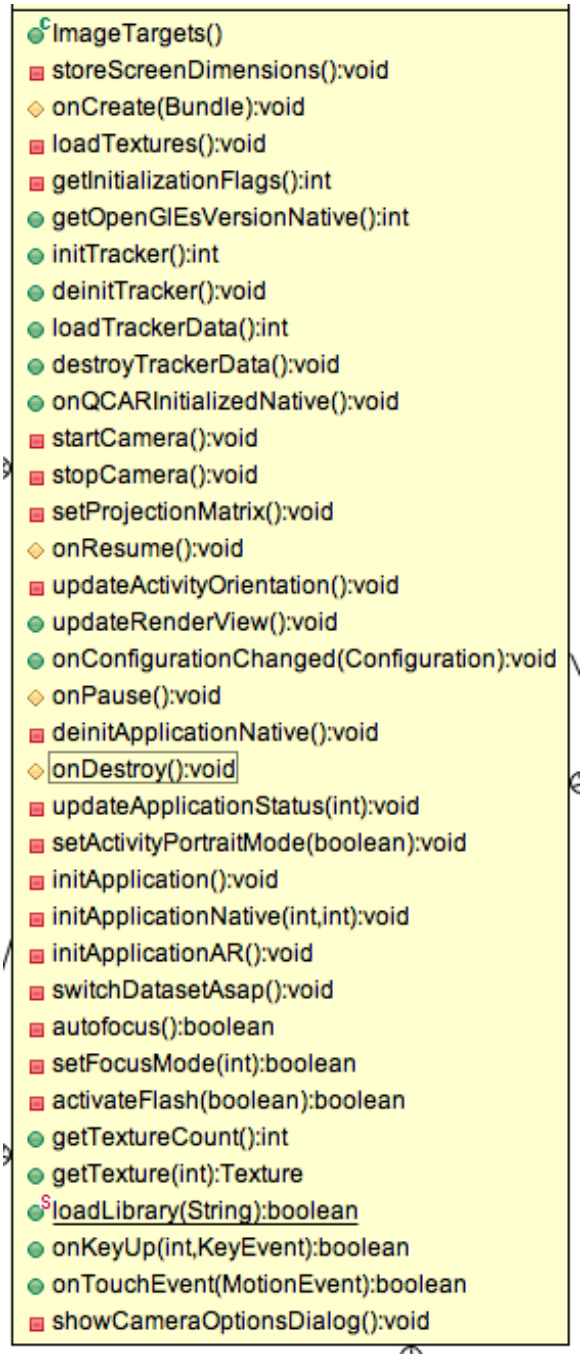

Figura 4.41 Clase ImageTargets (2)

Clase LoadingDialogHandler

Esta clase crea un manejador para actualizar el estado de un Android LoadingDialog mientras se muestra la actividad ImageTargets.

Figura 4.42 Clase LoadingDialogHandler

Clase InitQCARTask

Esta clase tiene la función de crear una tarea asíncrona para inicializar el Api QCAR de Vuforia.

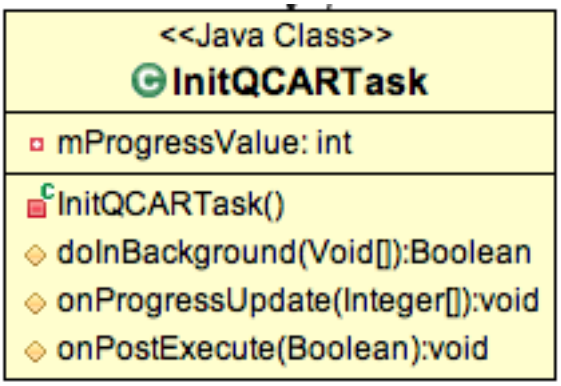

Figura 4.43 Clase InitQCARTask

Clase GestureListener

Esta clase se ocupa de controlar las acciones sobre la pantalla al dar dos golpes con la intención de mostrar las opciones del visualizador RA

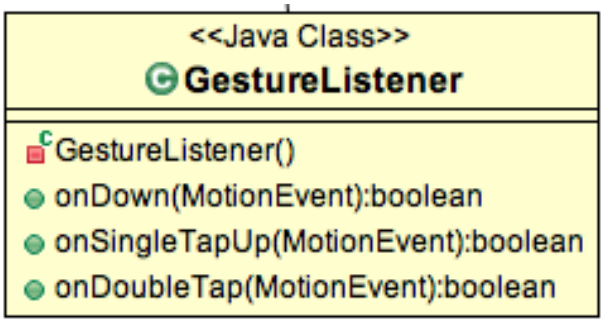

Figura 4.44 Clase GestureListener

Clase LoadTrackerTask

Esta clase tiene la función de crear una tarea asíncrona del rastreador de datos de Vuforia.

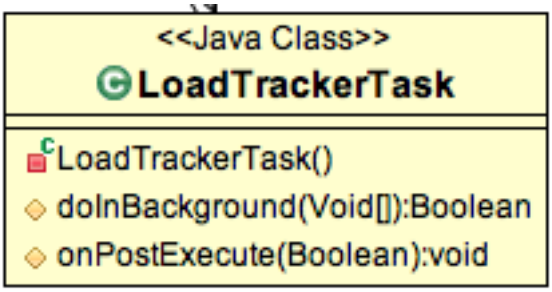

Figura 4.45 Clase LoadTrackerTask

Clase ImageTargetsRenderer

Esta clase es la encargada del render del objeto 3D que se visualiza en la pantalla.

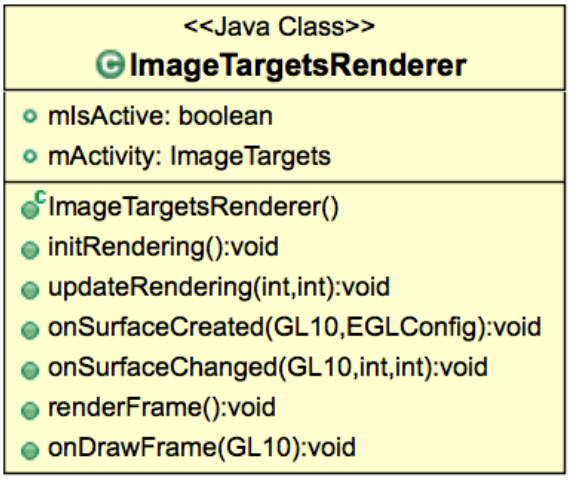

Figura 4.46 ImagetargetsRender

4.5 Bloque de Mapas

4.5.1 Capa Mapas

Esta capa contiene las clases para conectarse al API de Google Maps y mostrar marcadores y rutas en un mapa.

Clase MapaGoogle

Esta clase es la encargada de mostrar el mapa en la pantalla a partir del API de Google Maps. Es capaz de mostrar los marcadores y las rutas para llegar a determinado marcador.

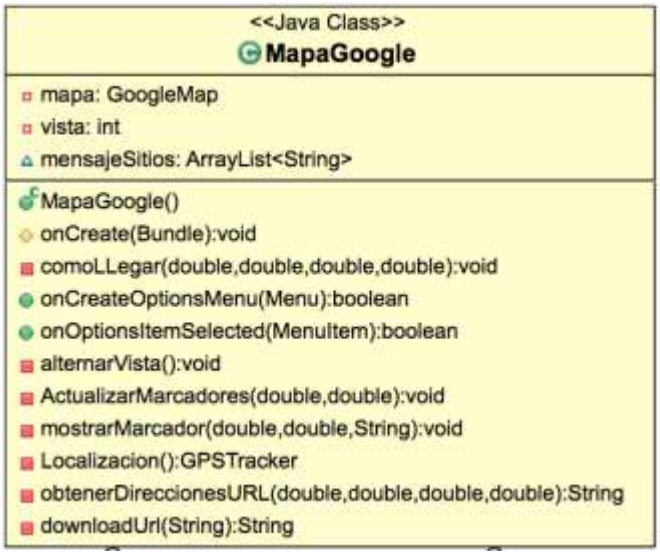

Figura 4.47 Clase MapaGoogle

Clase RastreadorGPS

Esta clase se encarga de manejar las conexiones del GPS y del A-GPS para obtener la ubicación donde se encuentra el dispositivo.

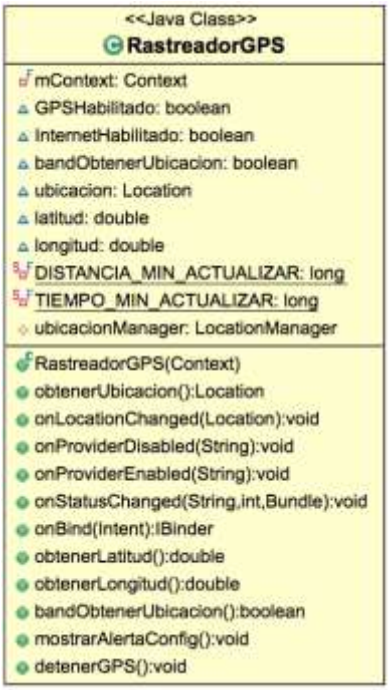

Figura 4.48 Clase RastreadorGPS

Clase DireccionesJSONParser

Esta clase se encarga de convertir las direcciones recibidas del API de rutas de Google Maps en coordenadas para poder trazarlas sobre el mapa.

Figura 4.49 Clase DireccionesJSONParser

4.6 Bloque de Servicio Web

4.6.1 Capa de Servicio Web Rest

Esta capa es la encargada de recibir peticiones y enviar información hacia los clientes que necesiten los marcadores. Realiza el proceso de selección de marcadores y también se encarga de las consultas que se realizan con una base de datos.

Clase PeticionesController

Esta clase se encarga de controlar del servicio web. Contiene las acciones que se pueden llamar según la URL y los datos http que recibimos como parámetros de entrada al servicio.

Figura 4.50 Clase PeticionesController

4.6.2 Capa de Modelos.

Esta capa es contiene las operaciones y consultas que se realizan con la base de datos y de procesar la información que se reciba para devolverla en el formato correcto. Contiene todas las clases que serán serializadas y enviadas al cliente móvil.

Clase Sitios

Esta clase contiene las propiedades de los sitios que serán enviadas como retorno hacia el cliente que solicitó los sitios.

Figura 4.51 Clase Sitio

Clase ManejadorPeticiones

Esta clase contiene todas las operaciones que vamos a realizar sobre la base de datos BDSITIOSTUR

Figura 4.52 Clase ManejadorPeticiones

Clase Geocoordenada

Esta clase solo contiene las propiedades Latitud y Longitud usadas para asignarlas a determinada ubicación.

Figura 4.53 Clase Geocoordenada

Clase geo

Esta clase contiene una lista de objetos Sitios. Dicha clase será instanciada para enviarla como objeto JSON.

Figura 4.54 Clase geo

4.6.3 Capa API

Clase ApiAreaRegistration

Esta clase tiene la función de dirigir las peticiones dirigidas hacia determinada acción del controlador (PeticionesController) según la URL utilizada al realizarse la llamada al servicio web.

Figura 4.55 Clase APiAreaRegistration

4.7 Diseño de la Base de datos

4.7.1 BDSITIOSTUR

La base de datos solo contiene 2 tablas: Sitio, que será la encargada de almacenar todos los sitios que representarán los marcadores en la aplicación cliente, y la tabla categoría, que representa la categoría de cada sitio.

Figura 4.56 Diseño de la base de datos BDSITIOSTUR

4.8 Diagrama de clases del sistema

La figura 1.56 muestra el diagrama de clases del sistema.

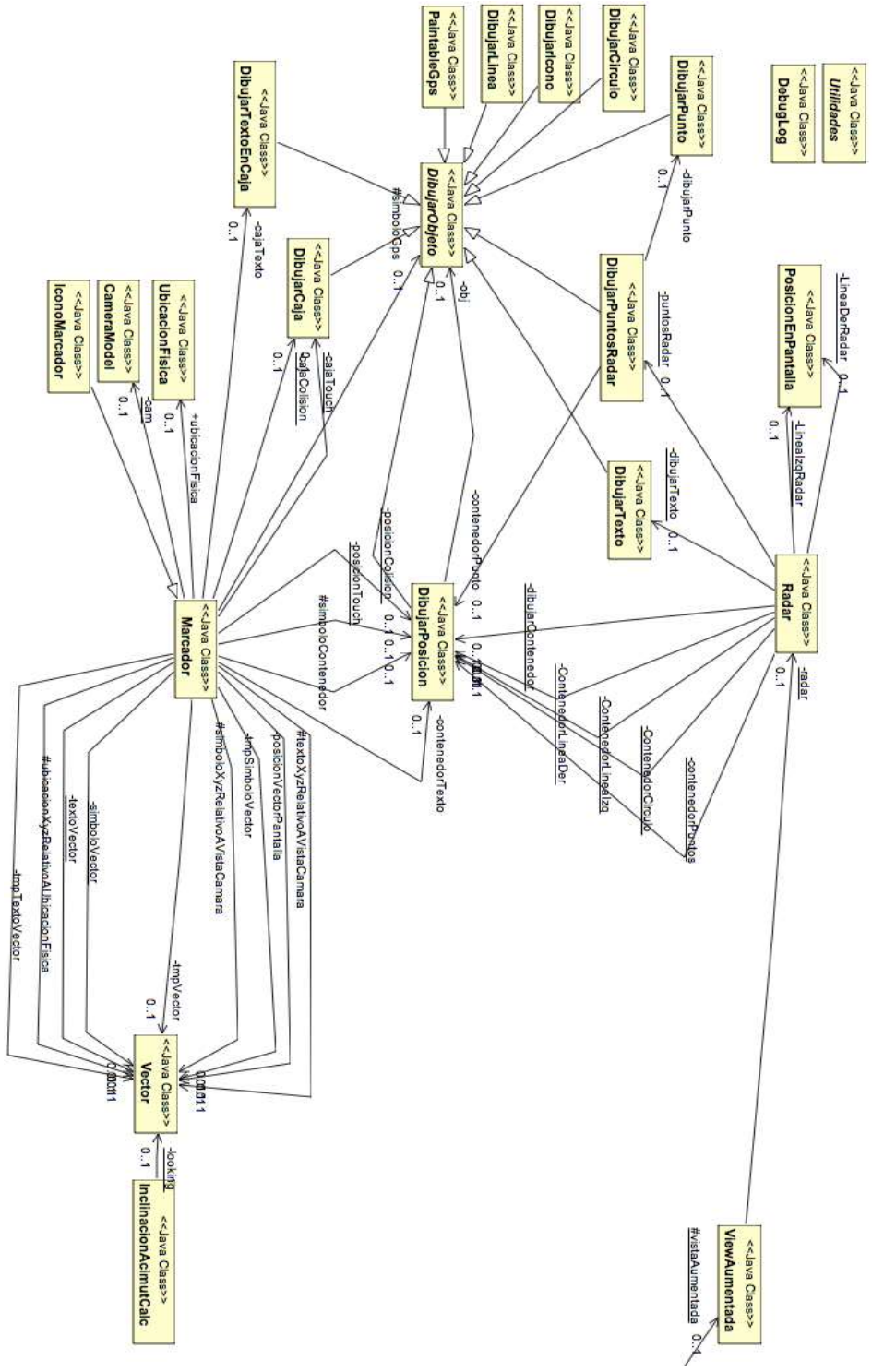

Figura 4.57 Diagrama de clases del sistema (1)

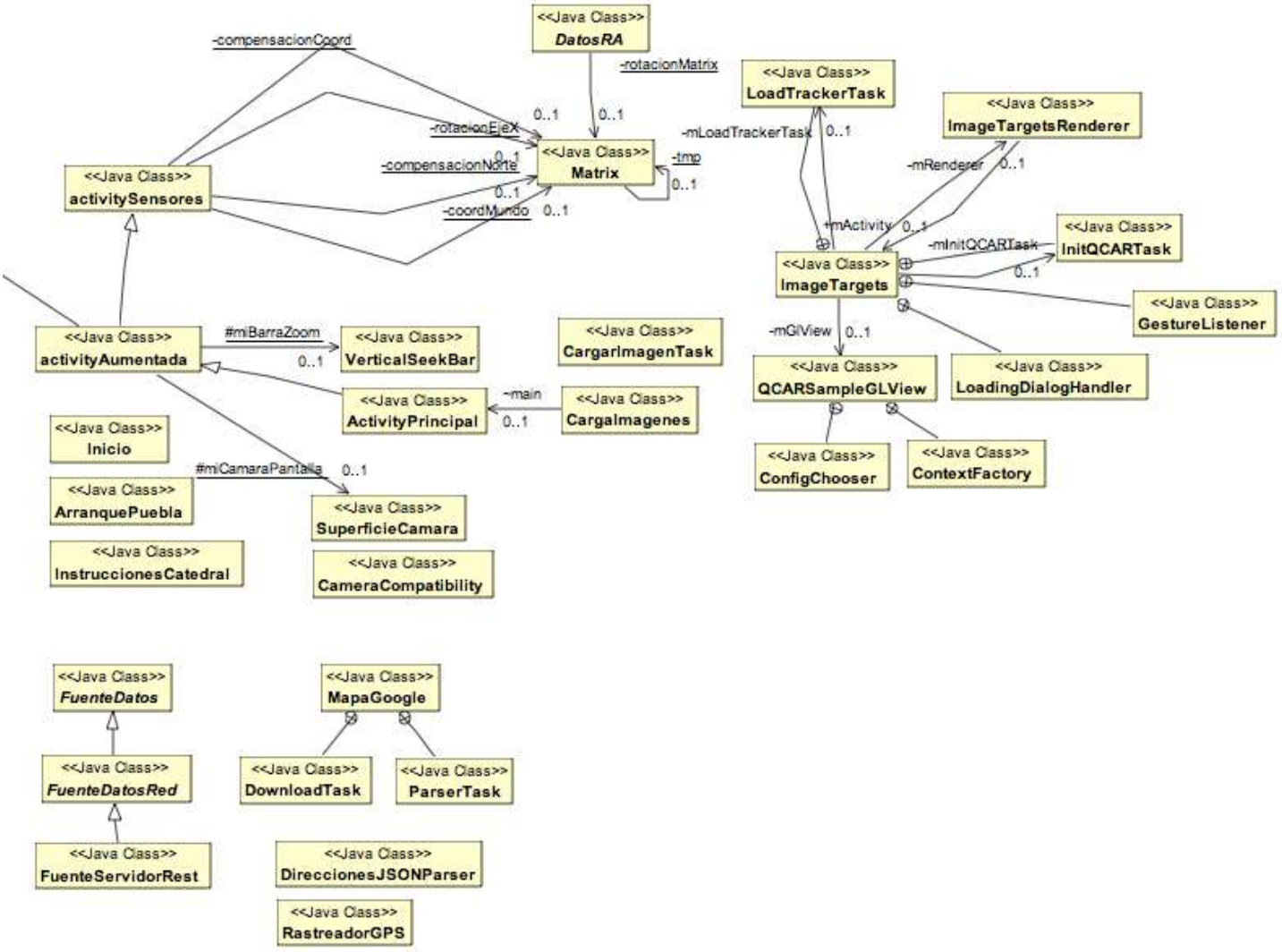

Figura 4.58 Diagrama de clases del sistema (2)

5. Implementación del Sistema

Para la implementación del sistema se tomaron en cuenta distintas tecnologías de desarrollo. En el caso del desarrollo de la aplicación en Android, se programaron las clases en Java y las interfaces en XML con el IDE Eclipse. Para los objetos 3D que se despliegan mientras se observa la realidad aumentada, se usó el lenguaje C++ para programar dicho objetos, además de usar las librerías de Vuforia. En el caso del servicio web, se implementó un Servicio web Rest con las tecnologías de ASP.NET MVC para la programación y SQL Server para el almacenamiento de los sitios turísticos. A continuación, se presenta a detalle las tecnologías elegidas para el desarrollo de este sistema de acuerdo a cada bloque.

Bloque de presentación

- Java. Todas las actividades que muestran las interfaces se programaron en Java, de acuerdo al modelo MVC, las actividades programadas corresponden al controlador de la aplicación Android.
- XML. Las interfaces de la aplicación se programaron en lenguaje de marcas extensible (XML). En esta incluimos todos los objetos como botones, imágenes y etiquetas construidas con XML.

Bloque de Navegación RA

- Java. Los controladores necesarios para obtener información desde el servidor interactuar, con los componentes de hardware del móvil, así como agregar elementos en la pantalla se programaron en Java de acuerdo al Modelo Vista Controlador y utilizando el Framework de Android.
- JSON. Es el formato de intercambio de datos utilizado para la comunicación que existe entre el dispositivo móvil y el servidor web Rest para adquirir los marcadores.

Bloque de Mapas

- Google Maps API. El API de Google Maps se accede desde el móvil para descargar las vistas de mapas. Además se accede al API de rutas de Google Maps para trazar rutas entre dos puntos.
- Java. Se programaron las clases necesarias para mostrar el objeto mapa en el móvil e el lenguaje Java.
- JSON. Se utiliza el formato JSON para acceder al API de rutas de Google Maps con la intención de obtener las rutas más cortas desde el punto donde se encuentra el móvil a otro.

Bloque de visualización RA

- Vuforia. Las librerías de Vuforia son utilizadas para el reconocimiento de targets o marcadores naturales y el renderizado de objetos 3d.
- C++. Se usó lenguaje C++ para obtener los objetos 3D desde las librerías que se crearon con cada objeto 3D.
- XML. Es usado para la interfaz del visualizador de realidad aumentada.
- OpenGL. Son el conjunto de librerías para producir gráficos en dos y tres dimensiones. Se crearon objetos 3D en archivos .h para que sean renderizados por la aplicación.

Bloque de servicio web

- C#. Todos los métodos para acceder a la base de datos, así como las clases necesarias a enviar a un cliente fueron programadas en C#.
- ASP.NET MVC. El Modelo Vista controlador de ASP.NET nos ayudó a establecer las funciones de acceso a los datos a partir de peticiones http.
- SQL Server 2008. El lenguaje SQL se usó para crear la base de datos BDSITIOSTUR así como las tablas que esta base de datos contiene.
- JSON. El formato JSON se usó para serializar un objeto que se envía cada vez que se realiza una petición desde un cliente.

5.1 Herramientas usadas para la implementación

Android Developer Tools. (http://developer.android.com/intl/es/sdk/index.html). Es una herramienta que proporciona Google con todos los elementos necesarios para desarrollar aplicaciones para Android. El paquete incluye los siguientes elementos:

- Java JDK
- SDK de Android
- Eclipse
- Plugin Android para Eclipse (ADT)
- Android Platform-Tools
- API de Android de acuerdo a la versión descargada

SDK Android. El Kit de desarrollo de Android incluye un conjunto de herramientas de desarrollo, las cuales son un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales.

Vuforia Qualcomm AR SDK (QCAR): Rebautizado como Vuforia, es una solución gratuita para la detección y seguimiento de imágenes de referencia y marcadores usando características de detección natural.

NDK Android (Desarrollo nativo). El NDK permite instalar bibliotecas escritas en C y otros lenguajes, una vez compiladas para ARM o código x86 nativo. Los programas Java corriendo en la máquina virtual Dalvik (Dalvik VM) pueden llamar a clases nativas por medio de la función System.loadLibrary, que forma parte de las clases estándar Java en Android.

Blender. Es un programa que se distribuye como Open Source dedicado al modelado de animaciones y creación de gráficos tridimensionales.

OBJ2OPENGL. Esta herramienta convierte modelos realizados en Blender con la extensión .obj en archivos .h C/C++ que describen los vértices, caras, normales y coordenadas de texturas en arreglos de flotantes. Fue desarrollada en PERL por Heiko Behrens (http://heikobehrens.net/2009/08/27/obj2opengl/).

Microsoft Visual Studio 2012. Es un entorno de desarrollo integrado (IDE) que soporta los lenguajes C#, Visual Basic, C++, entre otros. Este IDE se usa para desarrollar aplicaciones de escritorio,

aplicaciones para dispositivos móviles así como en Web usando las tecnologías ASP.NET.

Microsoft SQL Server 2008. Es el motor de base de datos de Microsoft. Su lenguaje de consulta es Transact-SQL y está basado en el modelo relacional.

5.2 Desarrollo del sistema

De acuerdo al diagrama de bloques del sistema que se muestra en la figura 4.1 se implementaron cada uno de los bloques, de los cuales, los cuatro primeros contienen las clases para la aplicación móvil, mientras que el último bloque contiene las clases para el servidor web Rest.

5.2.1 Implementación del bloque de presentación

5.2.1.1 Capa de vista aumentada y actividades

Implementación de actividades

En Android se conocen como activities a las clases públicas que representan cada una de las pantallas de una aplicación. Todas las actividades en Android funcionan dentro de una pila. Una actividad sale de la pila por medio del botón Back. Las actividades están estrechamente relacionadas con las interfaces que se desarrollan en XML. En la figura 1.1 se muestra el ciclo de vida de una actividad en Android.

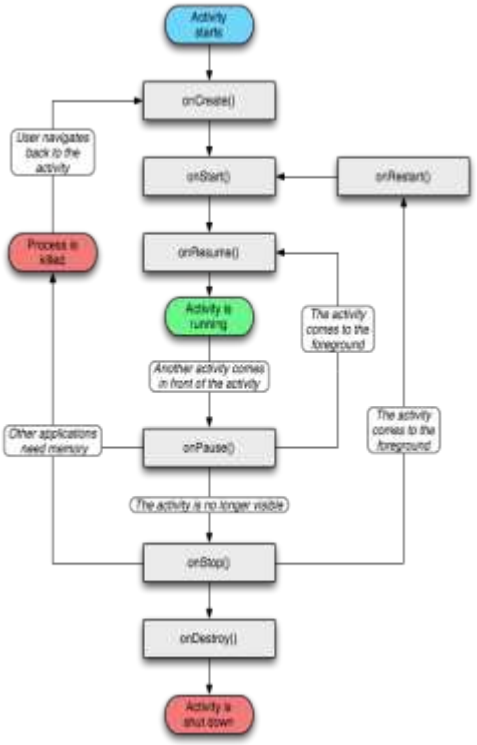

Figura 5.1 Ciclo de vida de las actividades en Android.

A continuación se describen las siete actividades principales (ArranquePuebla, Inicio, InstruccionesCatedral, AcercaDe, MapaGoogle, ActivityPrincipal e ImageTargets) mostradas en la figura 4.2 del capítulo diseño del sistema.

Actividad: Arranque Puebla

Descripción: Esta Actividad es mostrada cada vez que se inicia la aplicación en el móvil, programada en la clase ArranquePuebla. Su objetivo es mostrar el nombre de la aplicación con un ícono de la misma durante dos segundos. También muestra la marca registrada Vuforia con fines de derechos reservados.

Interfaz: La interfaz es generada en xml en el archivo splashArranquePuebla.xml (Figuras 5.2 y 5.3).

Figura 5.2 Interfaz de la actividad ArranquePuebla

Figura 5.3 Interfaz en modo land (apaisado) de la actividad ArranquePuebla

Actividad: Inicio

Descripción: Esta actividad contiene el menú general de la aplicación. Fue programada en la clase

Inicio. Existen tres opciones de navegación:

- Navegador RA: Esta acción envía a la actividad denominada ActivityPrincipal, la cual es la actividad que controla la navegación con realidad aumentada (ver ActivityPrincipal)
- Catedral: Esta acción envía a la actividad Instrucciones catedral (ver ActivityInstrucciones)
- Acerca de… Esta acción envía a la actividad AcercaDe, que contiene información acerca del sistema (Ver ActivityAcercaDe)

Interfaz: En la figura 5.4 y 5.5 se muestra la actividad, en la cual, la interfaz fue diseñada en XML, y se encuentra en el archivo Inicio.xml

Figura 5.4 Interfaz de la actividad Inicio

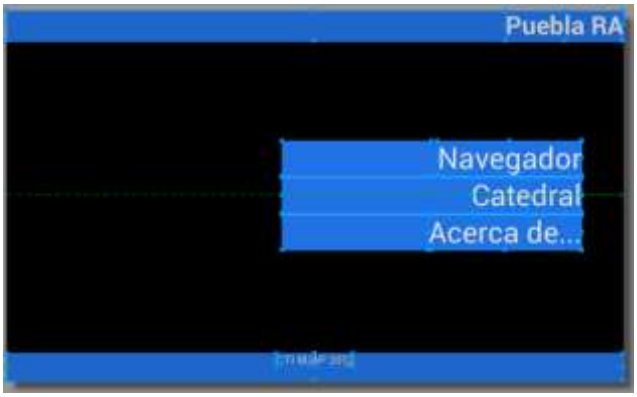

Figura 5.5 Interfaz en modo land (apaisado) de la actividad Inicio

Actividad: InstruccionesCatedral

Descripción: Esta actividad contiene las instrucciones necesarias para poder visualizar la realidad aumentada en la catedral y un botón llamado 'Iniciar', el cual envía a la actividad ImageTargets (ver Bloque de visualización RA). Fue programada en la clase InstruccionesCatedral.

Interfaz: En la figura 5.6 y 5.7 se muestra la actividad, en la cual, la interfaz fue diseñada en XML y se encuentra en el archivo InstruccionesCatedral.xml

Figura 5.6 Interfaz de la actividad InstruccionesCatedral

Figura 5.7 Interfaz de la actividad InstruccionesCatedral en modo land (apaisado)

Actividad: ActivityPrincipal

Descripción: Esta actividad es la encargada de controlar la navegación por realidad aumentada. Contiene los componentes definidos en la capa de componentes así como la visualización de la cámara. Fue programada en la clase ActivityPrincipal.

Interfaz: La interfaz es generada en tiempo de ejecución de acuerdo a las clases activityAumentada (ver bloque de navegación RA). En la figura 5.8 se muestra la interfaz de la actividad ActivityPrincipal.

Figura 5.8 Interfaz de la actividad ActivityPrincipal

Actividad: MapaGoogle

Descripción: Esta actividad es la encargada de obtener las capas de mapas desde el API de GoogleMaps y mostrarla al usuario de acuerdo a su ubicación y los marcadores que se hayan solicitado. Fue programada en la clase MapaGoogle

Interfaz: La interfaz se genera en tiempo de diseño de acuerdo a la clase SupportMapFragment del Framework de Android. En la figura 5.9 se muestra la interfaz de la actividad MapaGoogle.

Figura 5.9 Interfaz de la actividad MapaGoogle

Actividad: ImageTargets

Descripción: Esta actividad es la encargada de controlar el procesamiento de imágenes en tiempo

real para superponer objetos en 3D sobre la búsqueda de un marcador natural. Está programada en la clase ImageTargets y usa las clases de Vuforia para su funcionamiento

Interfaz: La interfaz se genera en tiempo de ejecución de acuerdo a la captura de imágenes y el procesamiento de las mismas. Superpone los objetos 3D renderizados a partir del marcador que se obtenga por medio de la cámara del dispositivo. La figura 5.10 muestra la interfaz de la actividad ImageTargets.

Figura 5.10 Interfaz de la actividad ImageTargets mostrando un menú de dialogo.

5.2.1.2 Capa de obtención de datos

Para la obtención de datos se crearon tres clases: FuenteDatos, FuenteDatosRed y FuenteServidorRest.

- **FuenteDatos**: Su única función es obtener una lista de marcadores. Es heredada por las clases FuenteDatosRed y FuenteServidorRest
- **FuenteDatosRed**: Contiene los elementos básicos para obtener la información desde un servidor. También tiene el número máximo de marcadores que se deben mostrar. A continuación se muestran sus atributos y métodos:

  Atributos

```
String TAG : Etiqueta de control del LogCat ("FuenteDatosServidorRest").
int MAX = 1000 : Tiempo máximo en milisegundos para espera.
int TIEMPO_LECTURA = 10000 :    Tiempo máximo de lectura en milisegundos
int TIEMPO_CONECTAR = 10000 : Tiempo máximo de conexión en milisegundos.

List<Marcador> marcadoresCache : Lista de marcadores que se almacenan en el caché
```

`String crearPeticionURL(double lat, double lon, float radius) : Creación de la petición url.`

`List<Marcador> parse(JSONObject root) : Lista de Marcadores devueltos por el objeto parse enviando como parámetro un objeto JSON.`

Métodos

`obtenerHttpInputStream (String urlStr) : Este método es usado para obtener un objeto InputStream para la URL del servidor especificado, el cual es obtenido a partir cadena urlStr.`

`obtenerHttpInputString(InputStream is) : En este método usamos un objeto InputStream para ponerlo en un objeto StringBuilder`

`parse(String url) : En este método obtenemos un objeto JSON desde nuestra fuente de datos, en este caso, FuenteServidorRest.`

- **FuenteServidorRest**: Usamos un servicio web almacenado en la dirección `http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/` para obtener la lista de marcadores que se encuentren cercanos a nuestra ubicación. A continuación se describen los métodos de la clase:

  Atributos

  `String BASE_URL_SITOS_PERSONALIZADOS = "http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/": Cadena de conexión al servicio web Rest.`

  `Bitmap icon : Almacena un ícono desde una imagen de mapa de bits.`
  `Resources res : Almacena los recursos del sistema (Clase R).`

  Métodos

  `crearPeticionURL(double lat, double lon, float radio)`, Este método se usa para formar la cadena que solicita los marcadores al servicio web. Para ello, concatenamos la dirección de nuestro servicio web con los parámetros lat (latitud), lon (longitud) y radio (el radio en kilómetros al que deseamos obtener marcadores). Una solicitud válida para el servicio web queda de la siguiente manera:

  `http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/19.00491,-98.20483,1`

  `parse(JSONObject root)`, Este método es una sobrecarga de la clase FuenteDatosRed, a partir de un objeto JSONObject verifica que el objeto recibido sea correcto y devuelve la lista de marcadores con aquellos que sean válidos para usarlos en el sistema.

  `procesarJSONObject(JSONObject jo)`, Este método se encarga de procesar cada uno de los objetos recibidos mediante un objeto JSONObject, para ello, es necesario mapear el objeto

con el fin de obtener los siguientes atributos:

- "tittle"
- "lat"
- "lng"
- "elevation"
- "id_categoria"
- "summary"
- "url_imagen"

El método devuelve un marcador en caso de que el objeto sea correcto.

`ElegirIcono(int categoria)`, Este método asigna un Icono de acuerdo al identificador de la categoría del marcador recibido. Existen once categorías entre los marcadores las cuales se muestran a continuación:

- Monumento
- Iglesia
- Parque
- Museo
- Plaza
- Entretenimiento
- Biblioteca
- Negocio
- Sitio histórico
- Fuente
- Otro

5.2.2 Implementación del bloque de navegación RA

5.2.2.1 Capa de posicionamiento

- **Asignación de la ubicación física** (Clase UbicacionFisica): Esta clase es usada para saber la ubicación física en el mundo del dispositivo. Contiene seis atributos:

  Atributos

  ```
  latitud : contiene la latitud de la ubicación.
  longitud : contiene la longitud de la ubicación.
  altitud : contiene la altitud de la ubicación.

  x : almacena la posición tridimensional x.
  y : almacena la posición tridimensional y.
  z : almacena la posición tridimensional z.
  ```

  Métodos

  Esta clase contiene tres métodos para asignar la latitud, longitud y altitud respectivamente:

```
asignarLatitud(double latitud)
asignarLongitud(double longitud)
asignarAltitud(double altitud)
```

La clase contiene también tres métodos para obtener la latitud, longitud y altitud respectivamente:

```
obtenerLatitud()
obtenerLongitud()
obtenerAltitud()
```

El método `convUbicacionAVector(Location org, UbicacionFisica gp, Vector v)` obtiene la distancia aproximada entre dos ubicaciones.

- **Asignación de posición en pantalla de objetos** (Clase PosicionEnPantalla): Esta clase es usada para desplegar las líneas del radar en pantalla. Contiene dos tributos, *x* y *y*, que son usados para asignarlos a dos puntos en una línea.

```
x : contiene los métodos asignar y obtener.
Y : contiene los métodos asignar y obtener.
```

El método `rotar(double t)` El método rota los valores de *x* y *y* alrededor de un ángulo t.

5.2.2.2 Capa de UI

- **Clases de dibujo** (Clase Dibujar Objeto): Esta es la clase base para todos los pedazos de los componentes de la interfaz. Esta clase contiene los métodos para dibujar cajas, textos, imágenes, círculos, iconos, líneas y puntos.

Atributos

```
paint
```

Métodos

```
getWidth(): Método para obtener el ancho del objeto. Es sobrecargado por otras clases para dibujar.

getHeight():Método para obtener la altura del objeto. Es sobrecargado por otras clases para dibujar.

setColor(int c): Método para obtener el color de acuerdo a un atributo entero. Es sobrecargado por otras clases para dibujar.

setStrokeWidth(float w): Habilita el suavizado de dibujo de un objeto.

getTextWidth(String txt): Obtiene el ancho del texto al que toma como argumento.

getTextAsc(): Obtiene el texto de forma ascendente.
```

getTextDesc(): Obtiene el texto de forma descendente.

setFontSize(float size): Asigna el tamaño del texto de acuerdo al parámetro size.

pintarLinea(Canvas canvas, float x1, float y1, float x2, float y2): Dibuja una línea de acuerdo a las coordenadas que toma de los parámetros .

pintarRecta(Canvas canvas, float x, float y, float width, float height): Dibuja una recta de acuerdo a la posición que toma de los parámetros.

pintarCircunferencia(Canvas canvas, float x, float y, float width, float height): Dibuja un circunferencia de acuerdo a los parámetros que toma.

pintarBitmap(Canvas canvas, Bitmap bitmap, Rect src, Rect dst): Dibuja una imagen desde un Mapa de bits que toma de los argumentos en las coordenadas.

pintarBitmap(Canvas canvas, Bitmap bitmap, float left, float top): Dibuja una imagen desde un Mapa de bits que toma de los argumentos en las coordenadas.

pintarCirculo(Canvas canvas, float x, float y, float radius) Dibuja una circulo de acuerdo a las coordenadas y el radio que toma de los argumentos.

pintarTexto(Canvas canvas, float x, float y, String text): Pinta o escribe el texto que se pasa como parámetro en las coordenadas específicas.

- **Caja** (Clase DibujarCaja): Esta clase es usada para dibujar una caja con el borde y el color especificados.

  Atributos

  width: contiene el ancho de la caja.
  height: contiene el alto de la caja.
  borderColor: establece el color del borde.
  backgroundColor: establece el color de fondo de la caja.

  Métodos

  set(float width, float height): Asigna el alto y ancho del objeto caja tomando en cuenta los valores por defecto del color de borde y del color de fondo.

  set(float width, float height, int borderColor, int bgColor): Asigna el ancho, el alto, el color de borde y el color de fondo de la caja de acuerdo a los parámetros.

  paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se trazan las líneas y se asigna el color de las mismas así como el color de fondo de la caja.

  getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.

  getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

En la figura 5.11 se observa una caja dibujada sobre la interfaz de navegación RA.

Figura 5.11 Caja con los atributos borderColor: BLUE y background: BLUE

- **Texto en caja** (Clase DibujarTextoEnCaja): Dibuja o escribe el texto dentro de una caja.

  Atributos

  Width: Ancho del texto.
  Height: Alto del texto.
  areaWidth: Ancho del área de la caja que contiene el texto.
  areaHeight: Alto del área de la caja que contiene el texto.
  lineList: Lista de líneas que contendrá la caja de texto.
  lines: Línea en la que se escribe el texto.
  lineWidths: Ancho de la línea.
  lineHeight: Alto de la línea.
  maxLineWidth: Almacena las líneas que se van creando.
  pad: Almacena el texto de forma ascendente

  txt: Almacena el texto a escribir en la caja.
  fontSize: Tamaño del texto.
  borderColor: Color de borde de la caja.
  backgroundColor: Color de fondo de la caja.
  textColor: Color del texto.

  Métodos

  set(String txtInit, **float** fontSizeInit, **float** maxWidth, **int** borderColor, **int** bgColor, **int** textColor) : Este método asigna el texto, el tamaño del texto, el ancho del área de la caja,el color de borde, el color de fondo y el color de texto del objeto.

  set(String txtInit, **float** fontSizeInit, **float** maxWidth) : Este método asigna el texto, el tamaño del texto, y el ancho del área de la caja.

  prepTxt(String txtInit, **float** fontSizeInit, **float** maxWidth) : Este método es el encargado de preparar el texto para ser mostrado en diferentes líneas de acuerdo al tamaño de la caja contenedora.

  paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se trazan las líneas y se asigna el color de las mismas así como el color de fondo de la caja. Por defecto, las líneas y el color de fondo son transparentes, puesto que se superponen sobre un objeto DibujarCaja. El color de las letras es blanco.

`getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.`

`getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.`

En la figura 5.12 se observa el texto dibujado sobre una caja.

Figura 5.12 Texto en una caja con parámetros de un marcador.

- **Círculo** (Clase DibujaCirculo): Esta clase dibuja un círculo. Contiene los siguiente atributos y métodos:

  Atributos

  ```
  int color: Color del círculo.
  float radio: Radio del círculo
  boolean fill: Booleano con el estilo de línea. Por default se establece true (FILL)
  ```

  Métodos

  `set(int color, float radius, boolean fill): Este método asigna el color, tamaño del radio y el estilo de línea del objeto.`

  `paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se traza el círculo.`

  `getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.`

  `getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.`

  En la figura 5.13 se observa un círculo dibujado sobre la interfaz de navegación RA.

Figura 5.13 Círculo dibujado para simular un Radar.

- **Ícono** (Clase DibujarIcono): Esta clase es la encargada de dibujar el ícono del marcador. Lo

hace a través de una imagen de mapa de bits. Contiene los siguientes atributos y métodos.

Atributos

`Bitmap bitmap`: Objeto que contendrá la imagen a ser mostrada.

Métodos

`set(Bitmap bitmap, int width, int height)`: Este método asigna el bitmap, el ancho y el alto del ícono a dibujar.

`paint(Canvas canvas)`: Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se dibuja el mapa de bits en el canvas de acuerdo al ancho y alto especificados.

`getWidth()`: Sobrecarga del método getWidth de la clase base DibujarObjeto.

`getHeight()`: Sobrecarga del método getHeight de la clase base DibujarObjeto.

En la figura 5.14 se observa un ícono para una categoría.

Figura 5.14 Ejemplo de ícono de un marcador de la categoría museo.

- **Línea** (Clase DibujarLinea): Esta clase sirve para dibujar una línea. Contiene los siguientes atributos y métodos:

  Atributos

  `int color`: Color de la línea.
  `float x`: Valor para el punto o coordenada x de la línea.
  `float y`: Valor para el punto o coordenada y de la línea.

  Métodos

  `set(int color, float x, float y)`: Este método asigna el color, la coordenada x y la coordenada y de la línea.

  `paint(Canvas canvas)`: Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se dibuja una línea en el canvas de acuerdo al punto x y el punto y especificados.

  `getWidth()`: Sobrecarga del método getWidth de la clase base DibujarObjeto.

getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

En la figura 5.15 se observan las líneas trazadas sobre el Radar.

Figura 5.15 Ejemplo de dos líneas trazadas sobre un círculo.

- **Punto** (Clase DibujarPunto): Esta clase dibuja un punto que muestra la posición de un marcador dentro del radar. Contiene los siguientes atributos y métodos:

  Atributos

  ```
  int width: Ancho del punto.
  int height: Alto del punto.
  int color: Color del punto.
  boolean fill: Booleano con el estilo del punto. Por default se establece true
  (FILL).
  ```

  Métodos

  set(int color, boolean fill): Este método asigna el color, y el estilo de punto.

  paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se dibuja un punto en el canvas, específicamente dentro del objeto Radar.

  getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.

  getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

- **Posición** (Clase DibujarPosicion): Esta clase tiene la particularidad de rotar y escalar los objetos que se han pintado dentro de un canvas. Contiene los siguiente atributos y métodos

  Atributos

  ```
  float  width: Ancho del objeto
         height: Altura del objeto:
  float  objX: Coordenada x del objeto.
         objY: Coordenada y del objeto.
         objRotacion: Valor de rotación del objeto.
         objEscala: Valor de escalamiento del objeto.
  ```

DibujarObjeto obj: Instancia de la clase DibujarObjeto.

Métodos

move(float x, float y): Este método permite mover el objeto a las nuevas coordenadas x y y.

getObjectsX(): Obtiene la coordenada en x del objeto.

getObjectsY(): Obtiene la coordenada en y del objeto.

paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se dibuja el objeto en el canvas de acuerdo al punto x y el punto y nuevos así como tomando la rotación y el escalamiento realizado sobre el objeto.

getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.

getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

- **Puntos en Radar** (Clase DibujarPuntosEnRadar): Esta clase usada para dibujar puntos que corresponden a la posición relativa de los marcadores dentro de un radar. Consta de los siguientes atributos y métodos.

Atributos

float[3] locationArray: Ubicación relativa de un marcador
DibujarPunto dibujarPunto: Instancia de la clase DibujarPunto.
DibujarPosicion contenedorPunto: Instancia de la clase contenedorPunto.

Métodos

paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se obtienen todos los marcadores que estén disponibles y establecer su ubicación física de acuerdo a un vector de asignación. La ubicación física es asignada con sus coordenadas x y y para trazarlas dentro de un circulo que simula un radar.

getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.

getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

En la figura 5.16 se observan las líneas, puntos y círculo usados para formar un Radar.

Figura 5.16 Ejemplo de puntos trazado sobre el círculo

- **Texto** (Clase DibujarTexto). Esta clase es usada para desplegar texto en el radar. Con esta clase se muestra la orientación del dispositivo y el radio en Km elegido.

  Atributos

  ```
  String texto: Texto a mostrar.
  int color: Color del texto.
  int size: Tamaño del texto.
  float width: Anchura del contenedor del texto.
  float height: Altura del contendor del texto.
  boolean bg: Variable booleana para el color de fondo del contenedor de texto.
  ```

  Métodos

  set(String text, int color, int size, boolean paintBackground): Este método asigna el texto, el color del texto, el tamaño del texto y la variable booleana para elegir si el contenedor de texto tendrá un color de fondo.

  paint(Canvas canvas): Es una sobrecarga de método paint de la clase base DibujarObjeto. Con este método se dibuja el texto dentro del radar de acuerdo a su color y tamaño. También se elige si lleva un color de fondo de acuerdo a la bandera booleana denominada bg.

  getWidth(): Sobrecarga del método getWidth de la clase base DibujarObjeto.

  getHeight(): Sobrecarga del método getHeight de la clase base DibujarObjeto.

  En la figura 5.17 se observa el texto para la parte superior del radar donde muestra la orientación del dispositivo. En la figura 5.18, se puede ver el texto dibujado para mostrar el radio en kilómetros para el radar.

Figura 5.17 Ejemplo de Texto en Radar.

Figura 5.18 Texto con los kilómetros del radar.

- **Cargar Imagen** (Clase CargarImagenTask) Esta clase corresponde a una tarea asíncrona que se emplea para mostrar información acerca de un sitio turístico mientras se carga una imagen desde el servidor. Contiene los siguiente atributos y métodos:

  Atributos

  `WeakReference<ImageView> imagenRef:` Es una referencia a la vista que recibe la imagen desde el servidor web.

  Métodos

  `Bitmap doInBackground(String... params):` Este método ejecuta la descarga desde el servidor de la imagen en un proceso en Background.

  `onPostExecute(Bitmap bitmap):` Este método es llamado una vez que se completó la descarga de la imagen desde el servidor y la muestra en pantalla.

  En la figura 5.19 se muestra el ícono que aparece mientras se descarga una imagen desde una dirección especificada. En la figura 5.20, se puede ver la imagen una vez que ya ha sido descargada.

Figura 5.19 Icono mostrado mientras se descarga la imagen

Figura 5.20 Imagen mostrada una vez que es descargada.

5.2.2.3 Capa de Componentes

- **Radar** (Clase Radar). Esta clase es la encargada de mostrar el Radar en la pantalla. Consta

de los siguientes atributos y métodos.

int *COLOR_LINEA*: Color de las líneas del radar
float *PAD_X* : X
float *PAD_Y* = 20;
int *COLOR_RADAR* : Color de fondo del radar.
int *COLOR_TEXTO* : Color del texto del radar.
int *TAM_TEXTO* : Tamaño del texto del radar.

PosicionEnPantalla *LineaIzqRadar* : Instancia de la clase PosicionEnPantalla para trazar una línea desde el centro del círculo del radar hacia la izquierda.

PosicionEnPantalla *LineaDerRadar* : Instancia de la clase PosicionEnPantalla para trazar una línea desde el centro del círculo del radar hacia la derecha.

DibujarPosicion *ContenedorLineaIzq* : Instancia de la clase DibujarPosicion usada para dibujar un contenedor para la línea izquierda del radar.

DibujarPosicion *ContenedorLineaDer* : Instancia de la clase DibujarPosicion usada para dibujar un contenedor para la línea derecha del radar.

DibujarPosicion *ContenedorCirculo* : Instancia de la clase DibujarPosicion usada para dibujar un contenedor para el círculo del radar.

DibujarPuntosRadar *puntosRadar* : Instancia de la clase DibuparPuntosRadar que es usada para dibujar los puntos dentro del círculo del radar.

DibujarPosicion *contenedorPuntos* Instancia de la clase DibujarPosicion que es usada para dibujar un contenedor para los puntos dentro del radar.

DibujarTexto *dibujarTexto* : Instancia de la clase DibujarTexto que es usada para dibujar o escribir el texto dentro del radar.

DibujarPosicion *dibujarContenedor* : Distancia de la clase DibujarPosicion que es usada para dibujar un contenedor para el componente Radar.

Métodos

void draw(Canvas canvas): Este método es usado para iniciar el proceso de dibujar los elementos dentro del Radar. Obtiene la inclinación del dispositivo con respecto al horizonte y el punto acimut. Después, llama a los métodos de dibujo en el siguiente orden: dibujarCirculoRadar(canvas), dibujarPuntosRadar(canvas), dibujarLineasRadar(canvas), dibujarTextoRadar(canvas);

void dibujarCirculoRadar(Canvas canvas): Este método es usado para dibujar el círculo del radar dentro de un contenedor. Crea un objeto DibujarCirculo con sus respectivos atributos.

void dibujarPuntosRadar(Canvas canvas): Este método es usado para dibujar los puntos que equivalen a los marcadores en el radar dentro de un contenedor. Crea un objeto DibujarPuntos con sus respectivos atributos.

void dibujarLineasRadar(Canvas canvas): Este método es usado para dibujar las líneas del radar dentro de un contenedor. Se crean dos objetos DibujarLinea, cada uno de ellos para la línea izquierda y la línea derecha respectivamente.

void dibujarTextoRadar(Canvas canvas): Este método es usado para mostrar el texto del radar. Toma un rango de acuerdo al valor acimut (la orientación del dispositivo en grados) y muestra el valor del punto cardinal al que se apunta. De acuerdo al rango en grados, se elige el punto cardinal de la siguiente manera:

```
String  dirTxt = "";
if (range == 15 || range == 0) dirTxt = "N";
else if (range == 1 || range == 2) dirTxt = "NE";
else if (range == 3 || range == 4) dirTxt = "E";
else if (range == 5 || range == 6) dirTxt = "SE";
else if (range == 7 || range == 8) dirTxt= "S";
else if (range == 9 || range == 10) dirTxt = "SO";
else if (range == 11 || range == 12) dirTxt = "O";
else if (range == 13 || range == 14) dirTxt = "NO";
```

Después se muestran los grados de orientación junto con el punto cardinal llamando al método textoRadar.

También este método se encarga de mostrar el radio en metros o kilómetros para ver los marcadores dentro de ese rango de alcance.

void textoRadar(Canvas canvas, String txt, **float** x, **float** y, **boolean** bg): Este método es el encargado de dibujar el texto dentro de un contenedor en el radar. Crea un objeto DibujarTexto con sus respectivos atributos.

String formatDist(**float** meters): Define un formato para mostrar la distancia del radio en forma de metros o kilómetros.

String formatDec(**float** val, **int** dec): Obtiene la distancia en kilómetros a partir de un valor en metros especificado.

En la figura 5.21, se muestra el Radar con todos sus complementos dibujados sobre la interfaz de navegación RA.

Figura 5.21 Radar en la interfaz de navegación RA.

- **Marcador** (Clase Marcador). Esta clase es la encargada de gestionar los marcadores en

pantalla. Cumple con funciones de cálculo de distancias para elegir que marcadores se pueden visualizar en pantalla así como dibujar la imagen y el texto adecuado al marcador. Contiene los siguientes atributos y métodos:

Atributos

DecimalFormat *FORMATO_DECIMAL* : Construye un formato decimal a partir de una expresión regular ("@#") para mostrar la distancia que se visualiza en el Radar.

Vector *simboloVector* : Es un vector usado para encontrar la ubicación de un marcador.
Vector *textoVector* : Vector usado para encontrar la ubicación del texto de un marcador.

Vector posicionVectorPantalla : Vector usado para la posición en pantalla.
Vector tmpSimboloVector : vector usado para posicionar el marcador en pantalla.
Vector tmpVector
Vector tmpTextoVector

float[1] distanciaArray : Distancia entre el punto de referencia y un marcador.
float[3] ubicacionArray : Ubicación en el espacio físico
float[3] posicionPantallaArray : Posicionamiento en la pantalla

float inicialY : Posición del eje inicial x de cada marcador. Por default: 0.0f

CameraModel *cam* : Instancia de la clase CameraModel para compatibilidad con la cámara.

DibujarTextoEnCaja cajaTexto : Instancia de la clase DibujarTetxtoEnCaja
DibujarPosicion contenedorTexto : Instancia de la clase Dibujar Posicion

float[3] simboloArray : Usados para dibujar el símbolo.
float[3] textoArray : Usados para dibujar el texto.

String name : Identificador único del nombre de un marcador.
UbicacionFisica ubicacionFisica : Ubicación física en el mundo del marcador.
double distancia : Almacena la distancia entre el dispositivo móvil y el marcador.
boolean isOnRadar : Bandera de visibilidad del marcador en el Radar.
boolean isInView : Bandera de visibilidad del marcador en la vista de pantalla.
Vector simboloXyzRelativoAVistaCamara : Usado para saber la ubicación del marcador en la vista de la cámara. (x, arriba y abajo, y, izquierda y derecha, z, no usado)
Vector textoXyzRelativoAVistaCamara : Usado para saber la ubicación del texto del marcador en la vista de la cámara. (x, arriba y abajo, y, izquierda y derecha, z, no usado)
Vector ubicacionXyzRelativoAUbicacionFisica Usado para saber la ubicación física relativa del marcador. (x, arriba y abajo, y, izquierda y derecha, z, no usado).
int color : Color del marcador.

int categoría : Identificador de la categoría del marcador.
String summary : Identificador del summary (Descripción del sitio turístico) del marcador.

String url_imagen : Identificador de la dirección de la imagen.

Constructores

```
public Marcador(String name, double latitude, double longitude, double altitude, int color, int categoria, String summary, String url_imagen) {
	set(name, latitude, longitude, altitude, color, categoria, summary, url_imagen);
}
```

El constructor asigna el nombre, latitud, longitud, altitud, color, categoría, descripción y url de la imagen con el método set.

Métodos

set(name, latitude, longitude, altitude, color, categoria, summary, url_imagen): Asignación de atributos.
String obtenerUrl() : Obtiene la url de la imagen.
obtenerCategoria() : Obtiene la categoría del marcador.
obtenerSummary() : Obtiene la descripción del marcador.
obtenerName() : Obtiene el nombre del marcador.
obtenerColor() : Obtiene el color del marcador.
obtenerDistance() : Obtiene la distancia entre dispositivo móvil y el marcador.
obtenerInitialY() : Obtiene el valor inicial de Y.
isOnRadar() : Obtiene el valor de la bandera para mostrar en Radar.
isInView() : Obtiene el valor de la bandera para mostrar en la vista de la cámara.
obtenerScreenPosition() : Obtiene la posición en pantalla del marcador.
obtenerLocation() : Obtiene la ubicación física relativa en el mundo del dispositivo móvil.
getHeight() : Devuelve el ancho del objeto contenedor del marcador.
getWidth() : Devuelve la altura del objeto contenedor del marcador.

actualizar(Canvas canvas, float addX, float addY): Este método usado para actualizar la vista de la cámara con el marcador adecuado y el Radar con la posición del marcador adecuado.

populateMatrices(CameraModel cam, float addX, float addY): Este método es usado para actualizar los vectores de posición del texto y del marcador de acuerdo a la matriz de rotación.

actualizarRadar() : Actualiza los parámetros del radar con el marcador adecuado. De acuerdo al radar, calcula las posiciones dentro del mismo para cada marcador que entre dentro del rango del círculo del radar.

actualizarVista() : Actualiza los parámetros de la vista de la cámara con el marcador adecuado. De acuerdo al rango del radar especificado, calcular las posiciones que se muestran en la vista de cámara y muestra el marcador cuando entra dentro del rango, de lo contrario, lo excluye.

calcPosicionRelativa(Location location) : Calcula una nueva posición relativa usando la ubicación recibida como parámetro. Actualiza la distancia y el radar.

actualizarDistancia(Location location) : Calcula una nueva distancia entre la

ubicación del marcador y la ubicación del dispositivo móvil.

handleClick(**float** x, **float** y) : Es un método asíncrono para verificar la pulsación realizada sobre un marcador. Devuelve las coordenadas en las que fue hecha la pulsación en la pantalla.

isMarcadorOnMarcador(Marcador marker, **boolean** reflect) : Contiene el código para determinar si el marcador recibido está sobrepuesto con un marcador adyacente. Se verifica si existe sobre posición en todas las esquinas del marcador, esto con la finalidad de evitar colisiones en los marcadores.

isPuntoOnMarcador(**float** x, **float** y, Marcador marker) : Este método es usado para verificar los puntos del marcador y que estos no colisionen con otros marcadores.

draw(Canvas canvas) : Este método determina si el marcador puede mostrarse. En caso de ser correcto, dibuja el ícono y el texto del marcador.

dibujarIcono(Canvas canvas) : Dibuja el ícono del marcador.
dibujarTexto(Canvas canvas) : Dibuja el texto del marcador.

compareTo(Marcador another) : Compara la longitud de nombre de dos marcadores.
equals(Object marker) : Compara dos marcadores.

En la figura 5.22 se muestra un marcador con todos sus componentes (caja, texto, ícono) sobre la interfaz de navegación RA.

Figura 5.22 Marcador en la interfaz de navegación RA.

- **Icono del Marcador** (Clase IconoMarcador). Esta clase dibuja un mapa de bits (bitmap) como un ícono para un marcador. Herede de la clase base Marcador.

  Atributos

  Bitmap bitmap : Es la imagen que se muestra como ícono.

  Métodos

  dibujarIcono(Canvas canvas) : Es una sobrecarga del método dibujarIcono de la clase Marcador. Se dibuja un ícono de 96 x 96 pixeles.

  En la figura 5.23 se muestra un ícono para un marcador determinado.

Figura 5.23 Ejemplo de un ícono para la categoría monumento.

- **Barra de zoom vertical** (Clase VerticalSeekBar). Es una extensión de la clase SeekBar. La principal diferencia con el objeto SeekBar es que en vez de ser horizontal, muestra una barra vertical personalizada. Este componente es usado para modificar el valor del radio en el Radar y los marcadores que se muestran en pantalla.

  Métodos

  `onSizeChanged(int w, int h, int oldw, int oldh) : Modifica el método onSizeChanged de la clase base SeekBar.`

  `onMeasure(int widthMeasureSpec, int heightMeasureSpec): Modifica el método onMeasure de la clase base SeekBar.`

  `onDraw(Canvas c) : Modifica el método onDraw de la clase base SeekBar. Rota 90 grados el objeto SeekBar para visualizar la barra de forma vertical.`

  `onTouchEvent(MotionEvent event) : Este método modifica el método de la clase base onTouchEvent para avanzar o reducir el progreso de la barra.`

  En la figura 5.24 se muestra un objeto Seekbar de forma vertical.

Figura 5.24 Barra vertical de zoom personalizada.

5.2.2.4 Capa de cámara

- **Objeto de vista de la cámara** (Clase SuperficieCamara). Esta clase maneja todo el código relacionado con la vista de la cámara. Hereda de la clase SurfaceView de Android.

  Atributos

SurfaceHolder *holder* : Inicializador de la cámara del dispositivo móvil.
Camera *camara* : Objeto cámara.

Métodos

surfaceCreated(SurfaceHolder holder) : Este método es usado para crear el objeto cámara y para liberar el objeto en caso de haber un problema (por ejemplo, que el dispositivo no cuente con una cámara).

surfaceDestroyed(SurfaceHolder holder): Este método es invocado cuando detenemos la ejecución de la aplicación para liberar los recursos de la cámara de forma adecuada.

surfaceChanged(SurfaceHolder holder, int format, int w, int h) : Este método es usado para calcular el tamaño de la vista previa de la cámara usando las clases de compatibilidad y de modelo para ajustar la vista a la medida correcta de la pantalla.

En la figura 5.25 se puede ver el acceso a la cámara del dispositivo móvil.

Figura 5.25 Acceso a la cámara del dispositivo móvil.

- **Compatibilidad** (Clase CameraCompatibility). Esta clase permite a la aplicación mantener la compatibilidad con versiones anteriores del Android y obtiene las limitaciones de versiones de APIs anteriores. (Clase adaptada del Framework Mixare Tools).

  Atributos

  Method *getSupportedPreviewSizes* : Almacena el tipo de pantalla del dispositivo para obtener la compatibilidad.
  Method *mDefaultDisplay_getRotation* : Almacena la rotación ejercida sobre el dispositivo.

  Métodos

  initCompatibility() : Verifica la compatibilidad del sistema operativo para que sea una versión superior de Android 2.0 y verifica que la pantalla tenga una resolución mínima de 480 x 320 px.

  getRotation(Activity activity) : Este método permite recuperar la rotación del dispositivo móvil.

  getSupportedPreviewSizes(Camera.Parameters params) : Este método devuelve una lista

de tamaños de vistas previas disponibles sobre el dispositivo.

- **Modelos de cámaras** (Clase CameraModel). Esta clase representa la cámara y su vista. (Clase adaptada del Framework MixareTools)

  Atributos

  ```
  int width : Almacena el ancho de la vista de la cámara.
  int height : Almacena la altura de la vista de la cámara.
  float distance : Almacena la distancia.
  ```

  Métodos

  ```
  set(int width, int height, boolean init)

  getWidth() : Retorna el ancho de la vista de la cámara.

  getHeight() : Retorna el alto de la vista de la cámara.

  setViewAngle(float viewAngle) : Actualiza la distancia con un nuevo ángulo de vista.
  ```

5.2.2.5 Capa de utilidades

- **Vector** (Clase Vector). Esta clase maneja todas las operaciones matemáticas detrás de los vectores. El código fue adaptado del Framework MixareTools.

  Atributos

  ```
  float[9] matrixArray : Es un arreglo usado en el método prod de esta clase.
  float x : Valor x del vector. Es inicializado en 0.
  float y : Valor y del vector. Es inicializado en 0.
  float z : Valor z del vector. Es inicializado en 0.
  ```

  Métodos

  ```
  getX() : Obtiene el valor de x.
  setX(float x) : Asigna el valor de x.
  getY(): Obtiene el valor de y.
  setY(float y) : Asigna el valor de y.
  getZ(): Obtiene el valor de z.
  setZ(float z) : Asigna el valor de z.

  get(float[] array) : Obtiene los valores de x, y y z.
   set(Vector v) : Asigna los valores a x, y y z desde un vector.
  set(float[] array) : Asigna los valores a x, y y z desde un arreglo.

  set(float x, float y, float z): Asigna los valores a x, y y z.

  equals(Object obj) : Compara el vector con un objeto dado para ver si estos son
  iguales.
  ```

add(float x, float y, float z) : Agregan los argumentos a un vector.

add(Vector v) : Agrega desde un vector hacia otro vector.

sub(Vector v) : Resta en el vector desde otro vector dado.

mult(float s) : Multiplica los valores del vector por el parámetro dado.

divide(float s) : Divide los valores del vector por el parámetro dado.

length() : Devuelve la longitud del vector.

norm() : Hace una división del vector a partir de su longitud.

cross(Vector u, Vector v) : Realiza una multiplicación de dos vectores.

prod(Matrix m) : Multiplica el vector con la matriz dada.

toString() : Devuelve el valor de x, y y z en una cadena.

- **Ángulo** (Clase Utilidades). Esta clase contiene un método para obtener el ángulo cuando calculamos la inclinación en la clase InclinacionAcimutCalc.

  Métodos

  obtenerAngulo(float center_x, float center_y, float post_x, float post_y) : Obtiene el ángulo de inclinación.

- **Inclinación y Acimut** (Clase InclinacionAcimutCalc). Esta clase es usada para obtener la inclinación y el ángulo acimut desde una matriz dada.

  Atributos

  float *acimut* : Almacena el ángulo acimut.
  float *inclinacion* : Almacena la inclinación

  Métodos

  obtenerAcimut() : Este método devuelve el ángulo Acimut.

  obtenerInclinacion() : Este método devuelve la inclinación.

  calcularInclinacionDespl(Matrix rotationM) : Este método obtiene la inclinación y el ángulo acimut desde una matriz dada.

- **Matriz** (Clase Matrix). Esta clase maneja todas las operaciones matemáticas al igual que la clase Vector (Esta clase fue adaptada del Framework Mixare Tools)

  Atributos

  float a1=0f, a2=0f, a3=0f : Almacena los valores de a1-a3.

`float b1=0f, b2=0f, b3=0f` : Almacena los valores de b1-b3.
`float c1=0f, c2=0f, c3=0f` : Almacena los valores de c1-c3.

Métodos

getA1() : Obtiene el valor de a1.

setA1(float a1) : Devuelve el valor de a1.

getA2(): Obtiene el valor de a2.

setA2(float a2) : Devuelve el valor de a2.

getA3(): Obtiene el valor de a3.

setA3(float a3) : Devuelve el valor de a3.

getB1(): Obtiene el valor de b1.

setB1(float b1) : Devuelve el valor de b1.

getB2(): Obtiene el valor de b2.

setB2(float b2) : Devuelve el valor de b2.

getB3(): Obtiene el valor de b3.

setB3(float b3) : Devuelve el valor de b3.

getC1(): Obtiene el valor de c1.

setC1(float c1) : Devuelve el valor de c1.

getC2(): Obtiene el valor de c2.

setC2(float c2) : Devuelve el valor de c2.

getC3(): Obtiene el valor de c3.

setC3(float c3) : Devuelve el valor de c3.

get(float[] array) : Obtiene los nueve elementos de la matriz.

set(Matrix m) : Asigna a los elementos de la matriz de acuerdo a una matriz dada.

set(float a1, float a2, float a3, float b1, float b2, float b3, float c1, float c2, float c3) : Asigna a los elementos de la matriz de acuerdo a los parámetros dados.

toIdentity() : Construye la matriz identidad.

adj() : Encuentra la matriz adjunta.

invert() : Encuentra la inversa de la matriz.

transpose() : Encuentra la matriz transpuesta.

det2x2(float a, float b, float c, float d) : Encuentra una matriz determinante de 2x2.

det() : Encuentra la matriz determinante.

mult(float c) : Multiplica los valores de la matriz por el valor dado en el parámetro.

prod(Matrix n) : Multiplica la matriz por otra matriz dada en los parámetros.toString() : Devuelve los valores de la matriz en una cadena.

5.2.2.6 Capa global

- **Almacenamiento de la información** (Clase DatosRA). Esta clase es la encargada del control global y el almacenamiento. Guarda todos los datos esenciales que se usan mientras la aplicación está corriendo.

  Atributos

  String *TAG* = "DatosRA" : Mensaje que se muestra en el LogCat.
  Map<String,Marcador> *listaMarcadores* : Lista completa de marcadores.
  List<Marcador> *cache* : Caché de marcadores que se almacenan.

  float[3] *arrayUbicacion* : Es un arreglo con los datos de ubicación.

  Location *ubicDefaultHardLoc* : Ubicación del dispositivo por defecto. Contiene la latitud, longitud y altitud almacenadas de la siguiente manera:

  ```
  ubicDefaultHardLoc.setLatitude(0);
  ubicDefaultHardLoc.setLongitude(0);
  ubicDefaultHardLoc.setAltitude(1);
  ```

  Object *radioLock* : Bloqueo en pantalla del componente Radar.
  float *radio* : Radio del radar.
  String *nivelZoom* : Nivel de zoom del componente barra.
  Object *progresoZoomLock* : Bloqueo en pantalla del componente Barra de zoom
  int *progresoZoom* : progreso de la barra de zoom.
  Location *ubicacionComun* : Almacena la ubicación en la que está el dispositivo. Por defecto se usa *ubicDefaultHardLoc*.
  Matrix *rotacionMatrix* : Almacena la matriz de rotación.
  Object *acimutLock* : Bloqueo del objeto Acimut.
  acimut : Valor del ángulo Acimut.
  Object *inclinacionLock* Bloqueo del objeto inclinación.
  float *inclinacion* Valor de inclinación.
  Object *rodarLock* : Bloqueo del objeto rodar.
  float *rodar* : valor de roll.

  Métodos

asignarNivelZoom(String nivelZoom) : Asigna el nivel de zoom.

asignarProgresoZoom(int progresoZoom) : Asigna el progreso de la barra de zoom.

asignarRadio(float radius) : Asigna el radio que se desea.

obtenerRadio() : Obtiene el valor del radio.

asignarUbicacionComun(Location ubicacionComun) : Asigna la ubicación actual del dispositivo móvil.

obtenerUbicacionComun() : Obtiene la ubicación actual del dispositivo.

asignarRotacionMatrix(Matrix rotacionMatrix) : Asigna la matriz de rotación.

obtenerRotacionMatrix() : Obtiene la matriz de rotación.

obtenerMarcadores() : Obtiene los marcadores.

asignarAcimut(float acimut) : Asigna el ángulo acimut.

obtenerAcimut() : Obtiene el ángulo acimut.

asignarInclinacion(float inclinacion) : Asigna la inclinación.

obtenerInclinacion() : Obtiene la inclinación.

asignarRodar(float rodar) : Asigna la variable roll.

obtenerRodar() : Obtiene la variable roll.

agregarMarcadores(Collection<Marcador> markers) : Agrega una colección de uno o más marcadores a la lista actual.

ubicacionCambiada(Location location) : Verifica el cambio de ubicación del dispositivo para actualizar los marcadores.

5.2.3 Implementación del bloque de Mapas

5.2.3.1 Capa de mapas

- **Conexión al API de Google Maps** (Clase MapaGoogle). Esta clase es la encargada de la conexión con los servidores de Google Maps y su API. Es una actividad heredada de la clase Fragment Activity. Muestra el mapa con la ubicación del dispositivo móvil además de los marcadores con los sitios turísticos en un radio cercano a la ubicación del dispositivo. Tiene la particularidad de trazar rutas desde el punto en que se encuentra el dispositivo y otro marcador seleccionado mediante el API de rutas Google Maps

  Atributos

  GoogleMap mapa : Objeto que hace referencia a la clase SupportMapFragment para mostrar el mapa en la pantalla.

int vista : Elemento para alternar entre cuatro diferentes vistas de mapa:

MAP_TYPE_NORMAL : Mapa normal.
MAP_TYPE_HYBRID : Mapa híbrido (con nombres de calles).
MAP_TYPE_SATELLITE : Mapa con fotografías satelitales.
MAP_TYPE_TERRAIN : Mapa con registro de elevaciones.

Métodos

onCreate(Bundle savedInstanceState) : Este método es lanzado cuando se inicia la actividad. En ella se cargan los marcadores enviados por el navegador RA, se determina la posición actual y se obtiene el mapa con la posición de cámara establecida.

comoLLegar(**double** originLat, **double** originLng, **double** destLat, **double** destLng): Este método es invocado cada vez que necesitamos trazar la ruta desde un punto de origen a otro punto de destino. Lanza una tarea asíncrona para cargar la ruta mientras se traza el mapa.

onCreateOptionsMenu(Menu menu): Este método es una sobrecarga de la clase base FragmentActivity y carga un menú para el botón físico de menú del dispositivo.

onOptionsItemSelected(MenuItem item) : Este método carga los ítems en el menú con las opciones de vista a elegir cuando se pulsa el botón físico de menú del dispositivo.

ActualizarMarcadores(**double** lat, **double** lng) : Este método actualiza los marcadores asignados a la vista del mapa a partir de su latitud y su longitud.

mostrarMarcador(**double** lat, **double** lng, String name) : Este método dibuja un marcador en el mapa de acuerdo a su latitud y su longitud. Muestra un ícono de acuerdo a su categoría y el nombre del marcador al pulsarlo.

RastreadorGPS Localizacion() : Obtiene la ubicación del dispositivo desde GPS asistido o desde el satélite GPS.

obtenerDireccionesURL(**double** originLat, **double** originLng, **double** destLat, **double** destLng) : Obtiene una cadena de solicitud para el API de Google Maps usada para solicitar la ruta desde un punto de origen hacia un destino. Por defecto, solicita una ruta a pie.

String downloadUrl(String strUrl) : Este método realiza la descarga de un objeto serializado en formato JSON desde con la dirección de rutas especificada desde los servidores de Google Maps.

En la figura 5.26 se puede observar el acceso al API de Google Maps.

Figura 5.26 Acceso a un mapa de Google Maps. En él se pueden ver distintos marcadores con sitios turísticos.

- **Descarga de rutas** (Clase DownloadTask). Esta clase es la encargada de lanzar una tarea asíncrona para descargar un objeto en formato JSON que contiene la ruta a pie desde un punto de origen hacia uno de destino. Dicha tarea se ejecuta en segundo plano y se traza una ruta en el mapa una vez que haya concluido la descarga.

  Métodos

  `doInBackground(String... url) : Método para iniciar la descarga en segundo plano.`

  `onPostExecute(String result) : Método que invoca la tarea que descarga el objeto JSON`

- **Tarea de obtención de datos** (Clase ParserTask). Esta clase obtiene la lista de lugares en formato JSON desde el servidor de Google.

  Métodos

  `doInBackground(String... jsonData) : Obtiene el objeto JSON con todos los lugares enviados desde el servidor de Google y lo asigna a una lista de lugares.`

  `onPostExecute(List<List<HashMap<String, String>>> result). Este método es ejecutado después de descargar la lista de lugares. Su función es trazar una ruta a partir de todos los lugares devueltos y mostrar la ruta en una línea de color rojo por las calles que dirijan al punto de destino.`

- **Rastreador GPS** (Clase RastreadorGPS). Esta clase se encarga de obtener la ubicación del dispositivo a través de internet con el GPS asistido o desde una triangulación con distintos satélites a través del chip GPS en caso de que lo incorpore el dispositivo. Contiene los siguientes atributos y métodos:

Atributos

boolean GPSHabilitado : Bandera para verificar si el chip GPS del dispositivo está habilitado.

boolean InternetHabilitado : Bandera para verificar si la conexión a internet del dispositivo está habilitada.

boolean bandObtenerUbicacion : Bandera para verificar si es posible obtener la ubicación del dispositivo móvil.

Location ubicacion : Ubicación actual del dispositivo móvil.
double latitud : Variable que almacena la latitud.
double longitud : Variable que almacena la longitud.

long *DISTANCIA_MIN_ACTUALIZAR* = 10 : Distancia mínima en metros para actualizar la ubicación.

long *TIEMPO_MIN_ACTUALIZAR* = 1000 * 60 * 1 : Tiempo mínimo entre actualizaciones de la ubicación.

LocationManager ubicacionManager : Manejador de ubicaciones.

Métodos

Location obtenerUbicacion() : Este método se encarga de obtener la ubicación del dispositivo poniendo como prioridad la ubicación desde el GPS, esto debido a la precisión de la ubicación. En caso de no contar con la ubicación desde los satélites GPS, se verifica la ubicación por internet con A-GPS.

obtenerLatitud() : Devuelve la latitud de la ubicación del dispositivo.

obtenerLongitud(): Devuelve la longitud de la ubicación del dispositivo.

bandObtenerUbicacion(): Este método verifica si es posible obtener la ubicación del dispositivo y devuelve un valor de verdadero o falso.

mostrarAlertaConfig(): En caso de contar con el chip GPS en el dispositivo, se muestra una alerta para poder configurarlo.

detenerGPS() : Detiene todos los servicios de forma segura.

- **Rutas en el mapa** (Clase DireccionesJSONParser). Esta clase se encarga de deserializar el objeto recibido en JSON desde el servidor de Google para obtener la lista de lugares.

Métodos

List<List<HashMap<String,String>>> parse(JSONObject jObject) : Recibe un objeto JSON y retorna una lista de listas que contienen la latitud y la longitud de cada lugar.

List<LatLng> decodePoly(String encoded) : Este método decodifica los puntos para formar las líneas que serán trazadas en el mapa.

Las rutas que se muestran, y que son descargadas desde el API de rutas de Google Maps, son trazadas con una línea roja, tomando en cuenta que están son rutas a pie. En la figura 5.27, se observa la ruta desde la posición actual del dispositivo móvil hasta el Monumento a Zaragoza.

Figura 5.27 Ruta desde la ubicación actual del dispositivo hasta un sitio turístico.

5.2.4 Implementación del bloque visualización RA

5.2.4.1 Capa de Vuforia

- **Registros del sistema** (Clase DebugLog). Esta clase es la encargada de mostrar los mensajes del sistema en el LogCat.

  Atributos

  ```
  String LOGTAG : Cadena para mostrar el identificador de la fuente del mensaje
  ```

  Métodos

  ```
  LOGE(String nMessage) : Mensaje de error.

  LOGW(String nMessage) : Mensaje de advertencia.

  LOGD(String nMessage) : Mensaje de depuración.

  LOGI(String nMessage) : Mensaje de entrada.
  ```

- **Procesamiento Vuforia** (Clase ImageTargets). Esta clase es la encargada del procesamiento de imágenes en tiempo real desde la cámara del dispositivo. Su función es encontrar el marcador en una escena y sobreponer un objeto renderizado en 3D para su visualización.

ATRIBUTOS

```
int FOCUS_MODE_NORMAL : Enfoque de la cámara normal
int FOCUS_MODE_CONTINUOUS_AUTO : Enfoque continuo de la cámara.

Constantes de estado de la aplicación
int APPSTATUS_UNINITED         = -1;
int APPSTATUS_INIT_APP         = 0;
int APPSTATUS_INIT_QCAR        = 1;
int APPSTATUS_INIT_TRACKER     = 2;
int APPSTATUS_INIT_APP_AR      = 3;
int APPSTATUS_LOAD_TRACKER     = 4;
int APPSTATUS_INITED           = 5;
int APPSTATUS_CAMERA_STOPPED   = 6;
int APPSTATUS_CAMERA_RUNNING   = 7;

Nombres de las librerías dinámicas pertenecientes las SDK de Vuforia que se cargan.
String NATIVE_LIB_SAMPLE = "ImageTargets"
String NATIVE_LIB_QCAR = "QCAR"

int HIDE_LOADING_DIALOG = 0 : Ocultar el cuadro de diálogo.
int SHOW_LOADING_DIALOG = 1 : Mostrar el cuadro de diálogo.

View mLoadingDialogContainer : Vista del cuadro de diálogo.

QCARSampleGLView mGlView : Vista OpenGL


private ImageTargetsRenderer mRenderer : Marcador con el renderizado.


private int mScreenWidth = 0 : Anchura de la pantalla del dispositivo
private int mScreenHeight = 0 : Altura de la pantalla del dispositivo.

private static final int INVALID_SCREEN_ROTATION = -1 : Constante que representa la
anulación de la orientación de la pantalla al desencadenar una consulta.

private int mLastScreenRotation = INVALID_SCREEN_ROTATION : Última detección de la
rotación de la pantalla.

private int mAppStatus = APPSTATUS_UNINITED : Estado de la aplicación

private InitQCARTask mInitQCARTask : Tarea asíncrona para inicializar el QCAR SDK
private LoadTrackerTask mLoadTrackerTask : Tarea asíncrona para inicializar QCAR SDK

Se usa un objeto para sincronizar de la inicialización del QCAR, el conjunto de
datos y el método de Android onDestroy() del ciclo de vida de eventos. Si la
aplicación es destruida mientras un dato es asignado o es cargado, entonces
esperamos la operación loading para finalizar antes de que se desconecte el QCAR.

Object mShutdownLock : Objeto de bloqueo para esperar la correcta finalización del
QCAR
```

int mQCARFlags : Inicialización de banderas QCAR.

Vector<Texture> mTextures : Carga de las texturas que se usan para la renderización.

GestureDetector mGestureDetector : Detección del gesto doble tap para cargar el menú de dialogo.

boolean mFlash : Menú contextual de opciones para activar o desactivar el flash de la cámara.

boolean mContAutofocus : Activar o desactivar el autoenfoque de la cámara

Menú para el intercambio de precisión en la parte alta o la parte baja de la catedral.
MenuItem mDataSetMenuItem
boolean mIsCatedralPresicion

RelativeLayout mUILayout : Capa que contendrá la vista de la aplicación

Métodos nativos

native int getOpenGlEsVersionNative() : Método nativo para consultar la versión de OpenGL.

native int initTracker() : Método nativo para inicializar un rastreador.
native void deinitTracker() : Método nativo para des inicializar un rastreador.

native int loadTrackerData() : Método nativo para cargar los datos del rastreador.
native void destroyTrackerData() Método nativo para terminar correctamente los datos del rastreador.

native void onQCARInitializedNative() : Método nativo para la inicialización del QCAR

native void startCamera() : Método nativo para iniciar la cámara del dispositivo.
native void stopCamera(): Método nativo para detener la cámara del dispositivo.

native void setProjectionMatrix() : Método nativo para ajustar o actualizar la proyección de la matriz para un contenedor que se renderiza para mostrar la realidad aumentada.

Métodos

Cargar librerías:

```
loadLibrary(NATIVE_LIB_QCAR);
loadLibrary(NATIVE_LIB_SAMPLE);
```

storeScreenDimensions() : Almacena las dimensiones de la pantalla del dispositivo móvil.

onCreate(Bundle savedInstanceState) : Este método es inicializado cuando se inicia la actividad o se regresa a la actividad con el botón BACK.

loadTextures() : Carga las texturas en formato JPEG o PNG, las cuales se usan en la etapa de renderizado.

getInitializationFlags(): Configura el QCAR con la versión optimizada de OpenGL para el dispositivo.

```
if (getOpenGlEsVersionNative() == 1)
      flags = QCAR.GL_11;
else
      flags = QCAR.GL_20;
```

onResume() : Este método se llama cuando comienza la interacción con el usuario. Muestra la vista OpenGL.

updateActivityOrientation() : Verifica la orientación del dispositivo. Existen tres posibles orientaciones:

ORIENTATION_PORTRAIT
ORIENTATION_LANDSCAPE
ORIENTATION_UNDEFINED

updateRenderView(): Actualiza la proyección de la matriz después de rotar el dispositivo.
onConfigurationChanged(Configuration config) : Método llamado al cambiar la configuración. Actualiza la orientación y almacena las dimensiones de la pantalla.

onPause() : Método llamado cuando se pausa la actividad. Se pausa la vista OpenGL, la cámara y se deshabilita el Flash en caso de que esté encendido.

onDestroy() : Método llamado cuando finaliza la aplicación. Se detienen todas las tareas asíncronas, se vacían los datos de memoria, el termina el rastreador así como el SDK QCAR se termina de forma correcta.

updateApplicationStatus(int appStatus) : Método sincronizado para actualizar el estado de la aplicación. Ejecuta una acción específica para cada estado como se ve a continuación:

APPSTATUS_INIT_APP : Inicializa los elementos gráficos de la aplicación (Aquellos que no intervienen con la visualización de realidad aumentada).
APPSTATUS_INIT_QCAR : Inicializa el SDK QCAR asíncronamente para evitar el bloqueo de la interfaz principal de la aplicación.
APPSTATUS_INIT_TRACKER : Inicializa el rastreador de marcadores.
APPSTATUS_INIT_APP_AR : Inicializa la realidad aumentada con los elementos específicos que se han inicializado con el SDK QCAR.
APPSTATUS_LOAD_TRACKER : Carga el conjunto de datos del rastreador.
APPSTATUS_INITED : Ejecuta el recolector de basura de Java.
APPSTATUS_CAMERA_STOPPED : Llama al método nativo para detener la cámara.
APPSTATUS_CAMERA_RUNNING : Llama al método nativo para iniciar la cámara.

initApplication() : Inicializa los elementos gráficos de la aplicación (Aquellos que no intervienen con la visualización de realidad aumentada).

initApplicationAR() : Inicializa los componentes de realidad aumentada.

getTextureCount() : Retorna el número de texturas registradas.

getTexture(int i) : Retorna la textura del objeto con el identificador especificado.

loadLibrary(String nLibName): Carga las librerías nativas almacenadas en libs/armeabi*

onKeyUp(int keyCode, KeyEvent event) : Muestra las opciones de la cámara cuando es presionado el botón físico Menú.

onTouchEvent(MotionEvent event) : Procesamiento de los gestos sobre la pantalla.

La creación de marcadores se realiza desde la página de Vuforia (Target Manager Vuforia). Para ello se necesitan fotografías del lugar en el que deseamos mostrar la realidad aumentada. Este sistema nos devuelve un archivo binario .dat y otro xml para referenciar el marcador cuando se ejecute la aplicación. En la figura 5.28 se muestra uno de los marcadores usados para la catedral de la ciudad de Puebla.

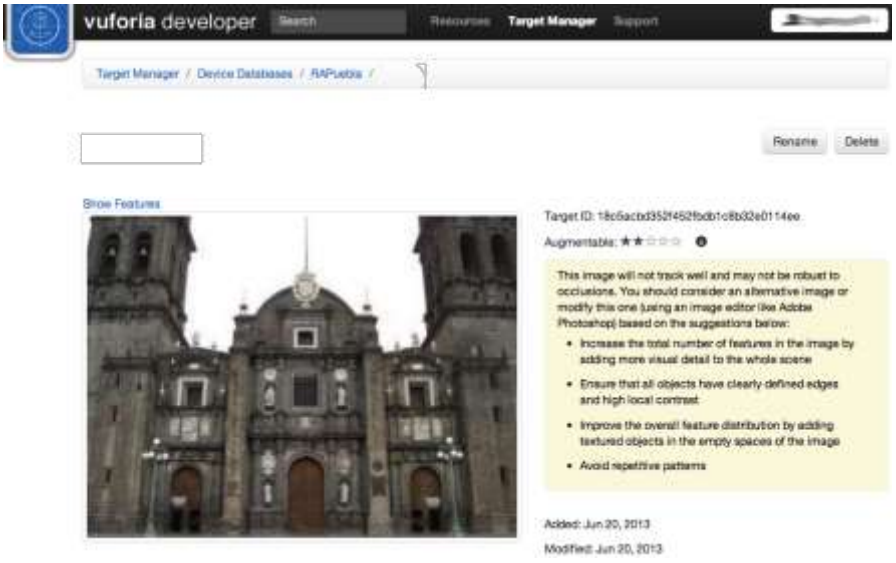

Figura 5.28 Marcador implementado para la catedral de Puebla. Está almacenado en un archivo binario .dat al que se accede desde el tracking generado en XML.

- **Cuadro de diálogo de espera** (Clase LoadingDialogHandler). Esta clase crea un manejador para actualizar el estado del Cuadro de diálogo desde un hilo UI con la finalidad de mostrar una barra de progreso mientras se cargan todos los componentes de realidad aumentada. Consta de los siguientes atributos y métodos:

  Atributos

`WeakReference<ImageTargets> mImageTargets` : Referencia a la clase ImageTargets.

Métodos

`LoadingDialogHandler(ImageTargets imageTargets)` : Carga la actividad ImageTargets una vez que se hayan creado todos los componentes.

`handleMessage(Message msg)` : Muestra una barra de progreso mientras se cargan todos los componentes.

En la figura 5.29 se observa una barra de progreso mientras se cargan todos los componentes de la aplicación de visualización de realidad aumentada.

Figura 5.29 Barra de progreso de la aplicación de visualización RA.

- **Tarea asíncrona : InitQCARTask**. Esta clase ejecuta una tarea asíncrona para inicializar el SDK QCAR. Esta tarea se carga cuando se muestra el cuadro de diálogo de espera.

  Métodos

  `doInBackground(Void... params)` : Muestra el progreso de la carga en segundo plano.

  `onPostExecute(Boolean result)` : Realiza la inicialización del QCAR actualizando el estado de la aplicación a *APPSTATUS_INIT_TRACKER*. En caso de que no se pueda cargar, envía un mensaje de error al LogCat y termina la aplicación de forma correcta.

- **Tarea asíncrona: LoadTrackerTask**. Esta clase ejecuta una tarea asíncrona para inicializar el rastreador de marcadores. Esta tarea es ejecutada cada vez que se busca un marcador con la cámara. Contiene los siguientes métodos:

  Métodos

  `doInBackground(Void... params)` : Carga los datos del rastreador.

  `onPostExecute(Boolean result)` : Inicializa la precisión de los marcadores para la catedral. Actualiza el estado de la aplicación a *APPSTATUS_INITED*. En caso de que no se pueda cargar un marcador, se envía un mensaje de error al LogCat y se termina la aplicación de forma correcta.

- **Lectura de gestos** (Clase GestureListener). Esta clase se encarga de procesar los eventos de gestos sobre la pantalla del dispositivo.

  Métodos

onSingleTapUp(MotionEvent e) : Este método registra un tap sobre la pantalla. La acción que realiza es activar o desactivar el enfoque de la cámara.

onDoubleTap(MotionEvent e) : Este método registra dos tap sobre la pantalla. La acción que realiza es mostrar un cuadro de diálogo con las opciones de la cámara.

- **Renderización de objetos 3D** (Clase ImageTargetsRenderer). Esta clase es la encargada de renderización de los objetos en OpenGL.

  Atributos

  boolean mIsActive : Bandera de activación.
  ImageTargets mActivity : Referencia a la clase ImageTargets.

  Métodos nativos

  native void initRendering() : Método nativo para inicializar el proceso de renderización.
  native void updateRendering(**int** width, **int** height) : Método nativo para actualizar la renderización.

  native void renderFrame() : Método nativo de la función de renderización.

  Métodos

  onSurfaceCreated(GL10 gl, EGLConfig config) : Este método es llamado para crear o recrear la superficie donde se renderiza el objeto 3D.

  onSurfaceChanged(GL10 gl, **int** width, **int** height) : Este método es llamado cuando la superficie de renderizado cambia de tamaño.

  onDrawFrame(GL10 gl) : Actualiza la matriz de renderización para dibujar el objeto 3D.

  En la figura 5.30 se observan las perspectivas el diseño de un objeto en 3D con el software Blender. Este objeto es usado para mostrarlo sobre el marcador al que se apunte con la cámara.

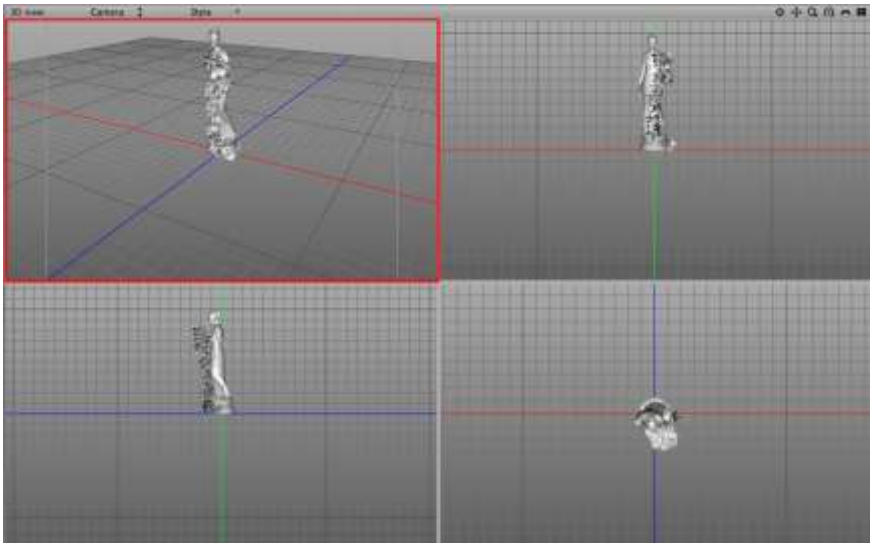

Figura 5.30 Objeto en 3D renderizado para la aplicación de visualización RA

5.2.5 Implementación del bloque de Servicio Web

5.2.5.1 Capa de servicio web Rest

- **Controlador de peticiones** (Clase PeticionesController). Esta clase es el controlador del servicio web. Es el encargado de contener las acciones que se podrán llamar según la URL y datos http que recibimos como petición de entrada al servicio web.

  Atributos

  `ManejadorPeticiones manejadorPeticiones : Es una instancia de la clase ManejadorPeticiones.`

  Métodos

  `JsonResult Sitios()`: Esta acción se limita a llamar al método ObtenerListaCompleta y serializar los datos como JSON.

```
[HttpGet]
 public JsonResult Sitios()
 {
     return Json(manejadorPeticiones.ObtenerListaCompleta(),
                 JsonRequestBehavior.AllowGet);
 }
```

  `JsonResult Sitio(string id, Sitio item)`: Este método es dependiente del método http que reciba (Las peticiones del modelo REST son “POST”, “PUT”, “GET”, “DELETE”). Para obtener la lista de lugares de acuerdo al radio más cercano, se usa una petición “GET”, la cual nos devolverá la lista de sitios que se encuentran en determinado radio.

```
public JsonResult Sitio(string id, Sitio item)
 {
     switch (Request.HttpMethod)
     {
         case "GET":
             return Json(manejadorPeticiones.ObtenerListaPersonalizada(id),
                         JsonRequestBehavior.AllowGet);
     }
       return Json(new { Error = true, Message = "Operación HTTP desconocida" });
 }
```

5.2.5.2 Capa de Modelos

- **Sitios** (Clase Sitios). Esta clase contiene los atributos de un marcador o sitio turístico. Esta clase es usada para serializarla y enviarla al dispositivo móvil. A continuación se muestran sus atributos:

  ```
  int Id : Almacena el identificador del marcador.
  int id_categoria : Almacena la categoría del marcador.
  string summary : Almacena una descripción del marcador.
  string tittle : Almacena el nombre del marcador.
  int elevation : Almacena la elevación del marcador.
  double lng : Almacena la longitud de la ubicación del marcador.
  double lat : Almacena la latitud de la ubicación del marcador.
  string url_imagen : Almacena una dirección de una imagen del marcador.
  ```

- **Manejador de peticiones** (Clase ManejadorPeticiones). Esta clase contiene las operaciones que se realizan sobre la base de datos BDSITIOSTUR y operaciones sobre los marcadores. A continuación se describen sus atributos y métodos.

  Atributos

  ```
  string cadenaConexion : Es la cadena de conexión hacia la base de datos BDSITIOSTUR.
  double RadioTierra : Es el radio aproximado en kilómetros de la tierra. Por defecto se
  asigna el valor 6371;
  ```

  Métodos

  ```
  double ObtenerDistancia(Geocoordenada punto1,Geocoordenada punto2) : Este método obtiene
  la distancia entre dos puntos.

  List<Sitio> ObtenerSitio(string id) : Obtiene la lista de sitios haciendo una consulta a
  la base de datos y siempre y cuando los sitios sean cercanos al radio enviado como
  parámetro. La consulta realizada a la base de datos es la siguiente:

     string sql = "SELECT id_sitio, id_categoria, summary, tittle, elevation, lng, lat,
                                 url_imagen FROM sitio";

  geo ObtenerListaPersonalizada(string id) : Obtiene un objeto geoCTI con una lista de
  marcadores específicos. Su función es llamar al método ObtenerSitio.

  List<Sitio> ObtenerSitios() : Obtiene la lista de todos los sitios existentes en la base
  de datos haciendo una consulta a la base de datos de la siguiente manera:
  ```

```
string sql = "SELECT id_sitio, id_categoria, summary, tittle, elevation, lng, lat,
                        url_imagen FROM sitio";
```

Para este método no importa el radio como parámetro.

geo ObtenerListaCompleta() : Devuelve un objeto geoCTI con la lista de todos los marcadores existentes. Su función es llamar al método ObtenerSitios.

- **Geocoordenada** (Clase GeoCoordenada). Esta clase almacena una latitud y una longitud como atributos.

  Atributos

  ```
  double Latitud : Almacena la latitud de una ubicación.
  double Longitud : Almacena la longitud de una ubicación.
  ```

- **geoCTI** (Clase geoCTI). Objeto que es devuelto al serializarlo en formato JSON.
  Atributos

  ```
  List<Sitio> geoCTI : Almacena una lista de sitios que serán serializados en formato
  JSON.
  ```

5.2.5.3 Capa de API

- **Área de registro del API** (Clase ApiAreaRegistration). La función de esta clase es dirigir las acciones recibidas hacia una u otra del controlador según la URL usada al realizarse la llamada al servicio web.

  Para este servicio web, solo se reconocerán dos tipos de URL, como se muestran a continuación:

  - http://148.228.xx.xx/ServicioWebRest/Api/Sitios : Obtiene la lista de todos los sitios que estén en la base de datos. La figura 5.31 muestra los datos que son devueltos en un navegador.

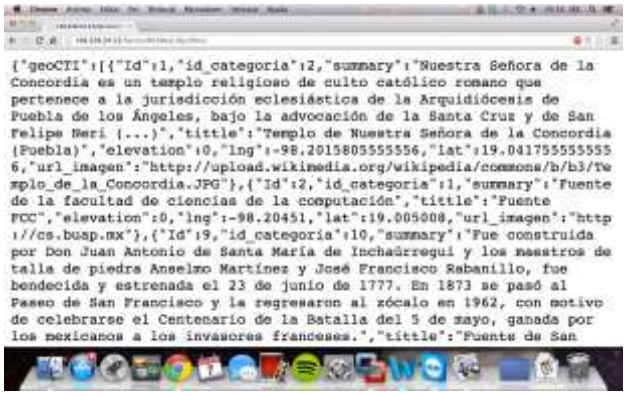

Figura 5.31 Ejemplo de objeto JSON consultado desde un navegador.

El patrón para acceder se registra por medio del método MapRoute dentro del método RegisterArea

```
public override void RegisterArea(AreaRegistrationContext context)
{
        …
        context.MapRoute(
        "AccesoSitios",
        "Api/Sitios",
        new
        {
                controller = "Peticiones",
                action = "Sitios"
        }
        );
…
```

- http://148.228.xx.xx/ServicioWebRest/Api/Sitios/Sitio/latitud,longitud,radio : Obtiene una lista específica de acuerdo con la ubicación (latitud y longitud) y del radio cercano de un sitio pasando como parámetro el radio en kilómetros al que deseamos obtener los marcadores. La figura 5.32 muestra los datos que son devueltos en un navegador.

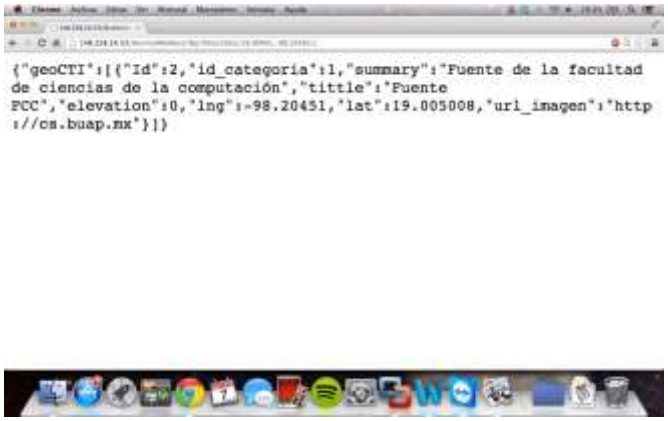

Figura 5.32 Ejemplo de un objeto JSON con los parámetros latitud, longitud y radio desde un navegador.

El patrón para acceder se registra por medio del método MapRoute dentro del método RegisterArea

```
public override void RegisterArea(AreaRegistrationContext context)
{
        …
        context.MapRoute(
        "AccesoSitio",
        "Api/Sitios/Sitio/{id}",
        new
        {
```

```
                    controller = "Peticiones",
                     action = "Sitio",
                     id = UrlParameter.Optional
                }
            );
            …
```

5.2.5.4 Objeto JSON geoCTI

La solicitud hacia el servidor web devuelve una lista de objetos en formato JSON. A continuación se explica cada uno de los atributos devueltos al realizar la petición. En este caso, se toma como referencia un solo sitio.

```
{
        "geoCTI":
        [
                {
                        "Id":2,
                        "id_categoria":1,
                        "summary":"Fuente de la facultad de ciencias de la
                        computación",
                        "tittle":"Fuente FCC",
                        "elevation":2029,
                        "lng":-98.20451,
                        "lat":19.005008,
                        "url_imagen":"http://cs.buap.mx"
                }
        ]
}
```

- geoCTI: Contiene la lista de marcadores.
- Id: Contiene el identificador del marcador
- id_categoria: Contiene el identificador de la categoría del marcador.
- summary: Contiene una descripción breve del marcador.
- tittle: Contiene el nombre del marcador
- elevation: Contiene la altitud del marcador
- lng: Contiene la longitud del marcador.
- lat: Contiene la latitud del marcador.
- url_imagen : Contiene una dirección con la imagen del marcador.

5.2.5.5 Implementación de la Base de datos BDSITIOSTUR

La base de datos fue implementada en SQL Server 2008. Contiene dos tablas:

- Sitios. Esta tabla almacena todos los sitios turísticos (marcadores) que serán serializados y enviados mediante en formato JSON.
- Categorías. Esta tabla contiene las categorías de los sitios turísticos.

- **Diseño de las tablas**
  La figura 5.33 y 5.34 muestran el diseño de las tablas en SQL Server 2008.

	Nombre de columna	Tipo de datos	Permitir valores NULL
▶🔑	id_sitio	int	☐
	id_categoria	int	☐
	summary	varchar(1000)	☐
	tittle	varchar(100)	☐
	elevation	int	☑
	lng	float	☐
	lat	float	☐
	url_imagen	varchar(300)	☑

Figura 5.33 Diseño de la tabla Sitios

	Nombre de columna	Tipo de datos	Permitir valores NULL
▶🔑	id_categoria	int	☐
	nombre	varchar(100)	☐

Figura 5.34 Diseño de la tabla Categorías.

5.2.6 Procedimiento para creación e importación de un modelo 3D.

Para la creación del objeto 3D se usó el software Blender. La intención es crear un objeto con extensión .obj que después será convertido a una cabecera .h. A continuación se muestra los pasos para importar un objeto 3D.

5.2.6.1 Creación del modelo en Blender

Se utiliza el editor de Blender para realizar el objeto que se mostrará en la visualización de realidad aumentada. En la figura 5.35, se muestra el objeto una vez renderizado en el editor de Blender.

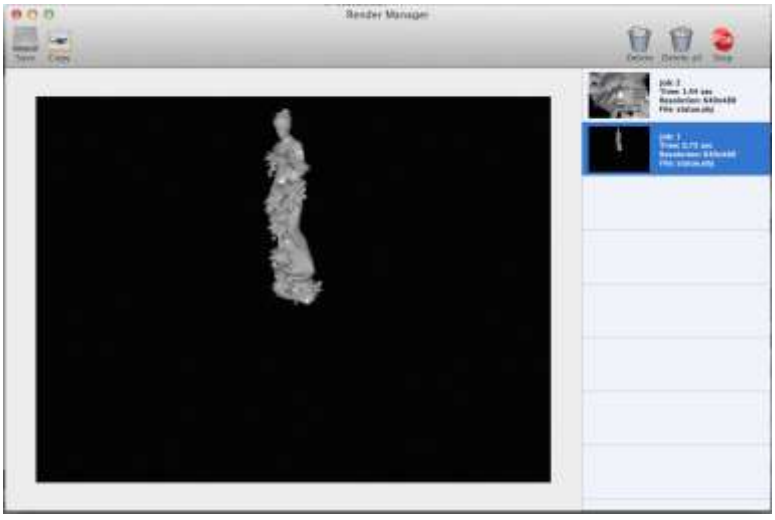

Figura 5.35 Render del objeto estatua.

Se exporta el archivo con la extensión .obj y se continúa con el siguiente paso.

5.2.6.2 Adaptación del modelo obj a .h

Para adaptar el archivo .obj a una cabecera .h se usa el programa OBJ2OPENGL. Es un script desarrollado en Perl. Para ejecutar esta herramienta se siguen los siguientes pasos (se usa un sistema UNIX):

- Ejecutar desde la terminal la siguiente línea: `./obj2opengl.pl estatua.obj`, tomando en cuenta que se encuentra dentro de la carpeta donde está el script y el archivo .obj
- El script genera una cabecera .h que contiene las siguientes líneas de código:

```
// include generated arrays #import "estatua.h"  // set input data to arrays
glVertexPointer(3, GL_FLOAT, 0, estatua Verts);
glNormalPointer(GL_FLOAT, 0, estatua Normals);
glTexCoordPointer(2, GL_FLOAT, 0, estatua TexCoords);  // draw data
glDrawArrays(GL_TRIANGLES, 0, estatuaNumVerts);
```

La figura 5.36 muestra el archivo .h y una imagen que será usada como textura del modelo realizado en 3D y su posterior renderizado.

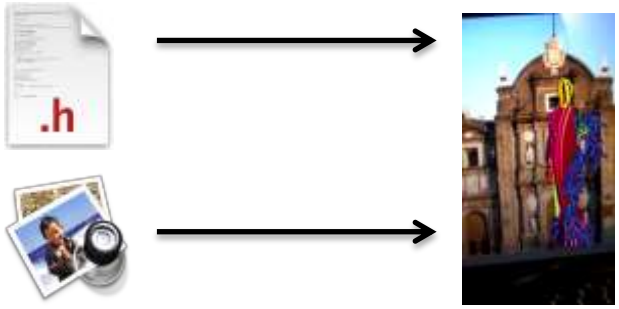

Figura 5.36 Archivos necesarios para renderizar un objeto en 3d von Vuforia.

Ahora debemos incluir esta cabecera con sus variables dentro del archivo ImageTargets.cpp.

5.2.7 Procedimiento para compilar un proyecto de Vuforia con NDK

Antes de crear un proyecto para Android desde Eclipse, debemos compilar los archivos .cpp con el NDK de Java. Para esto se siguen los siguientes pasos:

- Dentro de la carpeta con el proyecto de Vuforia, ejecutamos en el Terminal el siguiente comando: `./ndk-build.`
- La salida de la compilación debe lucir de la siguiente manera:

```
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi-v7a/gdbserver
Gdbsetup       : libs/armeabi-v7a/gdb.setup
Compile++ arm  : ImageTargets <= ImageTargets.cpp
```

```
Compile++ arm  : ImageTargets <= SampleUtils.cpp
Compile++ arm  : ImageTargets <= Texture.cpp
StaticLibrary  : libstdc++.a
Prebuilt       : libQCAR.so <= jni/../../../build/lib/armeabi/
SharedLibrary  : libImageTargets.so
Install        : libImageTargets.so => libs/armeabi/libImageTargets.so
Install        : libQCAR.so => libs/armeabi/libQCAR.so
Compile++ arm  : ImageTargets <= ImageTargets.cpp
Compile++ arm  : ImageTargets <= SampleUtils.cpp
Compile++ arm  : ImageTargets <= Texture.cpp
StaticLibrary  : libstdc++.a
Prebuilt       : libQCAR.so <= jni/../../../build/lib/armeabi-v7a/
SharedLibrary  : libImageTargets.so
Install        : libImageTargets.so => libs/armeabi-v7a/libImageTargets.so
Install        : libQCAR.so => libs/armeabi-v7a/libQCAR.so
```

6. Instalación y pruebas del sistema

6.1 Instalación de la aplicación

La aplicación puede ser instalada al generar un archivo .apk exportándolo desde el IDE Eclipse. Dicho archivo puede ser exportado al dar clic derecho sobre el proyecto /Exportar/Exportar Aplicación Android (Figura 6.1).

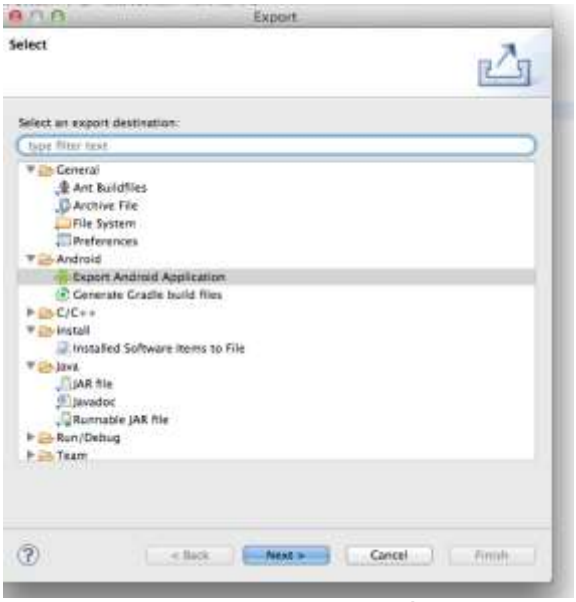

Figura 6.1 Exportar Aplicación Android

La segunda opción es copiar al dispositivo el archivo *proyecto.apk* que se encuentra en la carpeta proyecto/bin (Figura 6.2).

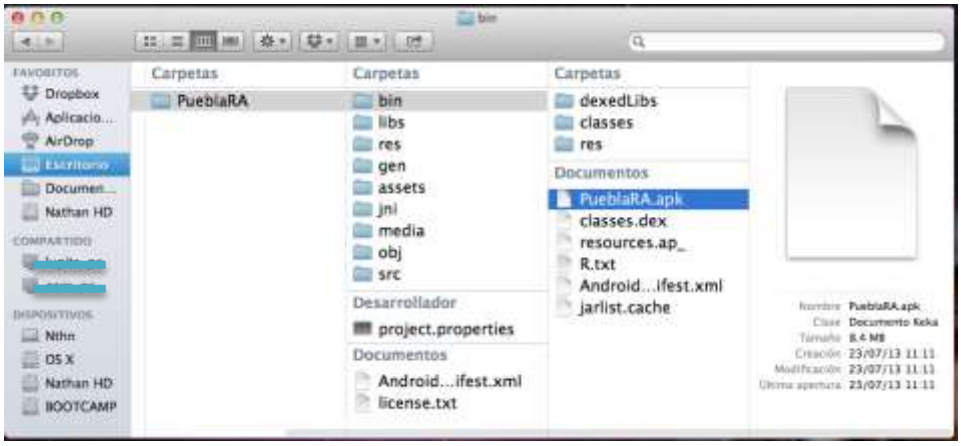

Figura 6.2 Archivo .apk que se copia al dispositivo con Android.

Se espera que la aplicación sea descargada desde la tienda de aplicaciones online de Google. Para

ello, se accederá desde el propio dispositivo conectado a Internet para poder instalarla, solo es necesario la autorización con una cuenta de Google.

6.2 Iniciar la aplicación

Al arrancar la aplicación en un dispositivo con Android, se muestra la interfaz de inicio con una actividad por dos segundos. Destacan el título con el nombre la aplicación, el ícono de la aplicación así como el logotipo del CTI y de Vuforia (Figura 6.3).

Figura 6.3 Arranque de la aplicación

6.3 Menú principal

En esta interfaz se muestra el menú para acceder a los modos de realidad aumentad. El primero pertenece al navegador de realidad aumentada. El segundo pertenece al visualizador de realidad aumentada para la catedral de Puebla. La tercera opción accede a una vista de información de la aplicación. En la figura 6.4 Se observa el menú principal en modo retrato y en le figura 6.5 se observa el menú en modo apaisado.

Figura 6.4 Modo retrato del menú.

Figura 6.5 Modo apaisado del menú

6.4 Navegador

El navegador de realidad aumentada se muestra accediendo desde el menú en la opción Navegador. Esto mostrará una actividad como se observa en la figura 6.6.

Figura 6.6 Vista principal del navegador de realidad aumentada.

6.4.1 Radar

Muestra todos los marcadores que se encuentren dentro del límite elegido con la barra de zoom. Cada marcador o sitio turístico es mostrado como un punto de color blanco y cambian de acuerdo a la orientación del dispositivo. En la parte superior del Radar se observa la orientación actual del dispositivo, mientras que en la parte inferior se muestra el radio en metros o kilómetros a los que se desea ver marcadores (Figura 6.7).

Figura 6.7 Vista del Radar en el navegador con orientación hacia el sur y un radio de 1 km.

6.4.2 Barra de Zoom

Se encarga de incrementar o disminuir el radio al que se desean ver los marcadores. El radio máximo de visualización es de 10 kilómetros. Al incrementar o disminuir el valor de la barra de zoom, el valor del Radio también cambia. La figura 6.8 muestra la barra de zoom de la aplicación.

Figura 6.8 Barra de zoom para cambiar el radio.

6.4.3 Marcadores

Los marcadores se muestran sobre la vista de la pantalla de acuerdo a su enfoque en determinada orientación. Todos los marcadores se muestran al enfocar hacia un lugar con el dispositivo móvil en posición horizontal. Estos son mostrados con el nombre del marcador y un ícono de acuerdo a su categoría. En la figura 6.9 se observan 2 marcadores al enfocar al sureste.

Figura 6.9 Marcadores o sitios enfocados al sureste.

Al pulsar sobre un marcador, nos muestra un cuadro de diálogo en el cual vemos el nombre del marcador, una fotografía de referencia del sitio y la descripción del sitio turístico. También tiene la opción “Como llegar” para visualizar la ruta más cercana a pie para llegar al sitio (Figura 6.10).

Figura 6.10 Al pulsar un marcador, se muestra información del mismo con la opción de trazar una ruta para llegar a él.

6.4.4 Mapa

Cuando giramos nuestro dispositivo de forma que quede en paralelo al suelo, se muestra el mapa con nuestra ubicación y todos los marcadores que se encuentren dentro del radio deseado. Cada marcador es representado con el ícono de su categoría (Figura 6.11).

Figura 6.11 Mapa con marcadores.

Al elegir la opción “Como llegar” sobre un marcador, se muestra la ruta más corta en el mapa para llegar a dicho sitio (Figura 6.12).

Figura 6.12 Ruta a pie para llegar desde la ubicación actual hacia el marcador elegido.

Podemos elegir entre cuatro distintas vistas del mapa al presionar el botón de menú. Estas vistas se pueden observar en la figura 6.13.

Figura 6.13 Vistas normal (arriba izquierda), hibrido (arriba derecha), satélite (abajo izquierda) y elevación (abajo derecha) del mapa.

6.5 Catedral

Al elegir la opción Catedral desde el menú principal, nos muestra una vista con las indicaciones para visualizar la realidad aumentada en la catedral de la ciudad de Puebla. Para ello, se presiona el botón Iniciar (Figura 6.14).

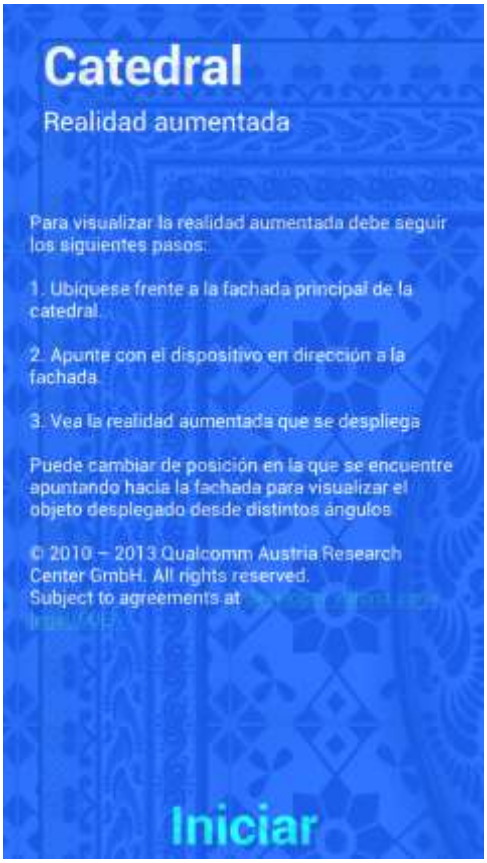

Figura 6.14 Instrucciones para visualizar la realidad aumentada en la catedral de Puebla.

6.5.1 Visualización de la realidad aumentada

Al iniciar esta actividad, se muestra una vista de la cámara del dispositivo. La principal diferencia entre la vista de esta cámara con la vista del navegador RA es la precisión para auto enfocar auxiliado del hardware del dispositivo (en el caso en que la cámara del dispositivo cuente con autoenfoque). Para mostrar la realidad aumentada es necesario seguir las siguientes indicaciones:

1. Ubíquese frente a la fachada principal de la catedral.
2. Apunte con el dispositivo con dirección a la fachada.
3. Visualizar la realidad aumentada que se despliega.

En la figura 6.15 se observa el objeto estatua mostrado al enfocar la catedral de la ciudad de Puebla. En la figura 6.16 se observa el mismo objeto en modo apaisado del dispositivo.

Figura 6.15 Objeto estatua superpuesto sobre la fachada de la catedral.

Figura 6.16 Objeto estatua superpuesto sobre la fachada de la catedral en modo apaisado.

6.6 Pruebas en dispositivos Android

6.6.1 Requerimientos del sistema

Para que el sistema funcione de forma fluida, se requieren de las siguientes características en el dispositivo móvil

- Sistema operativo Android en su versión 4.0 o superior.
- 512 Mb en memoria RAM
- 16 Mb de espacio en almacenamiento interno.
- Conectividad a Internet (Requiere un plan de datos).
- Chip interno GPS (opcional)
- Cámara VGA trasera

6.6.2 Pruebas realizadas

La aplicación móvil fue instalada sobre la siguiente lista de dispositivos Android:

- Samsung Galaxy S3
- Motorola RZR

En la figura 6.17 se muestra la aplicación corriendo sobre el Motorola RZR.

Figura 6.17 Menú principal de la aplicación corriendo sobre el dispositivo Motorola

En la figura 6.18 se observa la prueba realizada con el navegador de realidad aumentada sobre el mismo dispositivo.

Figura 6.18 Navegador de realidad aumentada corriendo sobre el dispositivo Motorola RZR.

En la figura 6.19 se muestra una captura de pantalla del marcador ubicado cerca de la capilla del Rosario en el centro histórico de la ciudad de Puebla. Se observa el nombre del sitio en cuestión así como la distancia en metros.

Figura 6.19 Capilla del Rosario vista con el navegador de realidad aumentada con vista al noroeste.

Conclusiones

Este trabajo presenta el desarrollo de un sistema para dispositivos móviles cuya principal finalidad es mostrar sitios turísticos de la ciudad de Puebla con realidad aumentada y geolocalización, haciendo énfasis en exponer la realidad aumentada sin la necesidad de usar marcadores artificiales. La metodología usada para el desarrollo del sistema fue Scrum, en la cual se usaron Sprints para la entrega de incrementos del sistema totalmente operativos. Gracias al uso de esta metodología, se fue desarrollando cada incremento en un tiempo no superior a un mes, con lo cual el sistema pudo ser completado en tiempo y forma de acuerdo al diagrama de actividades.

Uno de los aspectos a tomar en cuenta a la hora de desarrollar aplicaciones móviles son los recursos de hardware con los que cuenta el dispositivo. Para este trabajo, se emplean tareas de reconocimiento de imágenes en tiempo real, renderización de objetos en tercera dimensión, así como posicionamiento en pantalla de objetos en 2D, tareas que ocupan un gran porcentaje de la memoria y el poder de procesamiento del procesador de un dispositivo móvil. Ya que el consumo de recursos de memoria es muy alto para este tipo de tareas, la aplicación desarrollada solo está disponible para dispositivos que cuenten con ciertas características de hardware (Ver 6.6.1). El sistema cuenta con un límite de veinte marcadores que se muestran en tiempo real, esto debido a la carga de objetos que se muestran puede resultar muy grande en cuanto al consumo de memoria.

Las tecnologías usadas en este proyecto son tanto de uso libre como privativas. Es necesario mencionar que se usaron marcas registradas como es el caso de Vuforia, que exigen que un logotipo de dicha marca sea mostrada cuando se usa su tecnología para desarrollar y mostrar realidad aumentada. Otro caso es el uso de la API de Google Maps, la cual tiene un límite de 20,000 consultas diarias desde la aplicación y además se exige que los datos que sean consultados a sus servidores (como el uso de las rutas) sean expresamente utilizados para visualizarlos en un mapa.

Otro aspecto a tomar en cuenta es que la aplicación desarrollada necesita de una constante conexión a internet para descargar la información desde un servidor. Cuando se eligió la forma de transmisión de datos, se determinó usar el formato JSON, el cual es un formato muy popular de transmisión así como uno de los que menos tráfico genera a la hora de realizar las consultas a un servidor. Aunque el sistema está preparado para funcionar con redes wifi, se debe contemplar el uso de datos por parte de un proveedor de internet para móviles que garantice una conexión a internet en todo momento. Otro aspecto a tomar en cuenta es el uso de un chip GPS instalado en el dispositivo, puesto que la aplicación está preparada para funcionar sin la necesidad del mismo, el uso de este servicio mejorará la precisión de la ubicación del dispositivo y los sitios turísticos a su alrededor.

El servicio web se desarrolló con la finalidad de que cualquier persona pueda consultar los lugares que se encuentran a su alrededor, pero con la limitación de obtener solo los 20 lugares más cercanos de acuerdo a su posición o la posición que ellos especifiquen, pero existe la opción de consultar todos los lugares que se encuentren en la base de datos, con la limitantes de que cada persona debe ocupar y manipular los datos de acuerdo a su conveniencia. Dicho servicio web fue

probado en el laboratorio del Centro de Tecnologías de la Información de la Facultad de Ciencias de la Computación de la BUAP.

Gracias a este trabajo personalmente pude conocer gran parte de los sitios turísticos que existen de la ciudad de Puebla, ciudad considerada patrimonio de la humanidad por la Unesco, incentivando en mi la conciencia por la preservación de la herencia que generaciones anteriores dejaron en el legado de la historia de la ciudad, además de aplicar las tecnologías actuales en dispositivos móviles con la finalidad de acercar a más personas a estas, y que los ayuden a ampliar sus conocimientos sobre esta ciudad para ayudar a preservar sus lugares históricos.

Bibliografía

[1] Raghav Sood. Pro Android Augmented Reality. Appres 2012.
Chapter 1, Applications of Augmented Reality. Pages 1-12.
Chapter 2, Basics of Augmented Reality on the Android Platform. Pages 13-31.
Consultado el 25 de Junio de 2013.

[2] Lee, T., Höllerer, T. (2007) Handy AR: Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking. IEEE International Symposium on Wearable Computers (ISWC '07).
Consultado el 16 de Abril de 2013.

[3] Kato, H., Billinghurst, M.(1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system, Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR 99).
Consultado el 16 de Abril de 2013.

[4] Ana Serrano Mamolar . Herramientas de desarrollo libres para aplicaciones de Realidad Aumentada con Android. Análisis comparativo entre ellas. Universidad Politécnica de Valencia. Trabajo fin de Máster. Septiembre de 2012.
Consultado el 16 de Abril de 2013.

[5] Desarrollo de aplicación en Android con acceso a Web Service. Introducción al desarrollo de aplicaciones móviles en Android. Alfonso Felipe Lima Cortés. Tercer congreso regional en TIC. Noviembre de 2012.
Consultado el 16 de Abril de 2013.

[6] "More deeply, the framework exists to separate the representation of information from user interaction." The DCI Architecture: A New Vision of Object-Oriented Programming - Trygve Reenskaug and James Coplien - Marzo 20, 2009. Article.
www.artima.com/articles/dci_vision.html
Consultado el 21 de Abril de 2013.

[7] Geolocalización con HTML5 (API +JavaScript). Jorge Ayuso Molina.
http://jorgeayuso.com/geolocalizacion-con-html5-api/
Consultado el 11 de Abril de 2013.

[8] Definición de Geolocalización. Definición ABC.
http://www.definicionabc.com/geografia/geolocalizacion.php
Consultado el 12 de Abril de 2013.

[9] Milgram, Paul; H. Takemura, A Utsumi F. Kishino(1994).”Augmented Reality, A class of displays on the reality- virtuality continuum”. Pages 2351.
Consultado el 16 de Abril de 2013.

[10] About Nonuniform Rational B-Splines - NURBS
http://web.cs.wpi.edu/~matt/courses/cs563/talks/nurbs.html
Consultado el 16 de Abril de 2013.

[11] Gráficos 3D por computadora.
http://commons.wikimedia.org/wiki/3D_computer_graphics
Consultado el 16 de Abril de 2013.

[12] Servicio Web.
http://es.wikipedia.org/wiki/Servicio_web
Consultado el 16 de Abril de 2013.

[13] 7 de cada 10 mexicanos tendrán un Smartphone en 2015. Excélsior 2012.
http://www.excelsior.com.mx/2012/01/26/dinero/805541
Consultado el 11 de Diciembre de 2013.

[14] Anexo. Frameworks para servicios web.
http://es.wikipedia.org/wiki/Lista_de_frameworks_para_servicios_web
Consultado el 16 de Abril de 2013.

[15] DroidShooting. Quest-Com Co.,Ltd - 25 de febrero de 2012. Juegos de acción y arcade
https://play.google.com/store/apps/details?id=jp.co.questcom.droidshooting
Consultado el 16 de Abril de 2013.

[16] Simple Example of MVC (Model View Controller) Design Pattern for Abstraction.
http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design
Consultado el 21 de Abril de 2013.

[17] Programación Gráfica. Gráficos 3D.
http://www.oocities.org/valcoey/intro3d.html
Consultado el 17 de Abril de 2013.

[18] Introducción a ASP.NET MVC. Desarrollo web punto com
http://www.desarrolloweb.com/articulos/introduccion-asp-net-mvc-dotnet.html
Consultado el 5 de Agosto de 2013.

[19] Vuforia developer. Resources. Vuforia SDK Architecture.
https://developer.vuforia.com/resources/dev-guide/vuforia-ar-architecture
Consultado el 5 de Agosto de 2013.

[20] ABI Research: Cumulative mobile app revenues to exceed $30 billion by the end of the year.
http://www.intomobile.com/2012/11/26/abi-research-cumulative-mobile-app-revenues-exceed-30-billion-end-year/
Consultado el 11 de Diciembre de 2013.

[21] Promueven “rutas de Querétaro” a través de aplicación tecnológica. Diario Rotativo.
http://www.rotativo.com.mx/noticias/metropoli/queretaro/35436-promueven-rutas-de-queretaro-a-traves-de-aplicacion-tecnologica/
Consultado el 1 de Agosto de 2013.

[22] El fracaso de Apple Maps. MKT Capacitación.
http://www.marketingcapacitacion.com/index.php/noticias-marketing-digital/51-noticias-app-marketing/188-el-fracaso-de-apple-maps
Consultado el 1 de Agosto de 2013.

[23] 1/8/2013 Puebla desde el aire. Benemérita Universidad Autónoma de Puebla.
http://www.buap.mx/vision/patrim/Puebla/
Consultado el 1 de Agosto de 2013.