UNIVERSIDAD POLITÉCNICA DE VALENCIA

FACULTAD DE INFORMÁTICA

INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

HÉCATE

Aplicación Android con Realidad Aumentada
y Geolocalización

ALUMNO:

Jose Mª Martinez Lechon

DIRECTOR:

Juan Miguel García Gómez

TUTORES:

Zoe Valero Ramón
José Luis Bayo Montón

Año Académico 2011/2012

Universidad Politécnica de Valencia

Camino de Vera s/n

46022 Valencia

España

Índice general

Introducción

Actividades como desplazarse, comunicarse, usar o manipular forman parte de la actividad humana, dependiendo de las capacidades técnicas, cognitivas o físicas cada persona, se afrontan estas actividades con un grado diferente de culminación. Garantizar la accesibilidad significa asegurar que cualquier persona puede realizar estas actividades de forma segura, cómoda e independiente sin encontrar ningún tipo de barrera. Sin embargo, distintos factores tanto sociales como económicos o arquitectónicos limitan la realización de estas actividades a un colectivo importante de la población como son las personas con movilidad reducida o restricción en el desplazamiento. Según el Instituto Nacional de Estadística, en la sociedad española este colectivo ronda los cuatro millones de personas, cerca de un 9% de la población. Como consecuencia de las deficiencias en las infraestructuras públicas estas personas ven limitado su acceso al medio, lugar de trabajo, ocio o transporte. El conocimiento del estado y nivel de accesibilidad del entorno que les rodea puede suponer una gran ventaja para este colectivo, de forma que pueden planificar sus desplazamientos en base a la información de accesibilidad disponible.

El proyecto Hécate, dentro del que se enmarca el presente trabajo, tiene como objetivo la creación de un sistema integral de gestión de accesibilidad global y de consulta e información ciudadana que permita mejorar de forma continua la accesibilidad en los espacios públicos. La solución propuesta consiste en un sistema formado por dos aplicaciones independientes retroalimentadas.

La primera aplicación se trata de una herramienta de Gestión Integral de Accesibilidad (GIA). Es una herramienta profesional de accesibilidad con la que el profesional de accesibilidad valida, planifica y gestiona la accesibilidad del espacio público basándose en información urbanística y arquitectónica heterogénea generada por distintos actores: ciudadanía (perteneciente a algún colectivo o no), la entidad local y el propio profesional. Con esta herramienta el profesional podrá diagnosticar la accesibilidad del espacio, planificar las actuaciones necesarias, realizar la propuesta de solución a las carencias detectadas, certificar la accesibilidad del espacio según la

legislación y normativa actualizada (UNE e ISO), realizar presupuestos y gestionar la información generada.

La segunda es una aplicación de Consulta, Información y Rutas (VADEO 2.0) para el ciudadano basada en una aplicación de red social online y un servidor de mapas. A través de esta herramienta, el usuario puede consultar la información de accesibilidad generada por los distintos actores, trazar rutas accesibles entre distintos puntos, geo-referenciar dichos puntos y generar información sobre el nivel de accesibilidad de su entorno, la cual podrá ser analizada y validada posteriormente por el profesional. VADEO 2.0 está accesible en versión web y en versión móvil. La versión móvil ha sido desarrollada para dispositivos Android, albergando una aplicación de Realidad Aumentada añadida como módulo independiente que aporta una funcionalidad adicional. Esta funcionalidad adicional muestra el nivel de accesibilidad en tiempo real sobre el espacio público alrededor del usuario, y ha sido desarrollada como una aplicación independiente que se ajusta a las necesidades informativas de Vadeo, pero que además, dado que es una aplicación, pueda ser reaprovechada para incorporarse en otras aplicaciones o como aplicación independiente. La aplicación de Realidad Aumentada supone otra manera de consultar la información de accesibilidad generada por los distintos actores, superponiendo imágenes del mundo real con información de accesibilidad, facilitando la referencia de la accesibilidad de la zona rápidamente. Toda esa funcionalidad de es la desarrollada en el presente trabajo.

El proyecto Hécate se desarrolla en el departamento de Tecnologías para la Salud y el Bienestar (TSB), del Instituto de aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas (ITACA).

1. Objetivos

El proyecto tiene como objetivo desarrollar una aplicación de Realidad Aumentada (a partir de ahora R.A.) que se incorpore como complemento en la aplicación Vadeo 2.0 Móvil para Android realizada en el proyecto Hécate. La aplicación debe ofrecer dinámicamente información en tiempo real acerca del estado de la accesibilidad de las vías públicas.

Los principales objetivos que debe cumplir la aplicación son los siguientes:

- Utilizar tecnologías de localización para ubicar la posición del usuario en exteriores.
- Representar en la pantalla del dispositivo la información real captada por la cámara del dispositivo junto con la información descargada de los servidores de Hécate (Vadeo).
- Optimizar los recursos que consuma el dispositivo, tanto de tráfico de datos como de batería.
- Utilizar el modelo de datos heredado del proyecto Vadeo.
- Cumplir con los estándares básicos de accesibilidad establecidos por Android para facilitar la usabilidad de la aplicación.

Los siguientes objetivos son secundarios pero deben ser incorporados en la medida que sea posible:

- La comunicación con el servidor deberá ser escalable para que futuras mejoras puedan ser incorporadas al proyecto.
- La información de RA debe integrarse completamente dentro de la aplicación de Vadeo sin necesidad de lanzar otras aplicaciones.
- Emplear tecnologías o herramientas que permitan la futura explotación de la aplicación, sin ningún tipo de cláusula restrictiva.
- Incorporar herramientas de accesibilidad para poder facilitar la utilización por parte de personas con limitaciones físicas.

2. Estructura de este documento

El documento realiza una descripción progresiva sobre las nociones necesarias para comprender el trabajo realizado en el proyecto. El primer capítulo realiza un estudio para determinar a quien va dirigida esta aplicación, las personas con movilidad reducida, y que discapacidades pueden encontrase en ese sector de la sociedad, además describe el concepto de Realidad Aumentada en el que se basa el proyecto. El segundo capítulo resume el estado del arte de aplicaciones de Realidad Aumentada para determinar si existe alguna que se asemeje a los objetivos marcados en el proyecto. El tercer capítulo detalla todas las herramientas utilizadas en el proyecto y se concreta que Framework de Realidad Aumentada se ha utilizado. Con las herramientas de trabajo claras, el capítulo cuatro especifica las características, requisitos y funcionalidades que la aplicación de Realidad Aumentada debe cumplir. El quinto capítulo contiene una introducción a todo el contenido del proyecto Hécate y define todos y cada uno de los componentes de la aplicación de Realidad Aumentada para determinar el rol que desempeñan. Las conclusiones del proyecto están definidas en el capítulo seis. Para finalizar los siguientes capítulos contienen un glosario de términos para facilitar la comprensión de los términos o tecnicismos, la bibliografía utilizada y dos apéndices con información complementaria de apartados del documento.

1 Preliminares

La aplicación Hécate está dirigida a personas con movilidad reducida. Estas personas tienen limitadas, tanto temporal como permanentemente, las posibilidades de moverse o desplazarse. Esta limitación puede llegar a hacer que sean dependientes de otras personas o de ayudas técnicas para realizar sus desplazamientos. El conocimiento de las posibles barreras arquitectónicas que puedan impedir su libre circulación, puede suponer una ayuda importante a la hora de planificar y realizar sus desplazamientos.

1.1 Estudio de campo

En la actualidad la sociedad está trabajando para que personas con movilidad reducida puedan realizar desplazamientos con la misma facilidad que personas sin ningún tipo de problemas. Según el centro de información de las Naciones Unidas (UN [1]) más de 500 millones de personas en el mundo tienen algún impedimento físico, mental o sensorial y alrededor del 80% de esas personas viven en los países en desarrollo. El Instituto Nacional de Estadística [2], en el 2008 publicó por medio de una nota de prensa que había cerca de 4 millones de personas con discapacidad en España. Esto suponiendo casi un 9% de la población.

La principal labor en la mejora de las facilidades de vida para este colectivo ha sido eliminar aquellas barreras arquitectónicas, que por muy insignificantes que sean, pueden poner en serios apuros a cualquier persona de este colectivo. No siempre las ventajas que puede obtener este colectivo se realizan de manera física en nuestro entorno, gracias a la continua evolución de las tecnologías se están realizando esfuerzos en otros ámbitos como puede ser el desarrollo de aplicaciones. Un claro ejemplo es la aplicación Hécate, dirigida a mejorar de forma continua la accesibilidad en espacios públicos, el acceso a la información sobre el nivel de accesibilidad y en consecuencia a ayudar a todas aquellas personas con movilidad reducida a planificar sus desplazamientos dentro y fuera de su ciudad.

1.1.1 Discapacidades

Según la **Organización Mundial de la Salud** *la discapacidad es un término que abarca las deficiencias, las limitaciones de la actividad y las restricciones de la aparición. Las deficiencias son problemas que afectan a una estructura o función corporal; las limitaciones de la actividad son dificultades para ejecutar acciones o tareas, y las restricciones de la participación son problemas para participar en situaciones vitales.*

La principal discapacidad en la que se centra el proyecto es la física, también llamada *deficiencia motriz,* aunque se pueden encontrar otras tales como: *visual, mental y auditiva.* Las clases más comunes de deficiencias motrices existentes son:

Parálisis cerebral: según los médicos, se trata de un trastorno de carácter permanente que atenta contra la psicomotricidad de la persona que lo padece. Suelen producirse durante el periodo de embarazo o después del nacimiento. No es una enfermedad, no es contagiosa ni progresiva. Los desórdenes psicomotrices relacionados a la parálisis cerebral suelen presentarse junto a problemas sensitivos, cognitivos, de comunicación y percepción.

Tipos de parálisis según la afección corporal: [3]

Hemiplejia: afectación de una de las dos mitades laterales del cuerpo.

Diplejía: cuando la mitad inferior está más afectada que la superior.

Cuadriplejia: paralización de los cuatro miembros

Paraplejia: afección de los miembros inferiores.

Monoplejia: afección de un único miembro.

Espina Bífida: es un defecto de nacimiento de la columna vertebral que se presenta como consecuencia de un fallo en el cierre del tubo neural durante el primer mes de gestación. La médula espinal no se desarrolla con normalidad, apareciendo como consecuencia diferentes grados de lesión en la *médula espinal* y el *sistema nervioso. [4]*

Distrofia muscular de Duchenne: es la más común de las distrofias musculares, siendo una miopatía, de origen genético. Produce degeneración muscular, siendo hereditaria y afectando a todas las razas. Las distrofias musculares son enfermedades hereditarias, de comienzo en su mayoría en la edad infantil. Se caracterizan por atrofia progresiva muscular, perdida de reflejos y con aspecto hipertrófico de la musculatura. [5]

Distonía: es el nombre genérico de un conjunto de enfermedades neurológicas, así como de sus síntomas, que afectan a determinados músculos del cuerpo, y originan contracciones involuntarias sostenidas de tipo espasmódico, torsiones o movimientos desordenados [6].

1.1.2 Tecnología Asistiva

La tecnología está avanzando en todos los ámbitos, incluido el campo de la tecnología asistiva para personas con discapacidades. Se denomina tecnología asistiva a todo equipo, objeto, sistema, máquina, instrumento, programa o servicio que puede ser usado para suplir, compensar o mejorar las capacidades funcionales de las personas con algún tipo de discapacidad ya sea motora, sensorial o cognitiva [7].

Las personas con discapacidades motoras pueden verse en serias dificultades a la hora de utilizar dispositivos tales como ordenadores personales, teléfonos móviles, etc. Esto es debido a que las discapacidades motoras limitan el grado de movimiento, precisión y coordinación de sus extremidades y en consecuencia el uso que hacen de esos dispositivos. La forma tradicional de interactuar con un ordenador personal es mediante el uso de un teclado o ratón, necesitando de un usuario con cierta coordinación visual-motora, precisión, capacidad de movimiento y fuerza, llegando a ser imposibles de utilizar.

Por otro lado, los terminales móviles son por lo general pequeños, con botones diminutos y muy juntos entre sí en una pantalla reducida, lo que provoca que las personas discapacitadas encuentren muchas dificultades para usarlos. Para corregir estos defectos se están desarrollando nuevos elementos de interactuación entre el usuario y la máquina. A continuación se revisan algunas de las soluciones en tecnología

asistiva actuales más destacadas para la interacción de los usuarios con un dispositivo móvil:

- *Puntero:* accesorio que aumenta la precisión, útil con pantallas táctiles, cuando se usan guantes o los dedos presentan alguna deformidad.

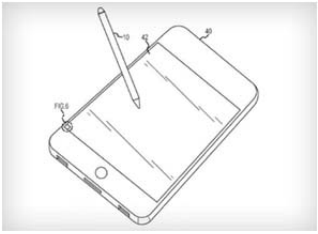

Figura 1 Puntero

- *Manos libres adaptados:* accesorio para el teléfono móvil que facilita el uso de manos libres por Bluetooth para aquellas personas con dificultades para acceder a los pequeños botones de los terminales.

Figura 2 Manos libres

- *Control por voz:* las nuevas versiones incorporan cada vez mejores asistentes con reconocimiento de órdenes por voz.

Figura 3 Control por voz

- *Teléfono con controles adaptados:* móvil pensado para aquellos con visión reducida y personas mayores. Controles y pantalla muy grandes.

1.1.3 Perfiles del usuario

El proyecto está destinado a usuarios con capacidad reducida que puedan llegar a controlar un dispositivo móvil. Su discapacidad puede ir desde el uso de un solo brazo, visión parcial o reducida, capacidad reducida para desplazamientos, etc. Se espera que el usuario disponga de buena coordinación visual/motora.

1.2 Realidad Aumentada

La Realidad Aumentada (R.A.) es un término acuñado en los años 90. Apareció después del desarrollo de un visor utilizado por trabajadores que precisaban de nueva tecnología que los guiara dentro de las instalaciones eléctricas de los aviones. El nuevo término trataba de definir la combinación de la visión de un entorno del mundo real con elementos digitales que lo enriquecieran.

La R.A. está ligada al uso de dispositivos capaces de mostrar la combinación del mundo real con la información sintética. Es una tecnología que sobrepone imágenes virtuales sobre la realidad consultada a través de una pantalla o dispositivo, por lo tanto se debe asumir que siempre irán ligados. Además, dependiendo de la tecnología los dispositivos añaden interacción con la información añadida para mejorar su utilización.

La investigación realizada en la especialidad de la R.A. explora la aplicación de imágenes generadas digitalmente en tiempo real a secuencias de video captadas con la finalidad de ampliar el mundo real. Los avances obtenidos desde los inicios incluyen e uso de pantallas colocadas en la cabeza, displays virtuales implantados en la retina para agudizar la visualización, el uso de gafas y todo ello unido al uso de sensores y actuadores para fomentar la interacción.

La R.A. puede ser utilizada en muchos campos, entre ellos por ejemplo el reconocimiento de patrones para representar imágenes o animaciones. No obstante, la aplicación *Vadeo* utiliza la rama que combina la geo-localización con información descriptiva, ya sea imágenes, audios, videos, etc. perfecta para añadir información a los puntos u obstáculos representados [8] [9].

1.2.1 Tecnología

Hardware

Los avances tecnológicos actuales en materia de dispositivos en general y en Smartphones en particular están teniendo una gran repercusión en el mundo de la R.A. En la actualidad, la gran mayoría de los Smartphone del mercado que utilizan el sistema operativo Android están equipados con cámaras digitales, sensores ópticos, acelerómetros, GPS, giroscopio, brújulas, CPUs potentes y suficiente memoria RAM para procesar cálculos. Toda esta tecnología ensamblada dentro de un mismo dispositivo posibilita la incorporación de aplicaciones de R.A. en ellos.

Software

Los dispositivos electrónicos que poco a poco van introduciéndose en la R.A. precisan de algoritmos que los controlen. Existen algoritmos que han evolucionado con el objetivo de integrar esos nuevos componentes de la manera más óptima. Su principal función es combinar la información del mundo real con información digital.

La fusión del mundo real captado con imágenes digitales de la cámara que incorpora debe ser atribuida a lugares del mundo. Toda información representada debe estar situada a partir de un sistema de coordenadas. La asociación de imágenes a coordenadas geográficas se denomina *registro de imágenes.* Este proceso utiliza algoritmos utilizados en el campo de la visión por ordenador que trabajan directamente sobre la información que contienen las imágenes captadas.

La *odometría visual,* es un proceso del que la R.A. ha heredado gran parte de métodos. Este proceso se encarga de determinar la posición y la orientación de una cámara mediante el análisis de una secuencia de imágenes adquiridas sin tener conocimiento previo del entorno. [10].

2 Estado del Arte

Las aplicaciones de R. A. para dispositivos móviles han ido apareciendo a lo largo de estos últimos años debido al aumento, tanto de las capacidades Hardware, como de los nuevos sistemas operativos que integran todos los componentes necesarios GPS, giroscopios o acelerómetros.

En el mercado actual existe un gran número de aplicaciones que incorporan la realidad aumentada para ofrecer de manera más interactiva y dinámica información sobre el entorno que rodea al usuario. Estas aplicaciones se asemejan a los objetivos que se desean alcanzar en el proyecto, aunque carecen de puntos que hacen que no todas sean válidas. Para el proyecto se está buscando una aplicación que disponga de tecnología de localización y premie la accesibilidad. La disponibilidad de documentación que el usuario pueda consultar para el aprendizaje puede valorarse como favorable aunque no es un objetivo establecido. La información a representar es otro de los puntos a destacar, la intención es que las aplicaciones puedan reproducir la información que ofrece Vadeo. A continuación se detallan las aplicaciones que, tras revisar el mercado, más se asemejan a las necesidades definidas.

2.1.1 Layar

Layar es una aplicación muy completa que utiliza su propio navegador para mostrar la información del entorno que rodeo al usuario. La información se agrupa en capas que representan una temática. El usuario puede seleccionar una o varias capas y la aplicación muestra aquellos puntos más cercanos relacionados con la información seleccionada. Dispone de herramientas para localizar la posición del usuario y realizar búsquedas detalladas del

Figura 4: Layar

entorno. Por último, dispone de una buena documentación además de un equipo de soporte técnico que puede facilitarle al usuario el proceso de aprendizaje de la aplicación [11].

2.1.2 Wikitude World Browser

Figura 5: Wikitude

Wikitude es una herramienta muy elaborada en el campo de la R.A. Utiliza capas para agrupar la información a mostrar. Los usuarios pueden asociarse a tantas capas como crean oportuno o añadir puntos en caso de estar registrado. Cuenta con un mapa donde poder consultar las capas asociadas en su página web. Entre la información que muestra se encuentran etiquetas, imágenes e incluso imágenes animadas. Con esta aplicación puede ocurrir que sea posible representar toda la información disponible en Vadeo, limitándose el contenido. Por otra parte, dispone de mucha documentación y opciones de aprendizaje que puede facilitar al usuario su utilización. Su interfaz gráfica es muy intuitiva aunque no dispone de todas las herramientas de accesibilidad que podrían permitir el acceso a la mayoría e usuarios. [12]

2.1.3 Mixare

Figura 6: Mixare

La aplicación Mixare es otra de las más destacadas en el ámbito de R.A. en dispositivos móviles. Su interfaz gráfica no es tan amigable como lo pueden ser las anteriores aplicaciones descritas, Layar y Wikitude, pero también ofrece una amplia información sobre los puntos cercanos al usuario. En este caso la

información esta organizada en una misma temática, sin necesidad que el usuario deba seleccionar las capas de información a mostrar en la pantalla. En aspectos de accesibilidad carece de herramientas que puedan reproducir el texto que aparece en la pantalla y no dispone de información de uso tan detallada como puedan tener por ejemplo las aplicaciones anteriormente descritas. [13]

2.1.4 LookAr

Figura 7: LookAR

Aplicación muy completa para localización en interior de edificios utilizando dispositivos WIFI y navegación inercial, aunque también puede ser ajustada para la localización del usuario en exteriores. Esta combinación de interior y exterior podría llegar a permitir un amplio conocimiento de toda la trayectoria por parte del usuario, aunque esa combinación excede de los objetivos buscados en el proyecto. Al igual que *Mixare*, la información no se almacena en capas y predomina una temática definida por la aplicación. Las desventajas de utilizar esta aplicación se basan en su interfaz gráfico a causa de su diseño o herramientas para elaborarlo y además del trabajo a realizar para ajustar la localización en exteriores. La accesibilidad tampoco es su punto fuerte, al no incorporar ninguna herramienta. Por último y aunque no sea un detalle crucial la documentación disponible para el usuario es bastante limitada y podría dificultar el aprendizaje de la aplicación. [15]

2.1.5 LibreGeosocial

La aplicación de R.A. LibreGeosocial ha sido desarrollada por la Universidad Complutense de Madrid. La información está agrupada en capas y puede ser consultada tanto en modalidad de realidad aumentada como también en un mapa. Está preparada para permitir a los usuarios consultar información asociada a puntos como imágenes, texto, videos o incluso audio. Incorpora una opción para que el usuario pueda añadir nuevos puntos por si mismo, los nuevos lugares pueden agregarse tomando un par de instantáneas del lugar junto con la localización del lugar. Como herramienta de aprendizaje, la web de la aplicación incorpora diferentes videos para explicar como se utiliza. Incorpora también un sistema de localización para obtener la información relacionada con la temática de la aplicación. Su amplio repertorio de información a mostrar permite que el usuario pueda consultar toda la información disponible en vadeo, además su interfaz gráfico es bastante intuitivo y fácil de utilizar. [16]

Figura 8: LibreGeoSocial

3 Materiales y métodos

El siguiente apartado realiza una descripción general de los materiales y métodos utilizados en el proyecto. En este capítulo se tratará de explicar como se ha realizado el estudio para determinar qué materiales emplear en el desarrollo.

La sección de materiales y métodos contempla dos apartados principales. El primero contiene la descripción de la plataforma Android y el entorno de desarrollo elegidos para el proyecto, Android debido a los requisitos de Vadeo. El segundo contiene la elección del Framework de desarrollo de R.A. entre las opciones disponibles. Entre ellas se encuentra Mixare, Layar, Wikitude, LookAR! y LibreGeoSocial, todas ellas muy próximas a los objetivos necesarios que deben cumplirse en el proyecto.

3.1 Android: Plataforma de desarrollo

Android es un sistema operativo móvil basado en Linux (libre, gratuito y multiplataforma), diseñado en un principio para dispositivos móviles pero en la actualidad es utilizado también para tabletas, Smart TV como Google TV, relojes y otros dispositivos que continuamente aparecen como novedades. Es desarrollado por la alianza de empresas llamada Open Handset Alliance. Esta alianza está compuesta por ochenta y cuatro empresas, lideradas por Google, y combina operadoras, empresas de software, fabricantes y compañías de comercialización: ejemplos de estas compañías son Vodafone, Telefónica, T-Mobile, Google, Ebay, SkyPop, LG, HTC o Intel.

La elección de esta plataforma para el desarrollo de la aplicación viene definida por el propio proyecto Hécate. El desarrollo de la aplicación Vadeo 2.0 de Hécate ha sido realizado para dispositivos Android, por lo tanto la aplicación de R.A. debe desarrollarse para la misma plataforma.

La plataforma Android dispone de diferentes versiones del sistema operativo que pueden elegirse para desarrollar aplicaciones. Dependiendo de la versión elegida la aplicación puede llegar a unos terminales u otros. Para este proyecto la versión elegida ha sido la API 8, que se trata de la versión 2.2 del S.O. Android llamada *Froyo*,

abreviatura de *Frozen Yogurt (yogur helado).* En la actualidad es la versión mínima recomendad para alcanzar el máximo de dispositivos del mercado, aunque el principal motivo de esta elección viene definido por el uso de la librería *OpenGL* en su versión 2.0 para la reproducción de gráficos. La nueva versión ha sido preparada para optimizar el rendimiento en los dispositivos y por consiguiente la gran mayoría de herramientas han sido desarrolladas para trabajar con esta tecnología. Por consiguiente la versión con la que se trabajará será la API 8.

A continuación se detallan algunos de los puntos más importantes que caracterizan al sistema operativo Android.

3.1.1 Estructura y características

Las características de los sistemas operativos publicados por Android han ido mejorando drásticamente en cada nueva versión. Algunas de estas características son su diseño adaptable a pantallas de mayor resolución o sus bibliotecas de gráficos 3D. También cabe destacar los sistemas de conexión disponibles como *GSM, CDMA, UMTS, Bluetooth, HSDPA* o incluso *WiMAX.* El soporte multimedia (MP4, AMR, PNG, JPG, etc.) y soporte para Streaming también son características importantes. Por lo general incorpora soporte para la gran mayoría de avances tecnológicos del mercado: como por ejemplo multitarea, multi-táctil, video llamada o Tethering (usar el móvil como punto de conexión para otros dispositivos por Wifi).

La siguiente imagen muestra una visión general de la arquitectura del sistema operativo, donde se puede observar que está formada por cuatro capas.

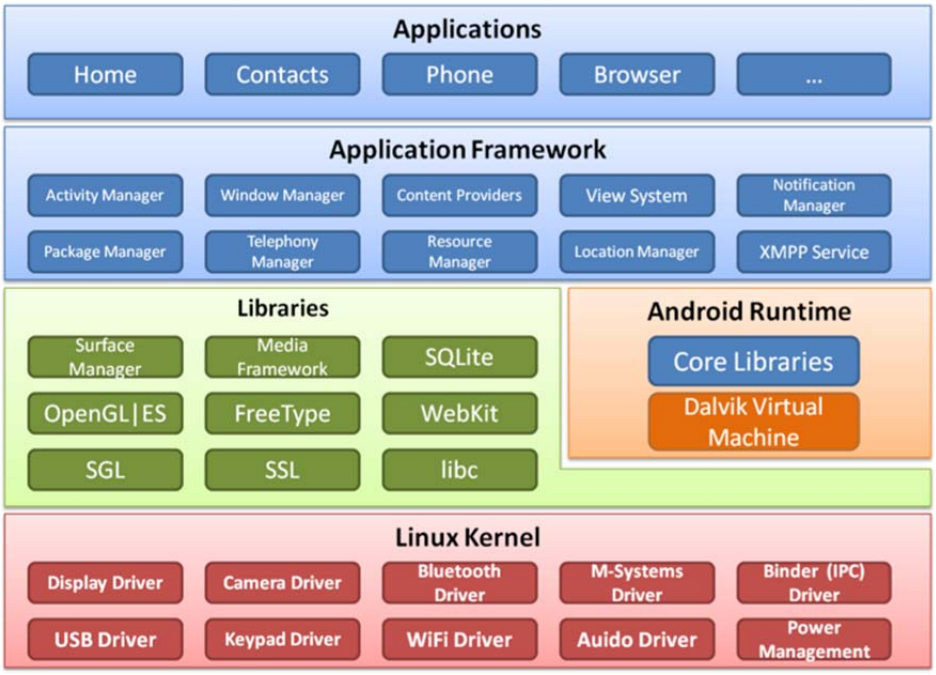

Figura 9: Android Core

A continuación se describen las principales características de cada una de las capas presentes en la arquitectura.[17]

3.1.1.1 Linux Kernel

El núcleo de Android está formado por el sistema operativo Linux con algunas modificaciones y características que se ajustan a las limitaciones de los dispositivos. Actúa como una capa de abstracción entre el Hardware del dispositivo y el resto de la pila de software, proporcionando interacción entre servicios como seguridad, manejo de memoria, multiproceso, pila de protocolos y soporte de drivers para dispositivos entre otros.

3.1.1.2 Android Runtime

El Runtime de Android es una Máquina Virtual llamada *Dalvik VM.* Es el componente principal del entorno de ejecución y precisa del uso de librerías del sistema para su funcionamiento. Ha sido diseñada como una variación de la MV de Java pero, aunque utilice su mismo lenguaje no son compatibles debido a su compilación en un formato específico (*.dex).*

Su diseño ha sido optimizado al máximo para reducir al mínimo el uso de recursos y su ejecución está basa en registros debido a que los teléfonos móviles están optimizados para esta misma ejecución. [18]

3.1.1.3 Libraries

Son un conjunto de librerías escritas en C/C++ y compiladas para el hardware del dispositivo. La gran mayoría son desarrolladas por los fabricantes para facilitar operaciones repetitivas de comunicación con el dispositivo. Su uso libra a los desarrolladores de codificar ciertas operaciones y garantiza que éstas se realicen correctamente. [17][18]

Las librerías más comunes en los dispositivos son:

- Administrador de Interfaz Gráfica: librería para la comunicación con los componentes relacionados con la interfaz gráfica del sistema.
- Base de datos SQLite: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- API de programación de gráficos para OpenGL ES 2.0 3D: librería para el desarrollo de gráficos en 2D y 3D.
- Motor de renderizado WebKit: librería que funciona como base para los navegadores. Safari o Chrome son algunos navegadores que la utilizan.
- Protocolos de comunicación SSL: protocolo de comunicación que permite la encriptación de las comunicaciones.

3.1.1.4 Applications Framework

El *Framework* se compone de todas las clases y servicios que disponibles para que las aplicaciones puedan realizar sus funciones. La gran mayoría de estas librerías están preparadas para comunicarse con la capa *Libraries* por medio de *Dalvik*. Su uso

se extiende también a los desarrolladores que disponen de acceso a ellas para realizar las mismas operaciones, realizándose una reutilización de este código. [19]

3.1.1.5 Applicactions

La última capa incluye todas las aplicaciones instaladas en el dispositivo. Entre ellas se encuentran las preinstaladas por los distribuidores (*gMail, Cámara, etc.),* las nativas y aquellas instaladas por el usuario.

3.1.2 Hardware

En la actualidad la plataforma Android está disponible no solo para móviles sino también para Tablets, dispositivos para ver la televisión (Google TV) o incluso PCs y portátiles. Su mercado alcanza todo tipo de dispositivos y cada día aparecen nuevos dispositivos que incluyen nuevas funcionalidades. Ejemplos de estos dispositivos son:

- Google publicó el anunció de sus gafas que utilizan la tecnología de Realidad Aumentada basada s en Android.

Figura 10: Gafas R.A.

- Aparece el Mini-PC Mk802 killer: mini procesador doble núcleo ARM Cortex-A9 a 1.6GHz, gráficos Malí 400, 1GB de RAM y 4GB de almacenamiento interno.

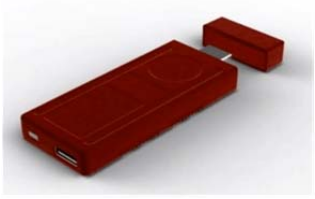

Figura 11: Mini-PC

- Samsung Galaxy Note 10.1: procesador cuatro núcleos a 1.4GHz con 2GB de RAM, memoria interna de hasta 64GB, WIFI, HSDPA, GPS, dos cámaras de hasta 5GB.

Figura 12: Samsung Galaxy Note

- Google Nexus D: un Quad Core a 1.8GHz con 1.5GHz de RAM, memoria interna de hasta 64GB, conectividad NFC, pantalla flexible…

Figura 13: Dispositivo móvil última genración

3.1.3 Herramientas de desarrollo

Android dispone de diferentes herramientas o editores para generar aplicaciones, aunque por lo general, se utiliza el SDK (Software Development Kit) que incluye todos los componentes necesarios para poder generar código y probarlo con emuladores entre otras funciones más.

La masiva utilización de este lenguaje y el aporte que realiza Google ha contribuido en la aparición de otros entornos de desarrollo, como por ejemplo *App Inventor*, un editor web para desarrolladores noveles. A continuación se describen algunas de estas herramientas de desarrollo.

3.1.3.1 Android SDK (Software Development Kit)

El SDK de Android está compuesto por un conjunto de herramientas diseñadas para facilitar el desarrollo de aplicaciones. Estas herramientas incluyen un depurador de código, bibliotecas, un simulador de dispositivos móviles, documentación, ejemplos de código y sobre todo tutoriales.

El SDK por si solo no puede ejecutarse, precisa de una plataforma de desarrollo (IDE, Entorno de Desarrollo Integrado) que lo contenga. *Eclipse* es la plataforma oficial que permite integrar todo el conjunto de herramientas, también llamado ADT (*Android Development Tools)* plugin. Asistentes para la creación de proyectos, automatizaciones en la compilación y editores de interfaz gráfico, son algunos de los elementos destacables. [20]

3.1.3.2 NDK (Native Development Kit)

El NDK es un Framework que facilita la utilización de librerías desarrolladas en C/C++ dentro de aplicaciones Android. Las librerías deben ser compiladas previamente para ARM o código x86 nativo. Su ejecución en aplicaciones que incorporan este Framework se realiza del mismo modo que las aplicaciones convencionales ejecutadas en la maquina virtual *Dalvik,* aunque el código de las librerías se ejecuta directamente, es decir, fuera de la máquina virtual, reduciendo la latencia en la comunicación con el Hardware.

El uso de esta tecnología permite incorporar código desarrollado en otras plataformas para mejorar el desarrollo o la comunicación con partes del Hardware del dispositivo. Las librerías desarrolladas en los distintos lenguajes, comúnmente en C/C++, pueden no ser escritas en el lenguaje Java y por lo tanto no se podrían utilizar. Con este *Framework* se asegura que estas librerías o sus actualizaciones se podrán incorporar en el mismo instante que se publiquen. [20] [21]

3.1.4 Diseño de aplicaciones accesibles

Android proporciona una capa de accesibilidad que ayuda a usuarios con discapacidades a navegar por las aplicaciones de un modo más sencillo. La capa de accesibilidad incorpora servicios como *Text-to-Speech* para la reproducción de textos, *Retroalimentación Háptica* para dar una respuesta física a las pulsaciones realizadas sobre la pantalla y navegación utilizando elementos como *Trackball* (bola del dispositivo que actúa como jostick) o d-pad (flechas de navegación Arriba-Abajo-Derecha-Izquierda).

3.1.5 Modelo de negocio

Android empezó como un sistema operativo diseñado para dispositivos móviles. Poco a poco ha ido evolucionando en muchos aspectos tanto tecnológico como económico. Las aplicaciones desarrolladas pueden acogerse a diferentes modelos de negocio, siendo por ejemplo la palabra gratis un sinónimo de obtención de beneficios en el caso de que la aplicación pueda tener éxito.

El desarrollo que se ha realizado en el proyecto Vadeo no se ha centrado en el objetivo de obtener beneficios y si en el objetivo de desarrollar un proyecto que pueda mejorar la calidad de vida de las personas con movilidad reducida. Una vez se consiga una versión estable del proyecto este objetivo puede variar y para ello es imprescindible tener en cuenta cómo aprovechar al cien por cien el desarrollo realizado. En ningún momento se ha cerrado la puerta a una futura explotación para obtener beneficios y por lo tanto el control de las licencias que se utilicen o el modo en

que se publique la aplicación van a tener una gran repercusión en este proceso tan delicado como puede ser la elección y explotación de un modelo de negocio.

A consecuencia de una posible futura explotación de la aplicación, este apartado procede a exponer los diferentes modelos de negocio que poco a poco se han ido asentando en el sector económico que engloba el mundo de Android. Es importante conocer como funciona el mercado en la actualidad para saber donde puede terminar recayendo el proyecto.

La descripción del modelo de negocio de aplicaciones Android, centrado en la utilización de la aplicación Google Play como medio de publicación y distribución, puede ayudar a entender las diferentes opciones del mercado para obtener beneficios. La siguiente imagen (Figura 14: Modelo de negocio software Android) expresa una jerarquía basada en Google Play. Cada elemento representa los diferentes actores que se benefician directa o indirectamente de este mercado de aplicaciones. Desde los usuarios finales consumidores de ellas pasando por los desarrolladores encargados de realizarlas e incluso teniendo presente entidades que puedan necesitarlas para mejorar el servició que ofrecen a sus clientes.

Figura 14: Modelo de negocio software Android

- **Usuario final**: Usuarios o entornos corporativos que consumen las aplicaciones.
- **Empresas especializadas en publicidad**: empresas que suministran publicidad a las aplicaciones interactuando entre desarrolladores y quien desea publicitarse.
- **Empresas que ofrecen servicios como valor añadido a sus clientes**: empresas que ofrecen un servicio y utilizan las aplicaciones Android para que el cliente pueda acceder. Este modelo sería adaptable para Vadeo si se vendiera como servició web y se utilizase la aplicación Android como medio de consulta.
- **Desarrollador independiente**: desarrolladores por cuenta propia con la finalidad de vender las aplicaciones que desarrollan.
- **Factorías de software**: empresas que desarrollan aplicaciones bajo demanda de terceros.
- **Tiendas de aplicaciones**: Entorno donde se publican las aplicaciones para que el público pueda comprarlas bajo la supervisión de la plataforma que las publica.

Android como plataforma de desarrollo permite un rápido y gratuito acceso a todas las herramientas necesarias, esta es una ventaja para empezar con el desarrollo de aplicaciones. Después de describir aquellos actores involucrados en el desarrollo de aplicaciones acto seguido se describen los principales modelos de negocio.

- **Venta en tiendas online**: Las tiendas de aplicaciones online permiten publicar las aplicaciones para que estén accesibles a un público mayoritario. Por cada una de las ventas remuneradas que se realicen las tiendas se llevan una comisión.
- **Distribuyendo gratuita**: Este modelo contempla la distribución de la aplicación gratuitamente pero con la condición de que aparezca publicidad dentro del contenido. Por cada acceso a la publicidad realizado por medio de la aplicación el usuario recibe un benefició.
- **Utilizar la modalidad Freemium**: La modalidad *Freemium* se caracteriza por la publicación gratuita de aplicaciones limitando el contenido. El usuario puede adquirir nuevas herramientas o pantallas abonando una cantidad económica.
- **Empresas de consultoría o desarrollo a medida:** desarrollos de aplicaciones para terceros bajo demanda. Tareas de formación a futuros desarrolladores. [22]

3.1.6 Problemas localizados en Android

3.1.6.1 Fragmentación

La fragmentación es un problema que se ha creado con la aparición o evolución de las nuevas versiones del sistema operativo. Cada nueva versión del S.O. publicada incorpora herramientas avanzadas que no son compatibles con las versiones anteriores. Las aplicaciones que se desarrollan utilizando esos avances son incompatibles con versiones del S.O. anteriores y por lo tanto no pueden ser instaladas en ellas. Este problema afecta en gran medida aquellos dispositivos con versiones del S.O. antiguas.

3.1.6.2 Estándares de Java

Android no utiliza los estándares establecidos de Java, por ejemplo Java SE y ME, lo cual es un problema para el desarrollo. Esto impide la compatibilidad entre aplicaciones Java escritas para otras plataformas. Android sólo utiliza la sintaxis y la semántica de Java, pero no incorpora en su totalidad las bibliotecas de clases de Java y APIs que acompañan a Java SE o ME. Sin embargo, hay diversas herramientas en el mercado de empresas como Myriad Group y UpOnTek que dan un servicio de conversión entre J2ME y Android.

3.2 Framework de desarrollo de R.A.

Las aplicaciones descritas en el apartado 2, Estado del Arte, tienen características similares a los requisitos establecidos en el proyecto, pero ninguna cumple con todos ellos. Algunos de los puntos que se incumplen son:

1. No pueden ser integradas como parte de la aplicación móvil de Vadeo.
2. Son difíciles de integrar con el modelo de datos establecido en Vadeo.
3. No son ampliables para incorporar elementos informativos de Vadeo.
4. No incorporan herramientas de accesibilidad.

Para cumplir con los objetivos establecidos es necesario desarrollar una aplicación de R.A. propia o en su caso editar alguna de las existentes en el mercado. Los siguientes puntos analizan cada una de las herramientas de desarrollo disponibles con el fin de obtener una aplicación de R.A. que cumpla con los objetivos marcados y tenga en cuenta las restricciones impuestas por el proyecto Hécate del que forma parte el presente trabajo.

Teniendo en cuenta los objetivos fijados en el presente proyecto y las particularidades heredadas de Hécate, las principales características que las herramientas de desarrollo deben cumplir y que ayudarán en la elección de la herramienta más óptima son:

- Disponibilidad del código fuente para realizar mejoras.
- Utilización de formato JSON para comunicación con el servidor.
- Herramientas de accesibilidad o en su ausencia posibilidad de incorporarlas.
- Utilización del patrón de diseño de R.A. o en su ausencia la posibilidad de crearlo. El patrón de diseño que siguen por regla general las aplicaciones de R.A. es un estándar de facto. La gran mayoría de las aplicaciones reparten el contenido situando un radar en las esquinas superiores, los puntos de información en el centro y las descripciones en la parte inferior.
- La información mostrada sobre un punto, pueda incluir comentarios, fotos, títulos, descripciones, iconos de accesibilidad y coordenadas.
- Licencias que permitan la libre explotación de la aplicación.
- Es preferible que no se utilicen capas para acceder a la información.

3.2.1 Mixare

Mixare es una librería de R.A. que cuenta con el respaldo de una comunidad de desarrollo abierta. Su código fuente está disponible para ser descargado y editado bajo licencia **GPLv3**, con lo cual, permite realizar los ajustes necesarios según necesidades del proyecto pero obliga a la publicación del código fuente.

La comunicación con el servidor se realiza de manera directa sin uso de intermediarios. En un principio existe una estructura de datos definida que utiliza JSON como lenguaje de comunicación, pero puede ser ajustada para transferir aquello que sea necesario (imágenes, comentarios, etc.). La documentación disponible explica paso a paso como configurar el servidor con código fuente y ejemplos en *PHP.*

El uso de la aplicación en modo *offline* también está incluido utilizando una base de datos, esta característica no tiene tanta relevancia ya que la aplicación precisa

de información que puede actualizarse diariamente, en consecuencia es recomendable que disponga de conexión de datos.

La librería ha sido desarrollada y testeada para garantizar que funciona con versiones superiores a la elegida para el proyecto. Esto garantiza que tanto dispositivos con la versión del proyecto como con versiones superiores van a poder utilizar la aplicación ya que no utiliza recursos que se hayan descartado en versiones superiores.

En la documentación disponible no determina si utiliza herramientas para mejorar la accesibilidad. De todos modos, como ocurre con los otros puntos, al disponer del código fuente permite que pueda ser añadido a posteriori.

La implementación actual utiliza un navegador lanzado como una aplicación. Está enfocada a enlazar los puntos con páginas webs. Esto puede ser utilizado para que las consultas de los puntos no recaigan sobre la aplicación, todo lo contrario al objetivo de que todo aparezca dentro de ella.

3.2.2 Layar

Layar es un navegador desarrollado para mostrar información segmentada en capas. Cada una de las capas almacena la información correspondiente a un tema publicado por un desarrollador. Su utilización implicaría la creación de una *capa* que contendría toda la información relacionada a Hé*cate.* De un modo más técnico una capa almacena la información de conexión con el servidor que contiene los puntos de información.

En la actualidad, el Framework Layar no ha desarrollado aun ningún kit de desarrollo *SDK* para usuarios Android, por lo tanto la única opción vigente para utilizar esta herramienta se centra en la utilización de las capas para segmentar la información. [23]

El uso de la librería de Layar obligaría a dirigir todas las comunicaciones hacia un servidor intermedio o *Proxy* encargado de redirigir las peticiones hacia el servidor de datos (*Hécate)* y luego de validar la información devuelta. Esta validación sirve para asegurar que la información devuelta contiene información correcta que pueda ser

representada en el navegador. Las comunicaciones se realizan utilizando JSON y dispone de una documentación detallada para realizar el montaje del servidor web y todo el esquema de bases de datos, documentación que facilita el ensamblaje con la información definida en el servidor de datos del proyecto.

La no posibilidad de descargar el código fuente es un aspecto negativo que contemplar. Los puntos que aparecen en la pantalla solo muestran texto e imágenes, sin disponer de la posibilidad de enlazar comentarios. El modelo de datos de Hécate incluye el uso de comentarios sobre los puntos y debido a que el código fuente no está disponible se imposibilita agregar los comentarios u otra información extra.

Por último, la aplicación es gratuita siempre que no se supere un número máximo de peticiones. Es otro de los aspectos negativos en caso de alcanzar un volumen considerable de usuarios.

3.2.3 Wikitude

Wikitude es una herramienta que ofrece la posibilidad de incorporar el servicio de R. A. dentro de las aplicaciones. Incluye un kit de herramientas de desarrollo *SDK* para la integración de un navegador encargado de mostrar la información. Su uso puede ser limitado dado que utiliza una licencia privada y no ofrece la posibilidad de acceder al código fuente para realizar ajustes en el contenido.

El SDK de desarrollo está basado en estándares de la tecnología como *HTML, Javascript y/o CSS* que favorecen la fácil integración de las tecnologías. Además de tener una amplia documentación y un portal con amplia cobertura, foros o también entorno de desarrollo, el Framework consume el mínimo espacio posible garantizando que la aplicación no exceda en tamaño.

Respecto a las limitaciones, cabe destacar el hecho de que la comunicación con le servidor se realiza por medio de un *Proxy* pero sin utilizar JSON para la comunicación, en su lugar se utiliza un formato propio llamado *ARML* (XML personalizado).

El uso de Wikitude es gratuito en el caso de que la aplicación no esté destinada a proyectos comerciales, de lo contrario debe pagarse una licencia por su uso, mantenimiento y número de instalaciones que se realicen del producto. Su instalación requiere dispositivos con API 8 o superior.

La web de la aplicación dispone de un apartado para los desarrolladores, en él se pueden publicar todas las dudas o consultar tutoriales.

3.2.4 LookAR!

LookAR es un Framework de R.A. preparado para trabajar en localizaciones internas de edificios utilizando dispositivos WIFI y navegación inercial. Su código fuente está disponible para ser descargado y editado según necesidades, entre ellas hay que destacar que puede ajustarse para utilizar coordenadas GPS para la localización. También contiene aplicaciones que han utilizado este *Framework* para descargar y consultar su código fuente.

El Framework no dispone ni de SDK ni de ningún interfaz implementado para mostrar información acerca de puntos o su descripción. Por el contrario, ofrece métodos para facilitar la implementación del diseño desde cero, dado que de lo contrario habría que interactuar con la librería OpenGL encargada de realizar estas operaciones e incrementaría la complejidad del desarrollo.

La comunicación con el servidor debe ser configurada por el desarrollador, esta flexibilidad es idónea para poder enviar la información ajustándola a las características u objetivos a representar (imágenes, comentarios, etc.)

El soporte técnico disponible es bastante básico. El equipo de desarrollo resuelve las dudas que se les envían con cierto tiempo de margen, esto es debido a que el proyecto fue terminado hace tiempo y no ha tenido más continuidad. En este caso es interesante que el proyecto haya terminado ya que todas las mejoras se realizan según necesidades sin tener en cuenta futuras versiones que puedan sobre escribir los cambios realizados.

Por último, ha sido desarrollada para la API 10 pero su funcionamiento está garantizado para la API 8 y la licencia utilizada ha sido GPLv3.

3.2.5 LibreGeoSocial

LibreGeoSocial ha sido desarrollada como una aplicación destinada a mostrar información agrupada en capas. En ningún caso ha sido desarrollada como Framework o librería pero su código fuente disponible permite extraer toda la lógica para incorporarla a cualquier proyecto. Su interfaz gráfica no incorpora aspectos de accesibilidad, pero al disponer del código fuente pueden ser añadidos.

La información se representa en capas que apuntan a los respectivos servidores donde están almacenadas, pero es completamente configurable para poder eliminar las capas y que directamente muestre la información correspondiente al proyecto.

El proyecto de LibreGeoSocial fue finalizado y no está preparado para que se realicen modificaciones, esto garantiza que los cambios que se realicen no sufrirán conflictos con publicaciones de futuras versiones, y permite crear una línea de desarrollo propia para ajustar el código a las necesidades de representación que se necesiten. No dispone de una comunidad de desarrollo pero si suficiente documentación para utilizarlo y además los desarrolladores son participes de resolver dudas sin ningún inconveniente.

El diseño y estructura de LibreGeoSocial puede ser aprovechable ya que cumple con el patrón de diseño de R.A. aunque algunos aspectos gráficos deben ser mejorados.

Por último, la licencia de publicación fue *Creative Commons* eliminando restricciones que obliguen a publicar la aplicación y restrinjan una futura explotación de la misma, uno de los objetivos marcados. Además, no contempla el pago de ninguna tarifa.

3.2.6 Resumen

La siguiente tabla, Tabla 1: Comparativa de características para el desarrollo, realiza una comparación de los puntos más importantes de cada uno de los apartados anteriores.

	Layar	Wikitude	Mixare	LookAR	LibreGeoSocial
Documentación y soporte	✓	✓	✓	✓	✓
Licencia sin restricciones de publicación o venta	✗	✗	✗	✗	✓
Sin Tarifas por uso	✗	✓	✓	✓	✓
Código fuente disponible	✗	✗	✓	✓	✓
No utiliza navegador Propio	✗	✗	✗	✓	✓
Servidor y comunicaciones configurables	✗	✗	✓	✓	✓
Accesibilidad configurable	✗	✗	✓	✓	✓
Información mostrada por puntos ajustable	✗	✗	✓	✓	✓

Tabla 1: Comparativa de características para el desarrollo

En la Tabla 1: Comparativa de características para el desarrollo se observa que solo la aplicación de *LibreGeoSocial* cumple con todos los objetivos marcados. Aspectos como el *código fuente disponible* influyen en la posibilidad de poder ajustar completamente la aplicación. La disponibilidad del código fuente para tener acceso a

todo el contenido es uno de los aspectos que más se ha tenido muy en cuenta. Herramientas como Layar o Wikitude, ambas punteras en el ámbito de la R.A. han sido descartadas, por dicha limitación y por el uso de tarifas, al incumplir con los objetivos marcados en el proyecto. Otros aspectos como la licencia determinan la explotación que pueda realizarse tras su publicación una vez terminada. Tal es la repercusión derivada por el uso de licencia *GPLv3* que tanto Mixare como LookAR son descartadas por ser publicadas bajo dicha licencia, hecho que obligaría a publicar el código fuente de la aplicación de R.A. Utilizar *LibreGeoSocial* permite decidir libremente la liberación del código o no.

Que la información mostrada de cada punto se ajuste a las necesidades heredadas de Hécate también es un aspecto muy importante a tener en cuenta. Solo aquellas aplicaciones con el código disponible garantizan que este punto pueda cumplirse al cien por cien.

Para finalizar, el diseño de la interfaz gráfica va a repercutir en mayor medida con la aceptación de la aplicación. Frameworks como *LibreGeoSocial o Mixare* han sido desarrollados siguiéndolo, todo lo contrario que *LookAR*. Aunque LookAR ofrezca herramientas para su creación y elaboración, la realización de la interfaz gráfica puede ser muy costosa tanto en preparación como en tiempo de desarrollo. La disponibilidad del código fuente permite que con la herramienta *LibreGeoSocial* puedan, perfectamente, adaptarse todas las necesidades que surjan en Vadeo. De hecho, gran parte del diseño puede ser reaprovechado e incluso mejorado en todos y cada uno de sus aspectos.

Por todo esto, el Framework elegido para realizar el desarrollo del proceso de R. A. en la aplicación Vadeo 2.0 es *LibreGeoSocial.*

4 Hécate: Aplicación de Realidad Aumentada

En este capítulo se introducen las principales características del proyecto Hécate en el que se enmarca el presente trabajo, para a continuación pasar a describir los requisitos y la funcionalidad de la aplicación de Realidad Aumentada desarrollada y su aportación al proyecto.

Como se ha comentado el proyecto Hécate está formado por dos herramientas o aplicaciones independientes retroalimentadas entre sí, la Herramienta de Gestión integral de Accesibilidad (GIA) y la aplicación de Consulta, Información y Rutas (Vadeo 2.0), todas ellas interactuando con el servidor de Hécate. En la siguiente figura, Figura 15: Arquitectura Hécate, se pueden ver todos estos elementos y la forma en que interactúan entre sí. El primer punto de conexión con el sistema está representado por el personal del ayuntamiento y los técnicos de accesibilidad a través de la herramienta GIA en versión web, que se trata de una aplicación web. Por su parte, los ciudadanos pueden acceder al sistema a través de la aplicación Vadeo 2.0 en su versión web y móvil. Por último el administrador del sistema accede vía web a través de la aplicación web de Vadeo. En la siguiente figura se pueden ver todos los actores involucrados y las diferentes modalidades de conexión con el sistema, en este apartado se describe en detalle la aplicación móvil para Android, ya que es el medio de acceso hacia la aplicación de R.A.

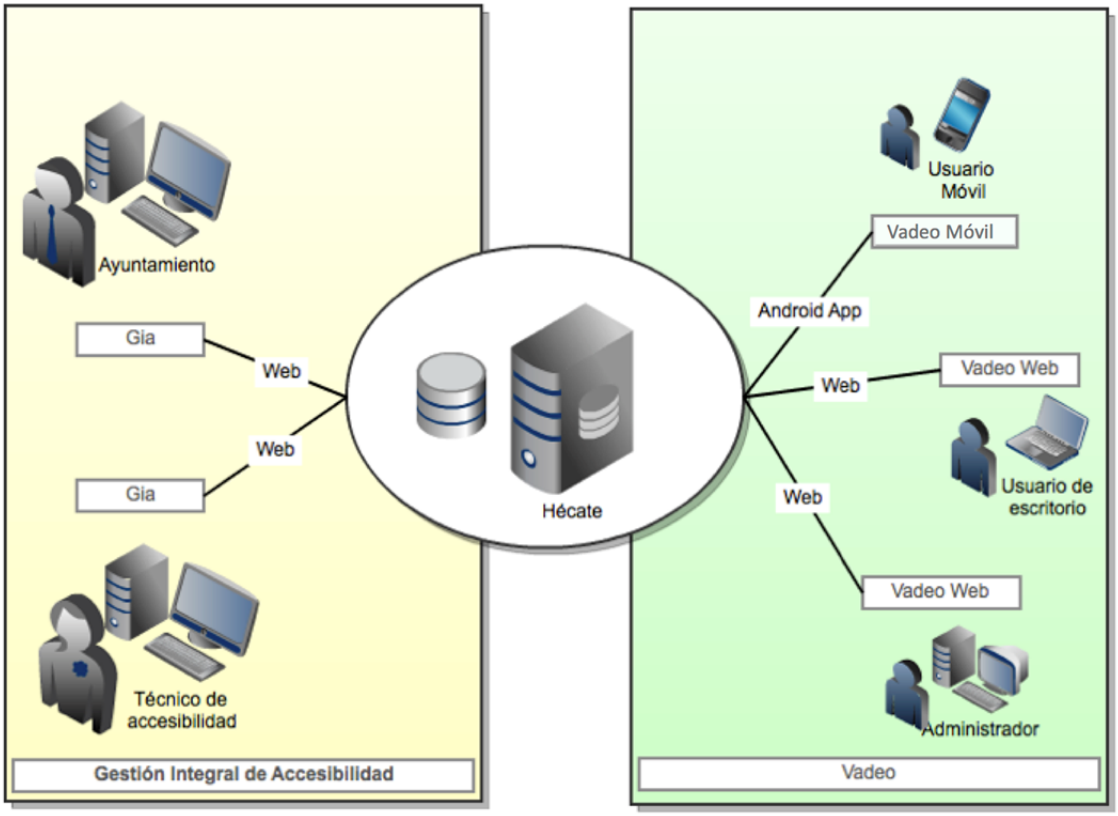

Figura 15: Arquitectura Hécate

4.1 Introducción a la funcionalidad

Antes de comenzar con la descripción de la aplicación, es importante detallar la información sobre el nivel de accesibilidad del entorno que va a ser mostrada a través de la aplicación de R.A.

La información a mostrar puede ser separada en dos grupos, en el primero entrarían los objetos que representan una barrera desde el punto de vista de la accesibilidad, llamados *obstáculos* o *Puntos del Mapa,* y en el segundo grupo se incluyen los *Puntos de Interés* o *PDI* que incluyen información relacionada con el ocio o servicios. A continuación se describen estos conceptos en detalle.

- *Obstáculo o Punto del Mapa:* objeto que reside en la calle o por una zona de viandantes y que supone una barrera para progresar en el camino o para acceder a un lugar determinado desde el punto de vista

de la accesibilidad al medio físico. Ejemplo: escaleras, aceras con bordillos, obras en las aceras, etc. La información que el usuario podrá consultar es la siguiente:

- Nombre o título del obstáculo.
- Descripción del obstáculo: breve descripción sobre el obstáculo.
- Tipo de Punto: ofrecerá información gráfica que indique el tipo de obstáculo, diferenciando entre si es un itinerario, vado, escalera, rampa, calle con pendiente, obstáculo vario, obras, banco, buzón, cabina, contenedor, fuente u otro tipo distinto.
- Ubicación: geo-localización del obstáculo.
- Atributos de accesibilidad de un punto del mapa: información gráfica certificada que representa los atributos de accesibilidad presentes o su ausencia en un punto del mapa: escalera, accesos, itinerarios anteriores, rampas y ubicación.
- Comentarios de otros usuarios.
- Imágenes del obstáculo.

- *Punto de Interés o PDI:* edificio, lugar o zona que ofrece alguna utilidad frecuentemente relacionada con el ocio o servicios. Ejemplo: restaurantes, oficinas de información, aparcamientos, paradas de metro o bus, etc. La información que el usuario podrá consultar es la siguiente:
  - Nombre o título del PDI.
  - Descripción del PDI: breve descripción sobre el punto de interés.
  - Tipo de PDI: ofrecerá información gráfica que indique el tipo de PDI de entre los siguientes: aparcamiento, aparcamiento reservado, bar/restaurante, comercio, turismo, ocio nocturno, ocio infantil, deportes, educación, taxi, tranvía, tren, autobús, metro y otros.

- Ubicación: geo-localización del PDI.
- Comentarios de otros usuarios
- Imágenes del PDI
- Web: enlace a la web del PDI.
- Iconos accesibilidad: información gráfica certificada que representa los distintos equipamientos o su ausencia, relacionados con la accesibilidad, que dispone el PDI: accesos adaptados, aseos adaptados, aparcamiento reservado, ascensor, rampa de acceso, deficiencias visuales y/o auditivas.

Esta información, que será obtenida del servidor común de Hécate, será representada en el dispositivo en forma de *Galería*. La Galería ofrece una filosofía de interacción sencilla y potente al mismo tiempo, e influye a la hora de organizar y representar la información.

- *Galería:* es una vista disponible en Android que permite cargar muchos elementos e ir mostrándolos uno a uno, desplazándolos de derecha a izquierda o de izquierda a derecha. Muy útil para mostrar muchos elementos como por ejemplo imágenes, textos, etc.

Figura 16: Iconos y figuras informativas de puntos u obstáculos

4.2 Especificación funcional

La R.A. estará disponible para todos los usuarios que accedan a la aplicación. En este apartado se abstiene de cómo el usuario accede en la aplicación móvil y se centra en la descripción suponiendo que el usuario dispone de acceso directo.

Por lo general, el usuario podrá consultar en la pantalla del dispositivo información del mundo real, captada por la cámara, combinada con información sintética que describa los puntos de interés y obstáculos, obtenida del servidor de Hécate.

La información del punto consultado, será descrita en la propia pantalla de la R.A., de forma que se traslade el mayor número posible de detalles de la forma más gráfica posible. En el siguiente punto viene detallada toda la funcionalidad disponible en la aplicación de R.A.

4.2.1 Funcionalidad

A continuación quedan detallados todos y cada uno de los aspectos de la funcionalidad de la aplicación.

Figura 17: Aplicación Hécate

- *Acceso al apartado de R.A.*: para ello accede desde el menú de opciones disponible para los dispositivos Android. Una vez desplegado el menú solo tiene que pulsar el botón de “Realidad Aumentada”.

- *Solicitud activación GPS*: en caso de no disponer del GPS activo se le solicitará vía mensaje en la pantalla. Al aceptar el mensaje, automáticamente se mostrará la pantalla para que active el GPS. Al volver ya se cargará la pantalla de R.A.

- *Radar:* el usuario podrá consultar en un radar la cantidad de puntos cercanos a su posición, determinados por el radio de consulta establecido.

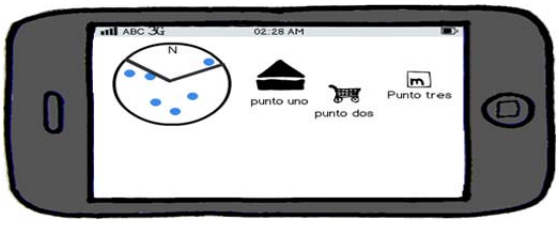

Figura 18: Mockup radar

- *Flecha:* en el caso de que el usuario esté consultando los puntos por la galería (definida a continuación) aparecerá una flecha que indique en qué dirección ha de enfocar la cámara para que aparezca el punto en pantalla.

- *Puntos en la pantalla:* iconos con descripción añadida que indicarán al usuario los puntos disponibles más cercanos. El usuario podrá seleccionarlos pulsando en ellos para que aparezca una galería con la información del punto indicado.

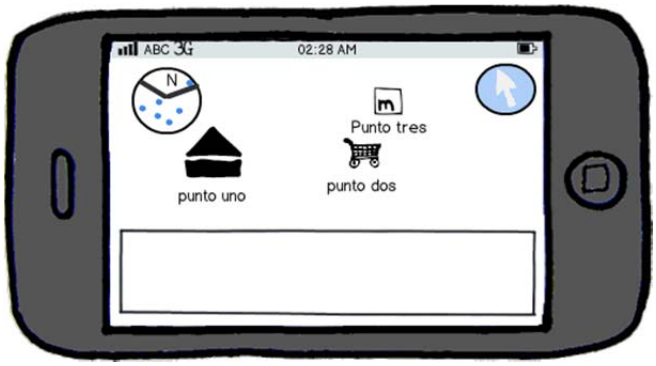

Figura 19: Mockup puntos pantalla y flecha

- *Consulta información Puntos:* al seleccionar un punto de la pantalla dispondrá de una sección de información en la pantalla donde podrá realizar diversas operaciones.

    - *Lectura información:* información del título del punto y descripción en caso de que exista, listado de iconos que

describen las accesibilidades disponibles en el punto y la distancia hasta el punto.

- *Text to Speech:* opción que leerá el texto que aparezca en la pantalla.

- *Consulta Imágenes:* botón que, en caso de que existan imágenes asociadas al punto, mostrará una galería con las imágenes.

- *Consulta de comentarios:* botón que, en caso de que existan comentarios asociados al punto, mostrará una galería con los comentarios.

- *Botón de cierre:* botón para cerrar el menú. Además de este botón incorporará un temporizador para cerrarlo pasado un tiempo.

- *Desplazamiento hacia otros puntos:* la información será mostrada en una galería, por lo tanto se puede desplazar a izquierda o derecha para ver la información de los otros puntos representados en pantalla. En la parte inferior aparecerá en qué punto se encuentra y el total de puntos. Si se consultasen puntos que no están dentro del ángulo de visión se cargará una flecha que indica en qué dirección hay que enfocar la cámara para que aparezca.

Figura 20: Mockup con la descripción de un punto y las opciones de imágenes, comentarios, text-to-speech y cerrar

- *Pausa o Play:* el usuario dispone de un botón de Pause para detener el contenido mostrado en la pantalla con el fin de que pueda seleccionar un icono fácilmente. Después aparecerá el botón de Play para que pueda volver a mostrarse el contenido actual.

Figura 21: Mockup botón Pause y barra para establecer el radio

- *Radio de distancia:* opción para poder establecer el radio máximo para el que se muestran puntos por pantalla.

- *Opciones de filtrado:* la cantidad de información que aparezca por pantalla será filtrable según el tipo de punto, obstáculo o distancia elegida. Los puntos también pueden filtrados por ser oficiales o sociales. Los primeros han sido verificados por entidades tales como ayuntamientos o técnicos e accesibilidad. Los segundos son añadidos por otros usuarios registrados en el sistema.

Figura 22: Mockup filtrado puntos

4.3 Requisitos de usabilidad

La usabilidad es un término importado de la lengua anglosajona que no dispone de definición oficial en la RAE (Real Academia Española) pero que es muy utilizado en ámbitos tecnológicos. El concepto de usabilidad trata de expresar la facilidad con que un usuario puede utilizar una herramienta fabricada por terceras personas con el fin de alcanzar un objetivo. La usabilidad está ligada con el diseño o interfaz de la herramienta. Destacan aquellos diseños simples, fáciles, cómodos y sobre todo prácticos. [24]

El principal objetivo ha sido crear un apartado de R. A. para que personas con movilidad reducida puedan obtener información acerca del nivel de accesibilidad en tiempo real del espacio público que les rodea, con el fin de facilitar sus desplazamientos.

Los siguientes puntos detallan todos los requisitos primarios que deben cumplirse para que sea una aplicación usable:

- Mostrar R.A.: mostrar en la pantalla del dispositivo todo lo que capture la cámara más la información sintética con opción a interactuar con ella.

- Adaptar el Framework para que sea lo más útil posible: el *Framework* está preparado para representar la información pero se le deben realizar ajustes para facilitar el uso que puedan realizar personas con discapacidad. Mejoras como pausar el contenido de la pantalla o reproducir el texto.

- Crear una interfaz gráfica lo más usable, simple, funcional, fácil e intuitiva posible: utilizar botones de un tamaño considerable puede facilitar la usabilidad, también la posibilidad de reproducir texto debe estar disponible en todos los elementos que tengan texto.

    - Botones con un tamaño considerable: los iconos son el medio que tendrán los usuarios para interactuar con la aplicación y

deben tener un tamaño considerable para que la interacción sea lo más simple posible.

- Interfaz limpia y sencilla: será necesario para que la información pueda leerse lo más rápidamente posible. La sencillez debe ser lo más importante para facilitar la usabilidad.

- El usuario no ha de introducir nada de texto solo debe utilizar un dedo para tocar los iconos.

- Destacar la usabilidad de la aplicación y optimización al máximo de los recursos hardware y software disponibles.

- Pantalla apaisada para mostrar toda la información: la aplicación de R.A. podrá mostrase solo apaisada para representar con máximo detalle toda la información. El Framework por defecto trabaja con la imagen apaisada y por lo tanto el desarrollo se realizará siguiendo esa filosofía.

Además de los requisitos anteriores, en el momento que aparezca el detalle de un punto en la pantalla, también deben considerarse los siguientes requerimientos:

1. **Text-to-Speech:** será un botón para reproducir el texto que aparezca por pantalla. Facilitará la lectura de la información tanto título, descripciones, distancia,…

2. **Mostrar imágenes extra:** los puntos podrán tener imágenes relacionadas, por ello aparecerá la opción de mostrarlas todas por pantalla.

3. **Mostrar comentarios:** ocurre lo mismo que con las imágenes.

4. **Los iconos más grandes:** al mostrar los detalles los iconos deben ser más grandes.

5. **Distancia máxima 2km:** la distancia máxima debe ser 2km para no sobrecargar la información en la pantalla.

6. **Título y descripción:** como detalles principales de cada punto deberán aparecer por pantalla el título y la descripción, además de iconos de accesibilidad que describan gráficamente los accesos que pueda tener.

La siguiente figura, Figura 23: Mockup con los detalles informativos destacando los puntos que deben cumplirse, representa la distribución y el diseño realizado para que la aplicación de R.A. cumpla con los elementos anteriores.

Figura 23: Mockup con los detalles informativos destacando los puntos que deben cumplirse

4.4 Especificación de requisitos (Casos de uso)

El diagrama de casos de uso representa la secuencia de interacciones que se desarrollarán entre un sistema de desarrollo y los actores (clientes) en respuesta de eventos lanzados por el actor sobre el propio sistema. Este tipo de diagramas de casos de uso permiten describir el funcionamiento que el sistema realizará en cada evento que reciba ya sea de los actores o del propio sistema como respuesta de operaciones previas.

Los elementos que forman el diagrama son los siguientes:

- Actor o Actores:

- Un actor es un rol que un usuario desempeña con respecto al sistema. El uso de **rol** en la definición determina que un actor no debe representar a una persona o ente físico en particular, sino que simula la labor que se realiza con el sistema.

Figura 24: Actor caso de uso

- Caso de uso:
  - Representa una operación que es lanzada o realizada tras recibir una orden de algún agente externo. Las peticiones pueden recibirse de un Actor o desde otro caso de uso que ha terminado de realizar su labor.

Figura 25: Caso de uso

- Relaciones:
  - Existen tres diferentes relaciones para según qué operación se realice en el caso de uso o que funcionalidad se le quiera dar. La **Asociación** indica la invocación del caso de uso, puede venir de un actor o de un caso de uso. La **Dependencia o Instanciación** denota una relación entre clases, una clase se instancia de otra. Por último la **Generalización** tiene doble significado, tanto *Uso* como *Herencia.*

El uso de esta técnica de representación es muy utilizado en sistemas interactivos, como puede ser la aplicación de R. A., ya que expresa la intención que tiene el *Actor* al lanzar eventos contra el sistema. Además permite al analista representar las necesidades que puede tener el cliente ofreciendo una imagen del sistema sin tener que mostrar ningún tecnicismo que pueda no ser comprendido. En definitiva, el analista puede centrarse en las tareas que realizará el cliente, describiendo los casos de uso principales del sistema.

A continuación se han detallado dos modos de representación para un caso de uso, el resto de casos de uso están descritos en el apéndice A: Especificación de Requisitos. El primer modo utiliza la iconografía para detallar las operaciones que el sistema puede realizar al recibir mensajes, eventos o interacciones tanto de *actores* como de otros casos de uso (modo gráfico). El segundo modo describe las diferentes interacciones de un modo más detallado, definiendo los flujos de entrada y salida entre otros detalles (modo texto).

4.4.1 Modo gráfico

Las siguientes figuras, Figura 26: Casos de uso lanzar R. A. y Figura 27: Casos de uso información punto, utilizan la representación gráfica para describir los pasos que se realizan para realizar los procesos del sistema. La primera figura representa el proceso realizado por el usuario para utilizar la R. A., desde que la activa hasta que obtiene la información en la pantalla. La segunda figura representa los pasos que se realizan para consultar la información de un punto seleccionado en la pantalla.

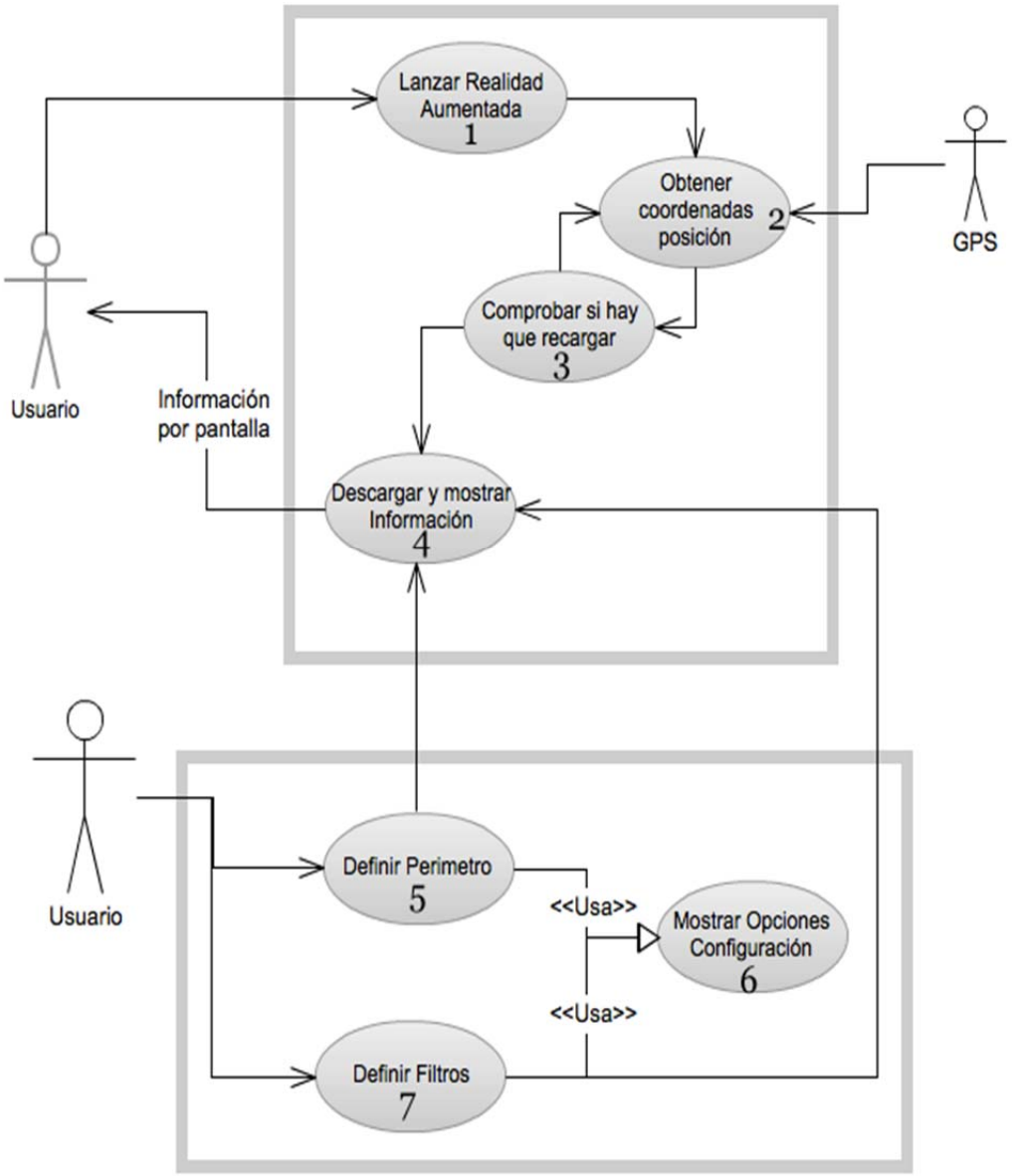

Figura 26: Casos de uso lanzar R. A.

- Mostrar vista R.A.: el primer bloque (nodos 1, 2, 3 y 4) describe el proceso que se realizará para mostrar el contenido de R.A. Al lanzarse activará una serie de pasos consecutivos, el primer paso intenta la localización del usuario (2), comprobando si es válida (3), y en caso de serlo descargar para pintar de nuevo por pantalla (4).

- Cambiar configuración R.A.: el resto de nodos definen la siguiente posibilidad del usuario para interactuar con la R.A. En este caso permitirá cambiar parámetros de la configuración y filtrado de puntos.

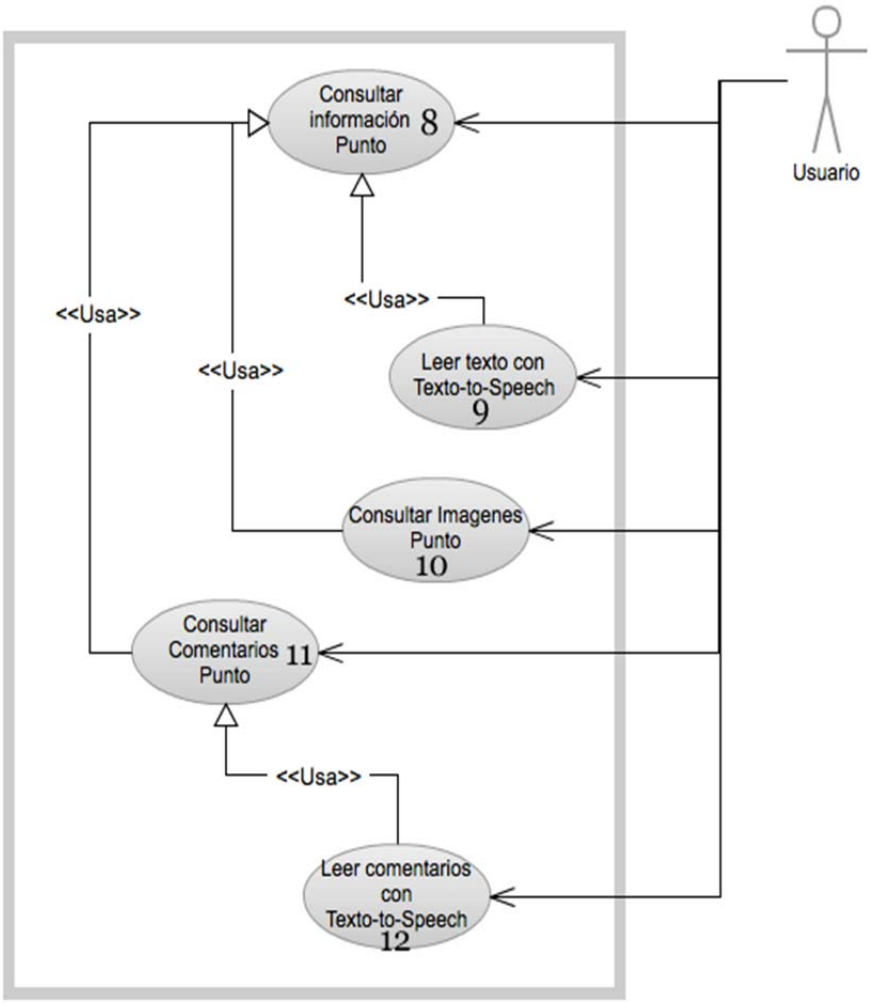

Figura 27: Casos de uso información punto

La Figura 27: Casos de uso define las diferentes opciones de interacción entre usuario y máquina para que aparezcan las descripciones de los puntos.

- Visualizar detalles de elemento en R.A.: el nodo 8 realizará el proceso de mostrar la información de un punto al ser llamado por el usuario.

- Text-to-Speech de la información de un elemento en R.A.: el detalle en pantalla permitirá al usuario activar la reproducción de texto.

- Consultar imágenes asociadas al elemento: otras de las acciones que estarán disponibles será mostrar la galería con imágenes relacionadas al punto.

- Consultar comentarios asociados al elemento: ocurre lo mismo que con las imágenes pero para los comentarios, con la opción de poder reproducirlos.

- Text-to-Speech de los comentarios: podrán leerse los comentarios. Para llegar a este punto primero deberán cargarse los detalles, luego los comentarios y como último paso el usuario podrá reproducirlos.

4.4.2 Modo texto

El caso de uso en modo texto posibilita una definición detalla de cada una de las operaciones realizadas por el sistema de R. A. En este apartado solo han sido expuestos unos ejemplos para mostrar su contenido, el contenido completo se encuentra en el Apéndice A: Especificación de requisitos: Casos de uso, donde se pueden seguir uno por uno todos los pasos realizados para que el usuario pueda obtener la información.

Módulo	Realidad Aumentada	**Componente**	Personalización
ID	1	**Nombre**	Lanzar Vista R.A.
Descripción	El usuario selecciona la vista de R.A. del menú principal de la aplicación.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Secundario		
Datos de entrada	Ninguno		
Precondiciones	El usuario solicita cambiar la vista a R.A.		

Flujo Normal	1. El usuario se encuentra en el menú principal de la aplicación 2. El usuario selecciona R.A. 3. La aplicación cambia la vista a R.A.
Flujo Alternativo	No hay
Post-condiciones	Se muestra la vista de R.A. con las opciones (elementos) por defecto.
Excepciones	No

Módulo	Realidad Aumentada	**Componente**	Personalización
ID	5	**Nombre**	Cambiar opciones R.A.
Descripción	El usuario quiere modificar las opciones de filtrado de elementos a mostrar en la vista de R.A., por defecto la aplicación muestra los elementos en un radio de 2 km alrededor del con esta opción se puede modificar este filtrado.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Secundario		
Datos de entrada	Ninguno		
Precondiciones	El usuario solicita cambiar las opciones de filtrado de la vista de R.A.		
Flujo Normal	1. El usuario se encuentra en la vista de R.A. 2. El usuario selecciona la opción “Configuración” 3. Selecciona la opción de definir perímetro. 4. El usuario selecciona el radio en el que desea visualizar los elementos 5. La aplicación muestra los elementos seleccionados en la vista de R.A.		

Flujo Alternativo	Si los datos recibidos están corruptos o no se reciben, mostrar mensaje al usuario y no aplicar cambios al mapa.
Post-condiciones	La vista de R.A. muestra los iconos de los respectivos elementos para el nuevo margen de filtrado.
Excepciones	No

5 Arquitectura de la aplicación

La aplicación móvil Vadeo es una parte de la arquitectura del sistema Hécate. Este sistema está compuesto, como se aprecia en la Figura 28: Arquitectura Hécate, por un conjunto de subsistemas que se encargan de dividir las funcionalidades del sistema completo. Es importante antes de empezar a desglosar todo lo relacionado con la R.A. hacer una mención de estos subsistemas implicados:

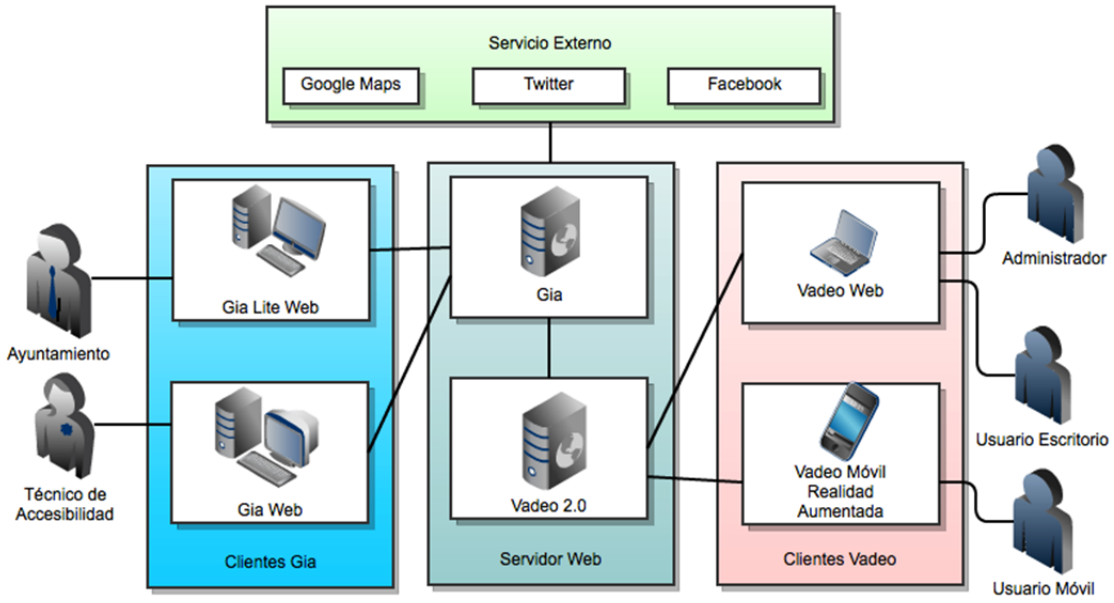

Figura 28: Arquitectura Hécate

- Servidor web: encargado de alojar todos los datos de accesibilidad, tanto de GIA como de Vadeo. Ofrece el contenido mediante web a sus clientes estáticos. En el caso del cliente móvil Vadeo, existe una API de acceso vía web.
- Clientes GIA: son clientes web mediantes los que acceden el personal técnico, ayuntamiento y técnicos de accesibilidad.
- Clientes Vadeo: son los clientes web, cliente móvil y clientes de RA.
- Servicios de terceros: engloba servicios de mapas y redes sociales.

Figura 29: logo Android

Dentro de los clientes disponible en el proyecto Hécate se encuentra la aplicación para dispositivos Android llamada Vadeo. Esta aplicación se puede dividir en dos partes o módulos. El módulo principal de la aplicación es el encargado de realizar el registro del usuario y las correspondientes operaciones para trazar rutas accesibles y generar y consultar información sobre la accesibilidad del entorno. Estas rutas incluyen información sobre puntos de interés y obstáculos. El segundo módulo y no menos importante, engloba toda la funcionalidad de R.A. El siguiente apartado desglosa toda la arquitectura empleada para que la utilización de R.A. sea posible.

De forma general, una vez arrancada la R.A. el cliente realiza una consulta al servidor y éste le devuelva toda la información sobre la accesibilidad de su entorno, según su posición. Los elementos involucrados en este proceso son:

- Cliente Android: arranca la aplicación Vadeo Móvil y carga la opción de R.A que realiza peticiones al servidor.
- Servidor Web: es la pieza principal del sistema. Es el encargado de almacenar toda la información referente a usuarios (no intervienen en la R.A.), obstáculos, puntos de interés, fotografías y comentarios. Además de filtrar las solicitudes para buscar aquellos puntos que coincidan con las peticiones de los clientes.

La Figura 30: Arquitectura de la aplicación, sirve como resumen para identificar todas las partes que van a ser descritas en las siguientes secciones. Se puede observar que Android es el núcleo del cliente por el que se interconecta tanto el Hardware como la aplicación Vadeo Móvil, a través de la máquina virtual (MV) Dalvik. La librería *OpenGL* es la única que conecta directamente con el sistema operativo por el uso del NDK para acelerar la representación gráfica. El resto de componentes deben comunicarse utilizando la aplicación Vadeo o MV Dalvik para interactuar con el S.O. y el Hardware que controla los sensores. Todo el conjunto del cliente está conectado por medio de internet con el servidor web de Hécate para obtener la información a representar. En ese punto aparece el uso también del *Framework LazyLoader* para reducir las búsquedas al servidor almacenando las imágenes en la tarjeta de memoria. El otro *Framework* es *ActionBarSherlock* que se utiliza para mejorar la interfaz gráfica.

Figura 30: Arquitectura de la aplicación

5.1 Cliente Android

La aplicación Vadeo Móvil del proyecto Hécate es el cliente encargado de representar la información almacenada en el servidor Hécate en un dispositivo Android. Una parte importante de esta aplicación es el módulo de R.A., desarrollado como una aplicación independiente, eliminando así toda dependencia entre ambas y siendo el módulo principal (Vadeo Móvil) una pasarela obligatoria para poder arrancar la R.A en este caso.

5.1.1 Ciclo de vida de la aplicación

Las aplicaciones diseñadas para dispositivos móviles y en concreto las de Android, tienen un ciclo de vida que precisa su mención pues difiere mucho de los tradicionales desarrollos de aplicaciones de escritorio.

El primer concepto a destacar es el de **Activity**. Cada pantalla se determina como una *Actividad* y tiene una funcionalidad determinada. El proceso habitual es ir cargando nuevas actividades según se consultan las diferentes partes de la aplicación, almacenando las anteriores en la llamada *Pila de ejecución [25].*

- El primer paso a realizar es lanzar la aplicación Vadeo Móvil seleccionando el icono del menú de aplicaciones en el dispositivo.

- El sistema determina cual es la actividad principal consultando el fichero de configuración *AndroidManifest.xml*. Este fichero almacena toda la información de la aplicación y su configuración.

- Toda clase que represente algo que el usuario quiere hacer debe interpretarse como una *Activity* y por lo tanto debe heredar de ella: Public *Class MainActivity extends Activity {}.* Al lanzar la *Actividad Principal* se van ejecutando una secuencia de métodos escalonados (Figura 24).

- El módulo de R.A. es llamado como cualquier otra actividad, siendo la clase *ShowMap.class* la Actividad previa que permitirá llamar a la R.A.

```
Intent arIntent = new Intent(ShowMap.this, ARView.class);
          startActivity(arIntent);
```

- Una vez en marcha el proceso de cargar la *Actividad* se inicia una secuencia de llamadas a métodos. Entre las más importantes se encuentra *onCreate* que es donde se definen todos los elementos que aparecerán en la pantalla. Estos pueden definirse por código en este método o utilizando ficheros *.xml* donde se describe el aspecto. Lo más

común es utilizar una combinación de ambas para dar el aspecto deseado.

```
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

- Otros métodos interesantes para controlar el ciclo de vida serían *onPause* para detener servicios o almacenar estados, *onStop* para detener todo antes de que se cierre la aplicación.

- Una vez terminado el uso de la actividad se vuelve a la anterior con el botón de *Return* o en su defecto con un botón de volver implantado en la aplicación (últimas versiones de Android). Este proceso iniciado pasará a *Paused,* luego a *Stopped* y por último a *Destroyed*, lanzando la *Activity* que se encuentre en la *Pila de ejecución.*

En la siguiente figura se puede observar el ciclo de vida de una aplicación en Android.

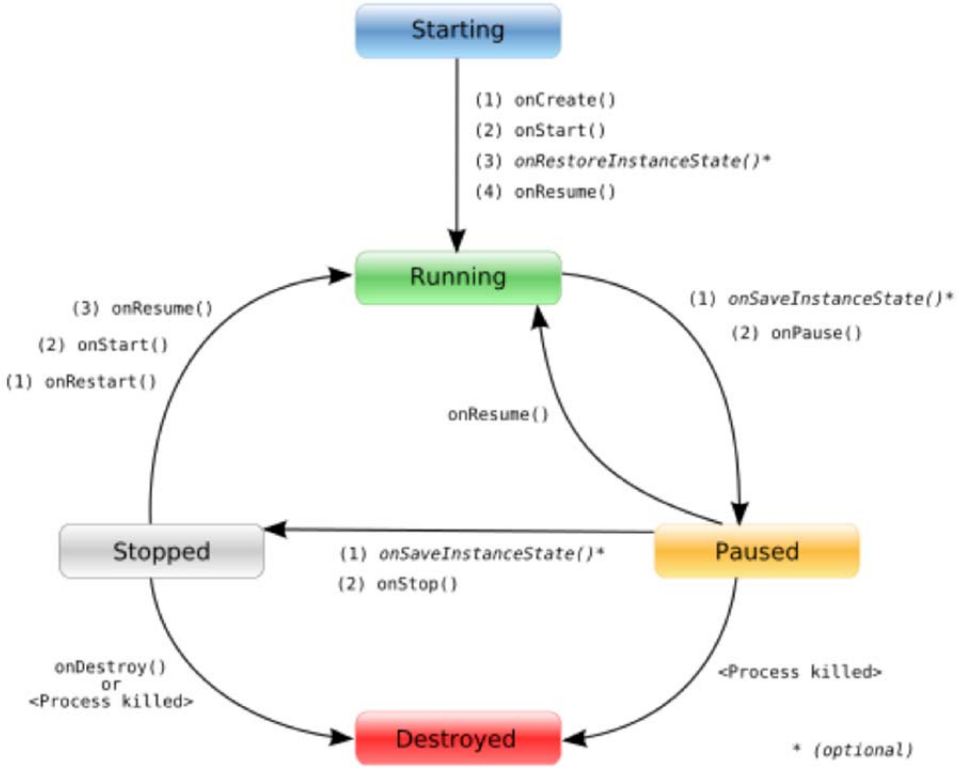

Figura 31: Ciclo de vida Android

En el *Apéndice B (10)* se detalla todos los estados y los métodos del ciclo de vida.

5.1.2 Manejo de la memoria

La nueva generación de dispositivos móviles está alcanzando unas prestaciones inimaginables unos años atrás. Aun así, el uso de los recursos del sistema o de la memoria del dispositivo siempre ha de ser controlado ya que siguen siendo limitados. Además, la gama de terminales que utilizan Android es muy amplia y va desde Hardware con pocos recursos hasta terminales con las últimas prestaciones incorporadas.

La máquina virtual de Android incorpora la figura del *Recolector de basura (Garbage colector, GC)*. Este sistema automatiza la gestión de memoria, reservando cuando es necesario, eliminando todo aquello que ya no es utilizado y compactando cuando es mejor agruparla.

En la mayoría de aplicaciones no es necesario controlar en exceso la gestión de memoria, siendo el *GC* el encargado en lugar del desarrollador. El módulo de R.A. ha sido desarrollado llevando a cabo un control del sistema para impedir el uso de recursos que también afecta en el consumo de la batería, elemento que limita la vida del terminal.

Esta metodología de trabajo incorpora partes buenas y malas, ya que la automatización puede llevar a desbordamientos de memoria. En el caso de la R.A. y el correspondiente uso de imágenes, implica que puede llegar a tener un elevado consumo dependiendo de la densidad de puntos encontrados dentro del perímetro.

Para impedir sobrecargas se ha limitado el perímetro a no más de dos km y un límite de cincuenta para los nodos representados por pantalla.

```
if (res_list.size() > 50)
```

En cada una de las peticiones recibidas del servidor se realiza un proceso de Mapping transformando toda la información descargada en listas de nodos, objetos con coordenadas a representar, etc. Entre ese proceso se encuentra la creación de listas ordenadas según la proximidad de las coordenadas con respecto del usuario. Estas listas serán utilizadas en todo momento para mostrar los nodos, son consultadas continuamente para determinar qué posición debe ocupar cada punto en la pantalla y lo más importante determinar si aparece en el rango de visión.

```
try {
        refreshed = false;
        ARGeoNode.clearClicked(getResourcesList());
        getLayers().cleanResouceLayer();
        setResourcesList(null);
       ArrayList<ARGeoNode>  res_list = ARUtils.cleanNoLocation(this, getLayers(),
getManagerHecateGeoSocial().getNodes());
        if (res_list == null || res_list.size()==0) {
        Toast.makeText(getBaseContext(),getString(R.string.error_no_avaliables),
Toast.LENGTH_LONG).show();
```

```
            Log.e("ARView", getString(R.string.error_no_resources_avaliables));
            return;
           }
           if (res_list.size() > 50) {
            ArrayList<ARGeoNode> list = (ArrayList<ARGeoNode>) res_list.clone();
            res_list.clear();
            for (int i = 0; i < 50; i++)
              res_list.add(list.get(i));
              Toast.makeText(getBaseContext(), getString(R.string.error_too_much_obj),
Toast.LENGTH_LONG).show();
              Log.e(ARView.class.getName(),getString(R.string.error_too_much_objects));
           }
           ARGeoNodeAzimuthComparator comparator = new
ARGeoNodeAzimuthComparator();
           Collections.sort(res_list, comparator);
           setResourcesList(res_list);
           mRadar.setResourcesList(res_list);
           ARGeoNode.setResourcesList(res_list);
           refreshed = true;
           if (altitude_status.equals(AltitudeManager.ALL_HEIGHTS))
                  actionRequestHeight();
} catch (Exception e) {
            Toast.makeText( getBaseContext(),getString(R.string.error_loading)
            + e.getMessage(), Toast.LENGTH_LONG).show();
            Log.e("ARView", "", e);
}
```

Además, la carga de imágenes se realiza de manera perezosa, solo son cargados los iconos de los puntos tanto para ser mostrados por pantalla como para la muestra de la información detallada. El resto de imágenes que puedan mostrarse en una galería única de un punto, son cargadas en el momento.

```
private ImageLoader imgLoader ;
public ARStaticSummaryPicturesBox(Activity mActivity, RelativeLayout container){
```

```
        super(container);
        this.mActivity = mActivity;
        imgLoader = es.tsb.hecate.lazylist.LazyLoader.getImageLoader(this.mActivity);
        }
        …
ImageView img = (ImageView) view.findViewById(R.id.ar_image_node);
imgLoader.DisplayImage(ServiceInterface.PDIPhotos +listUrls.get(position)[1],
mActivity, img);
int auxHg = img.getHeight();
int auxWg = img.getWidth();
img.setPadding(auxWg*2/4, auxHg*2/4, auxWg*2/4, auxHg*2/4);
        …
```

Como se aprecia en el código, el *imgLoader* es el encargado de realizar la carga perezosa utilizando la librería *LazyLoader* detallada en el punto Tarjeta de memoria y LazyLoader.

5.1.3 GPS y Localización

Éste es uno de los puntos más importantes en el proceso de R.A. La localización va a permitir al usuario poder consultar la información que tiene a su alrededor, cuanto más exacta sea la coordenada mejor va a ser la información obtenida.

En la actualidad y como se detalló en el punto 2.2, los terminales incorporan el componente GPS por defecto y es más, también utilizan la información de la señal de comunicación para triangular una posición relativa que sirve de orientación en los procesos de localización. Esta señal es captada de las torres o puntos Wifi donde se conecta el dispositivo. Para ello deben tener agregada la información de su localización o coordenadas. Esta localización permite hacer una estimación aproximada para determinar la ubicación donde puede encontrarse el terminal. El proceso aconsejado es realizar una primera y rápida estimación utilizando la red mientras el GPS se activa, ya que puede tardar un tiempo en realizar la primera localización.

Con toda esta información disponible, la localización del usuario no debe albergar muchos problemas, salvo que por su localización sea difícil la triangulación

con las señales de los satélites para determinar la ubicación. En ese caso el uso de los sistemas auxiliares puede ser de gran ayuda.

El proceso de obtención de las coordenadas ha sido declarado como un servicio que estará disponible durante el tiempo que esté activa la R.A. Como requisito previo para acceder a la R.A. se consultará si el GPS está activo, en caso contrario un aviso será lanzado, es el único requisito establecido. Los servicios tienen una semejanza a los **Activity** ya que utilizan métodos como *onCreate* u *onDestroy* aunque se ejecutan siempre en modo *background*. Continuando con el servicio creado, éste se ubica en la clase *LocationService.class* que hereda de la clase **Services**.

```
public class LocationService extends Service
        ....
@Override    public void onCreate() {
            startService();
            Toast.makeText(this, R.string.id_locations_service+" has been Created",
        Toast.LENGTH_SHORT).show();
  }
@Override    public void onDestroy() {
           hecateLocationContinued.destroyListeners();
           // Tell the user we stopped.
            Toast.makeText(this, R.string.id_locations_service+" has been destroyed",
        Toast.LENGTH_SHORT).show();
  }

private void startService() {
        loadConfig();
        myLibreGeoLocationResult = new LibreGeoLocationResult();
        hecateLocationContinued =  new MyLocation(0, 0, MyLocation.CONTINUED);
        hecateLocationContinued.getLocation(this, myLibreGeoLocationResult);
        if(auxILocationServiceListener!=null)
                myLibreGeoLocationResult.registerLocationListener(auxILocationService
Listener);
        auxILocationServiceListener=null;
        Log.e("LGS-Service:", "Service with MyLocation is enabled");
```

```
}
```

Al arrancar el servicio se lanza el método *startService* que es el encargado de cargar la configuración del GPS. En ella se detallan parámetros como la distancia mínima entre cada consulta, la periodicidad, las unidades de medida, etc. Lo importante viene en el siguiente paso. La clase **LibreGeoLocationResult** es la que recibirá las coordenadas de respuesta y **MyLocation** es la clase encargada de lanzar el proceso de obtención. El funcionamiento se resume en dos pasos simples, comprueba si el GPS está activo, comprueba si la obtención por la red telefónica está activa y lanza un TimerTask (proceso parecido a un hilo que se ejecuta durante un determinado tiempo) con un retraso de 20 segundos, tiempo más que suficiente para que el GPS obtenga una buena posición. Pasado el tiempo se leen los datos del GPS y de la red, se comparan para determinar cuál es más determinante y se envían a la instancia de clase **LibreGeoLocationResult** antes creada.

```
public class MyLocation {
   Timer timer1;
   LocationManager lm;
   LocationResult locationResult;
...
public MyLocation(int time, int distance , boolean continued){
        this.minTime = time;
        this.distance = distance;
        this.continued = continued;
}
public boolean getLocation(Context context, LocationResult result)   {
 //I use LocationResult callback class to pass location value from MyLocation to user
code.
  locationResult=result;
   if(lm==null)
    lm = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
//exceptions will be thrown if provider is not permitted.
    try{gps_enabled=lm.isProviderEnabled(LocationManager.GPS_PROVIDER);}
catch(Exception ex){}
```

```
try{network_enabled=lm.isProviderEnabled(LocationManager.NETWORK_PROVIDER);}
catch(Exception ex){}
  //don't start listeners if no provider is enable
if(!gps_enabled && !network_enabled)     return false;
if(gps_enabled)
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,  this.minTime,
this.distance, locationListenerGps);
if(network_enabled)
        lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
this.minTime, this.distance , locationListenerNetwork);
      timer1=new Timer();
      timer1.schedule(new GetLastLocation(), 20000);
return true;    }
class GetLastLocation extends TimerTask {
      @Override
      public void run() {
         Location net_loc=null, gps_loc=null;
         if(gps_enabled)
            gps_loc=lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
         if(network_enabled)
            net_loc=lm.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
         //if there are both values use the best one
         if(gps_loc!=null && net_loc!=null){
          locationResult.gotLocation(  UtilGPS.isBetterLocation(gps_loc, net_loc) ?
net_loc: gps_loc) ;
            return;
         }
         if(gps_loc!=null){       locationResult.gotLocation(gps_loc);
            return;
         }
         if(net_loc!=null){       locationResult.gotLocation(net_loc);
            return;
         }
         locationResult.gotLocation(null);
```

```
    }
  }
public static abstract class LocationResult{
    public abstract void gotLocation(Location location);
  }
}
```

El constructor de la clase **MyLocation** recibe como parámetros la configuración establecida para el GPS o la red. Esa configuración incluye la distancia mínima de consulta, el mínimo tiempo entre consultas y si solo se realiza una consulta, deteniendo el servicio al recibir la posición. Esta última opción no se contempla ya que debe estar consultando periódicamente. Con estos parámetros se define la lectura de la localización, intentando minimizarla al máximo debido al coste energético que puede tener.

Un detalle importante que hay que destacar acerca del GPS es que tenerlo encendido no consume batería. Su consumo viene dado por las consultas que realice, ese es el punto que se ha de perfilar y con los parámetros antes detallados se intenta reducir al máximo las consultas.

El método que pone en marcha las consultas al servidor es **getLocation**. Este método recibe el contexto (*información del sistema actual)* y una instancia de **LocationResult**. La clase **LocationResult** tiene un papel determinante, todas las clases que quieran utilizar este proceso deben heredar de ella e implementar el método getLocation*()* donde recibirán la nueva coordenada. Un ejemplo de esta implementación se encuentra en el siguiente código:

```
public class LibreGeoLocationResult extends LocationResult {
private List<ILocationServiceListener> arrayLocationListener;
private Location currentLocation;
public LibreGeoLocationResult() {
        arrayLocationListener = new ArrayList<ILocationServiceListener>();
}
@Override public void gotLocation(Location location) {
```

```
        // Comprobamos que sea una mejor posición.
        if (currentLocation != null
                && UtilGPS.isBetterLocation(currentLocation, location)) {
                this.currentLocation = location;
        } else {  this.currentLocation = location; }
        for (ILocationServiceListener iterable_element : arrayLocationListener)
                iterable_element.updateCurrentLocation(this.currentLocation);
        }
        public Location getCurrentLocation() {
                return this.currentLocation;
        }
        public void unRegisterLocationListener(Integer pos) {
                arrayLocationListener.remove(pos - 1);
        }
        public Integer registerLocationListener(ILocationServiceListener l) {
                arrayLocationListener.add(l);
                return arrayLocationListener.size();
        }
}
```

5.1.4 Conexión a internet

Internet es un medio imprescindible en este proyecto de R.A. Toda la información que se representa por pantalla es obtenida en tiempo real por la aplicación. La comunicación con el servidor no debe ser constante, es más, para el caso de la R.A. puede haber periodos de latencia elevada entre peticiones dado que un perímetro define el área de refresco. Pero llegado el momento de la petición es un momento crítico que debe realizarse con total fiabilidad. Es importante destacar que en ningún momento se ha tenido en cuenta el tipo de conexión 3G o Wifi, la razón es que este proceso es totalmente transparente para los desarrolladores, es el propio sistema el encargado de establecer la conexión en función de la configuración establecida por el usuario.

La tecnología Android dispone de unas APIs desarrolladas por **Apache Software Fundation (ASF)** para realizar todo tipo de consultas. Estas clases resumen en varios pasos el protocolo de comunicación entre el cliente y el servidor:

- DefaultHttpClient: es una implementación de la clase *HttpClient* que pre-configura los escenarios más comunes de comunicación.
- HttpGet: el protocolo GET (*RFC2616)* significa que cualquier identificador será enviado por la URI. La instancia debe recibir el tipo de cabecera de comunicación y la URI de destino.
- HttpPost: el protocolo POST *(RFC2616)* es utilizado para que el servidor acepte una entidad cerrada (envía los parámetros en la cabecera y no en la URI) en una petición.
- HttpResponse: después de recibir la petición, el servidor responde con un mensaje *http* que esta clase se encarga de almacenar.
- HttpEntity: las entidades permiten convertir el mensaje recibido en un elemento consistente en la memoria del sistema. Si no fuera así el mensaje está contenido en la comunicación y en el momento que se cerrase el contenido desaparecería. Es resumen, una entidad es una instancia del resultado de la comunicación.

La siguiente figura, **¡Error! No se encuentra el origen de la referencia.**, define las diferentes posibilidades que puede tener un cliente móvil para comunicarse con el servidor utilizando tanto la red móvil como las conexiones inalámbricas (*Wifi)*.

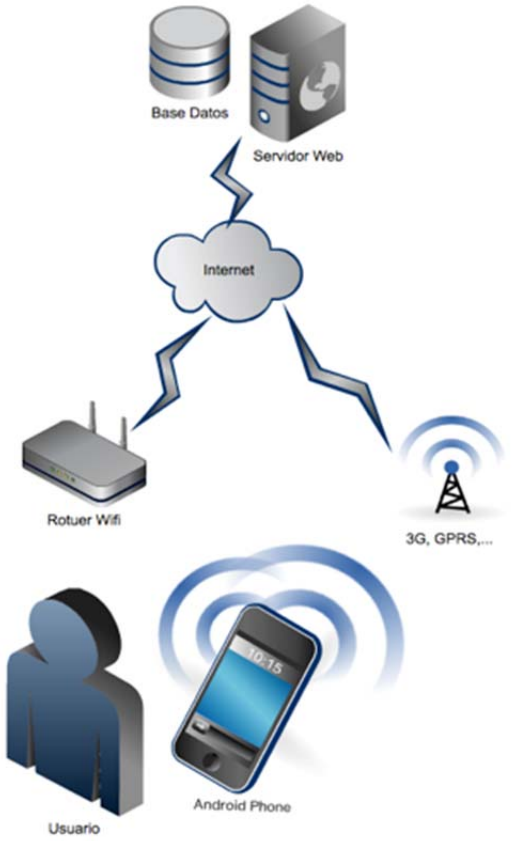

Figura 32: Arquitectura de comunicación entre usuario y el servidor

El siguiente código es utilizado para realizar consultas al servidor utilizando el protocolo de comunicación GET y gestionando tanto la respuesta del servidor como los posibles fallos en la comunicación.

```
private InputStream retrieveStream(String url, String parametros)
        throws IOException {
 DefaultHttpClient client = new DefaultHttpClient();
HttpGet getRequest = new HttpGet(url + "?" + parametros);
getRequest.setHeader("Content-Type", "text/plain; charset=utf-8");
try {
  HttpResponse getResponse = client.execute(getRequest);
  final int statusCode = getResponse.getStatusLine().getStatusCode();
        if (statusCode != HttpStatus.SC_OK) {
                Log.w("HecateGeoSocial.class", "Error " + statusCode
                + " for URL " + url);
                return null;
```

```
            }
  HttpEntity getResponseEntity = getResponse.getEntity();
  return getResponseEntity.getContent();
} catch (IOException e) {
            getRequest.abort();
            Log.w("UtilConectionServer.class", "Error for URL " + url, e);
            throw e;
}
}
```

Las respuestas del servidor adquieren complejidad en el momento de contener toda la información que debe ser enviada utilizando el estándar **JSON.**

El lenguaje **XML** es el antecesor de este estándar. Ambos están destinados a ser utilizados en el ámbito de intercambiar información, pero el uso de *XML* requiere una gran cantidad de meta-información. Toda esta meta-información hace que el texto a simple vista no sea legible, necesitando de analizadores sintácticos tanto en la parte emisora como en la receptora. Esto es un inconveniente que genera un retardo excesivo cuando lo importante es tener clientes ligeros que reciban respuestas inmediatas.

JSON es un formato ligero para el intercambio de datos. Al contrario que *XML*, el código *JSON* es muy compacto, fácil de entender y más fácil de interpretar. *GSON* es una librería de *Apache (ASF)* utilizada en el proyecto, que mejora la usabilidad de esta comunicación. Esta librería utiliza las anotaciones para poder mapear la información recibida en una estructura de objetos. Con esto se consigue que con un par de llamadas a la librería el resultado sea almacenado en los atributos, listas,…

```
public class StatusRequestServer implements Serializable {
            // Clase que recibe la respuesta del servidor y almacena un listado con todos
            // los nodos a representar
  private static final long serialVersionUID = -9121943636715457236L;
            @SerializedName("status")   public String status;
```

```
        @SerializedName("msg")    public String msg;
        @SerializedName("data")   public ArrayList<GeoNode> listNodes;
}

....

        // Realiza la llamada al servidor y mapea el resultado en la clase
        //StatusRequesServer
Reader reader =  new InputStreamReader(
retrieveStream(ServiceInterface.ListPDIRadio, parameters) );
 Gson gson = new Gson();
 StatusRequestServer  statServ =   gson.fromJson(reader,StatusRequestServer.class);
```

Por último, es posible que **el estado de la red** no sea siempre el correcto y, aun teniendo múltiples métodos de comunicación, no exista conexión alguna. La desconexión puede darse y durar un tiempo indeterminado. La aplicación, está preparada para soportar ese cambio en el estado de la red realizando comprobación de la conexión cada cierto tiempo, hasta detectar que la conexión se ha restablecido. Si la comprobación fuese continua incurriría en un consumo de batería elevado y/o incluso saturación de los recursos del terminal.

5.1.5 Cámara y OpenGL

La R.A. consiste en observar el mundo real y añadir información sintética para ampliar el contenido. La cámara es la encargada de recoger la información del mundo real para que sea mostrada en el dispositivo como *background (fondo)* de la aplicación.

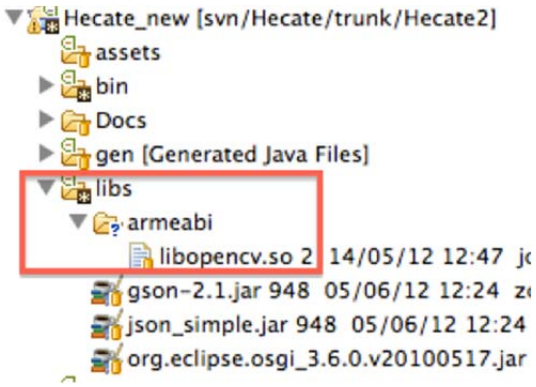

Figura 33: librería OpenGl ES

En la actualidad la gran mayoría de terminales salen al mercado incluyendo cámara, otros sensores o incluso brújula. Todos estos complementos se han convertido en un estándar actual de los terminales. Dependiendo de la gama del

terminal estos puede ser de mayor o menor calidad, pero aun así se garantiza que la aplicación no tendrá impedimentos hardware para ser utilizada.

Android incluye soporte para gráficos (2D y 3D) con la librería OpenGL, concretamente OpenGL ES (versión 2.0). Esta API es un estándar para interactuar con el hardware acelerador de gráficos que se incluye a partir de la versión 8 de Android (API 2.2). Este acelerador gráfico es un pilar fundamental en la manipulación de la información que incluye el Framework de R.A.

La librería está desarrollada completamente en lenguajes nativos como C o C++. El NDK permite utilizar este código sin tener que pasar por la máquina virtual Dalvik aumentando la velocidad de comunicación con el Hardware y por lo tanto incrementando las prestaciones.

```
public class OpenCV {
        static {
            System.loadLibrary("opencv");
          }
        }
public class CamPreview extends SurfaceView implements SurfaceHolder.Callback {
        private OpenCV ocv;
        public CamPreview(Context context) {
            super(context);
            ocv = new OpenCV();
            sem = new Semaphore(1);
        }
```

En la siguiente llamada se activa la cámara.

```
 public void surfaceCreated(SurfaceHolder holder) {
 try {
        mCamera = Camera.open();
```

```
                mCamera.setPreviewDisplay(holder);
            } catch (IOException e) {
                    mCamera.release();
                Log.e("CamPreview", e.toString());
                }
              }
            public void surfaceDestroyed(SurfaceHolder holder) {
                mCamera.stopPreview();
                mCamera.release();
                mCamera = null;
              }
public void setPreviewCallback(PreviewCallback pc){
            mCamera.setPreviewCallback(pc);
 }
public void takePicture(PictureCallback pic){
            mCamera.takePicture(null, null, pic);
 }
public void resumePreview(){
            mCamera.startPreview();
   }
public void pausePreview(){
            mCamera.stopPreview();
   }
}
```

La clase **CamPreview** es la encargada de configurar, activar e interactuar con la cámara. Los estados de *Pause* y *Resume* de *ArView* se encargan de detenerla para evitar una utilización inadecuada. Un detalle importante es que para poder utilizarla en la aplicación es imprescindible registrar el permiso.

```
<uses-permission android:name="android.permission.CAMERA" />
protected void onResume() {
```

```
        super.onResume();
        getLayers().addBaseLayer(mPreview);
        showMenu = true;
        . . .
protected void onPause() {
        super.onPause();
        getLayers().removeBaseElement(mPreview);
        . . .
```

5.1.6 Text-to-speech

El colectivo de personas a las que va dirigida la aplicación, pueden tener, entre otras, limitaciones que les impiden leer con claridad el texto contenido en la pantalla de su dispositivo. Para el usuario corriente una pantalla es más que suficiente para albergar toda la información, pero puede darse el caso que no ofrezca facilidades de lectura en otras circunstancias. A partir de la versión 1.6 de Android se introdujo un sintetizador de voz o también conocido como *Text-to-Speech (TTS).*

El TTS soporta un gran número de lenguajes: inglés, francés, alemán, italiano y español. Como curiosidad, dependiendo de la parte del Atlántico en que se encuentre puede utilizar acento británico o americano. Es una gran herramienta que se ha utilizado en el proyecto para reproducir todo el texto que se encuentre en la descripción de cada punto seleccionado y también en aquellos comentarios que pueda contener.

En la siguiente clase se aprecia como es necesario implementar la clase *TextToSpeech.OnInitListener* y su método *onInit:*

```
public class ARStaticSummaryCommentBox extends ARSummaryBox  implements
TextToSpeech.OnInitListener{
        private TextToSpeech tts;
        ...
@Override public void onInit(int status) {
   if (status == TextToSpeech.SUCCESS) {
```

```
        Locale loc = new Locale ("spa", "ESP");
        int result = tts.setLanguage(loc);
        if (result == TextToSpeech.LANG_MISSING_DATA  || result ==
TextToSpeech.LANG_NOT_SUPPORTED) {
        Toast.makeText(mActivity, mActivity.getString(R.string.error_text_to_speech),
Toast.LENGTH_SHORT).show();
        Log.e("TTS", "This Language is not supported");
    }
 } else {  Log.e("TTS", "Initilization Failed!");     }
            }
. . .
```

El uso es tan simple como invocar:

```
tts.speak("Texto a reproducir", TextToSpeech.QUEUE_FLUSH, null);
```

5.1.7 Tarjeta de memoria y LazyLoader

El consumo de datos es una de las limitaciones que tienen las aplicaciones móviles actualmente. Las tarifas de datos son limitadas con respecto a la cantidad de datos que pueden ser descargados. Para evitar descargar varias veces la misma imagen en diferentes consultas de un mismo punto se ha incluido la librería *LazyLoader* en el proyecto.

El funcionamiento de esta librería está basado en descargar una vez y utilizar tantas como sea necesario. Para ello utiliza la memoria externa o tarjeta de memoria del dispositivo como almacén. Antes de descargar la imagen que debe aparecer, normalmente en un **ImageView** (*componente encargado de mostrar imágenes)* se consulta si esa misma imagen está almacenada en la tarjeta. En caso afirmativo se procede a utilizar la imagen rápidamente, en caso contrario se consulta la dirección de la imagen para descargarla y almacenarla en la tarjeta de memoria. Así, en futuras utilizaciones la imagen estará disponible, ahorrando así tráfico de datos.

Es imprescindible activar el permiso de escritura en el *AndroidManifest.xml*, de lo contrario no se podría almacenar:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Figura 34: Icono sherlockActionBar

5.1.8 SherlockActionBar

Los continuos avances de Android tanto a nivel de equipos como de sistemas operativos parecen enfocarse a realizar toda la operatividad por medio de la pantalla. Lejos están quedando los terminales que incluían botones para mostrar opciones y las opciones están quedando relevadas a aparecer como un menú más en la pantalla.

Para poder tener una experiencia de uso similar tanto en versiones 2.x como las más actuales 4.x, se ha optado para añadir esta librería que permite simular una interfaz gráfica similar a las versiones más modernas pero en dispositivos con versiones anteriores y más limitados.

Figura 35: Aplicación R.A. con detalle de punto

Esta librería es una extensión de una librería que ha creado Google para compatibilizar los diseños y facilitar el uso de barras de diseños a través de las

diferentes APIs. La librería automáticamente usa código nativo del *Action Bar* cuando lo cree oportuno o se ajusta automáticamente a la implementación realizada en los diseños. Esto facilita la usabilidad en las versiones 2.x y superiores.

Figura 36: Salir de la R.A.

En la imagen anterior, Figura 35: Aplicación R.A. con detalle de punto, puede apreciarse en la parte superior izquierda de la pantalla el icono con el nombre de la aplicación y en la parte derecha los botones para abrir la configuración. Una de las funcionalidades que incorpora también es la opción de acceder a la pantalla anterior seleccionando el icono de la APP Figura 36: Salir de la R.A.. [26]

5.1.9 Opciones de búsqueda

El proceso de obtención de la información a representar se basa en realizar peticiones al servidor enviando la localización del dispositivo en cada momento. Para que las búsquedas no sean genéricas y descarguen todo tipo de información, el usuario dispone de opciones para determinar el filtrado que desee realizar.

Figura 37: Opciones de filtrado

Los parámetros de búsqueda permiten, entre otras cosas, elegir los puntos de interés (PDIs), obstáculos, la visualización de componentes como la flecha o incluso definir el radio de búsqueda. La cantidad de tipos de PDIs disponibles alcanza la treintena, por lo tanto puede llegar a mostrar una gran cantidad de información, que sea superior a la necesitada por el usuario. Como muestra la Figura 38: Listado de PDIs, el usuario solo tiene que ir marcando la casilla de aquellos puntos que le interesen y

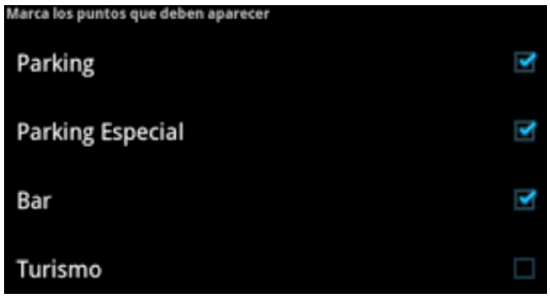

Figura 38: Listado de PDIs

desmarcando los que no sean de su interés en determinado momento. En la Figura 37: Opciones de filtrado, es posible elegir también entre puntos que hayan sido verificados *por técnicos de accesibilidad y/o* ayuntamientos (oficiales), o también puntos que los propios usuarios de la red social hayan añadido por cuenta propia (sociales).

5.1.10 Framework de R.A. LibreGeoSocial

El término de Framework para LibreGeoSocial ha sido acuñado a posteriori, en un principio se trataba de una aplicación concreta que fue liberada como *Open Source* y licencia *Creative Commons*. Una vez separada la parte que envolvía la R.A. de la parte de consulta que tenía la aplicación original, se le dio esta nomenclatura para diferenciar ambas partes.

El proceso de separación de ambas partes no fue un proceso fácil. La R.A. estaba enfocada para utilizarse en esa APP. Elementos como las clases que contienen los puntos a mostrar se han mantenido e incrementado con los nuevos atributos para poder utilizar la R.A. Esto también ha sido un punto a favor para desarrollar todo el contenido de la R. A. como una aplicación independiente de la propia aplicación Vadeo Móvil, además gracias al mapeo utilizando la librería *GSON* de *Apache (ASF)* ha sido sencillo poder ajustar los campos de la base de datos con los atributos de las clases del Framework.

Las características principales de la *Aplicación* han sido descritas en el capítulo 2.1.5 por lo que en este punto del documento se hace referencia a cómo se han desarrollado las partes principales del Framework y cómo se utilizan los elementos descritos en los apartados anteriores.

El siguiente código contiene la clase que mapea los puntos descargados.

```
import com.google.gson.annotations.SerializedName;
public class GeoNode implements Serializable {
    // Serializable UID
    private static final long serialVersionUID = -9121943636715457236L;
    @SerializedName("id")          private Integer mId;
    @SerializedName("titulo")      private String titulo;
```

```
        private Double  mRadius;
        @SerializedName("fecha_alta")      private String  mSince;
        @SerializedName("lat")             private Double mLatitude;
        @SerializedName("lon")             private Double mLongitude;
        private Double mAltitude = AltitudeManager.NO_ALTITUDE_VALUE;
        private String mPosition_since;
        @SerializedName("nickUsuario")     private String mUsername;
        @SerializedName("idUsuario")       private String idUsuario;
        @SerializedName("tipo")            private String type;
        @SerializedName("classPDI")        private Boolean classPdi;
        @SerializedName("classPunto")      private Boolean classPunto;
        @SerializedName("coments")         private List<Integer> coments;
        @SerializedName("fotos")           private List<String[]> fotos ;
        @SerializedName("accesib")         private List<AccessInformation>
accesib;
    . . .
```

Una de las características que utiliza la App LibreGeoSocial *(a partir de ahora acuñada con el término LGS)* son las capas. La APP permitía poder enlazar una capa para mostrar una información en concreto. Uno de los ejemplo son capas de *Youtube, Paranomico, etc*. Seleccionando una capa se descargaba su información.

Toda esta generalización no era necesaria en este proyecto. En lugar de enviar la información en capas se ha utilizado la clase principal encargada de descargar el contenido con la información para almacenar los métodos necesarios de comunicación con el servidor. Se podría considerar como el pilar central de los datos, todo esto está incluido en la clase ***HecateGeoSocial.class.***

```
    public class HecateGeoSocial {
    ...
     static public HecateGeoSocial getInstance() {
                if (singleton == null) {
                        singleton = new HecateGeoSocial();
                }
```

```
            return singleton;
        }
```

La localización se realiza consultado el servició activado al arrancar la R.A.

```
        public Location getLocation() {
            Location myLocation = LocationService.getCurrentLocation();
            return myLocation;
        }
```

El siguiente método es el que realiza las llamadas para obtener los nuevos nodos. Se le envía la posición junto con los parámetros del filtrado para que el servidor devuelva los puntos oportunos. En el siguiente código se puede apreciar el uso de GSON.

```
        public ArrayList<GeoNode> getLayerNodes(Location loc, String parameters)
throws Exception {
          try {
        . . .
            Reader reader  new InputStreamReader(
retrieveStream(ServiceInterface.ListPDIRadio, parameters) );
            Gson gson = new Gson();
            StatusRequestServer   statServ =
gson.fromJson(reader,StatusRequestServer.class);
        return mArrayNodes = statServ.getListNodes();
        } catch (Exception e) {
            Log.e(TAG, e.getMessage());
            throw e;
         }
        }
        . . .
```

Los comentarios también se descargan del mismo modo, pero en este caso bajo demanda del usuario.

```
        public List<Comment> getComments(String strListIdComents) throws Exception
{
```

```
            try {
             String parameters = "ids_comentarios="+strListIdComents;
              Reader reader = new InputStreamReader(
retrieveStream(ServiceInterface.Comments, parameters) );
              Gson gson = new Gson();
              StatusRequestServerComments  statServ =
gson.fromJson(reader,StatusRequestServerComments.class);
            return  statServ.getListComment();
        } catch (Exception e) {
            Log.e(TAG, e.getMessage());
            throw e;
        }
        }
        ...
```

El último de los métodos interesantes de nombrar es el getThreadtoUpdate, encargado de lanzar un nuevo *Hilo de ejecución* para realizar el proceso de obtención de nodos. Es necesario el uso de hilos para no detener la ejecución de la aplicación.

```
    public Thread getThreadtoUpdate(final Handler hNotification, final String[]
categories, final Location loc) {

     Log.e("HecateGeoSocialLayer", "getThreadtoUpdate");
     Thread th = new Thread() {
    public void run() {
            try {
                ArrayList<GeoNode> auxArray = new ArrayList<GeoNode>();

    auxArray.addAll(HecateGeoSocial.getInstance().getLayerNodes(loc));
            if (!auxArray.isEmpty()) {
              GeoNodePositionComparator comparator = new
GeoNodePositionComparator();
             Collections.sort(auxArray, comparator);
            }
```

```
            mArrayNodes = auxArray;
            hNotification.sendEmptyMessage(0);
        } catch (Exception e) {
        Log.e(TAG, e.getMessage());
        } } };
        return th;
        }
        }
```

5.1.11 Los sensores

Figura 39: Icono Pausa

Los sensores son utilizados continuamente para refrescar la información. Como mínimo la brújula marca cada uno de los movimientos que sufre el terminal. La clase **ARCompassManager** es la encargada de controlar las actualizaciones de los sensores y notificarlo para que elementos como la brújula aparezcan refrescados por pantalla. Una de las funcionalidades que se ha añadido es la opción de poder pausar el contenido. Tras realizar múltiples pruebas se determinó que en algunos casos puede ser difícil para el usuario utilizar ambas manos y se optó por añadir un botón de pausa que detuviese toda actualización. Con esa opción los iconos permanecen estáticos en la pantalla y son más fáciles de seleccionar. La pausa se realiza bloqueando los nuevos resultados capturados.

En el siguiente código se refleja lo explicado con anterioridad al definir una clase que implementa la *Interface* **SensorEventListener**. Esta *Interface* permite capturar las actualizaciones producidas en los sensores para realizar cálculos de las nuevas posiciones del dispositivo. Estos cálculos son utilizados a la hora de situar los elementos en la pantalla.

```
public class ARCompassManager implements SensorEventListener{
        OnCompassChangeListener onCompassChangeListener = null;
        private SensorManager sm;
        private static Boolean FLAG_PAUSE = false;
...
```

```
        public ARCompassManager(Context mContext){
                sm = (SensorManager)
mContext.getSystemService(Context.SENSOR_SERVICE);
        }
        private void setSensors(){
          //Setting a sensor listener for accelerometer and magnetic field sensors
          try {
        List<Sensor> sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
                if(sensors.size()>0){
                Sensor sensor = sensors.get(0);
                        sm.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
                        }
                sensors = sm.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
                if(sensors.size()>0){
                Sensor sensor = sensors.get(0);
                        sm.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
            }
        } catch (Exception e) {          Log.e("ARCompassManager", "", e); }    }

         public void setOnCompassChangeListener(OnCompassChangeListener
onCompassChangeListener){
                if (this.onCompassChangeListener == null)
                        setSensors();
                this.onCompassChangeListener = onCompassChangeListener;
            }

            public void unregisterListeners(){
                sm.unregisterListener(this);
                onCompassChangeListener = null;
            }

            private void OnCompassMeasure(){
                if(onCompassChangeListener == null)
```

```
            return;

        float[] values = new float[3];
        float[] inr = new float[9];
        float[] outr = new float[9];
        float[] i = new float[9];

        float[] values_acc = acc_values.clone();
        float[] values_mag = mag_values.clone();

        Boolean auxResuSensor = SensorManager.getRotationMatrix(inr, i,
values_acc, values_mag);
        if(inr!=null &&  !FLAG_PAUSE ){
          SensorManager.remapCoordinateSystem(inr, SensorManager.AXIS_X,
SensorManager.AXIS_Z, outr);

         SensorManager.getOrientation(outr, values);
         values[0] = (float) Math.toDegrees(values[0]);
         values[1] = (float) Math.toDegrees(values[1]);
        values[2] = (float) Math.toDegrees(values[2]);

        // correction of the measures
        //azimuth
        values[0] += correction;
        if(values[0] < 0)
            values[0] += 360;
            //elevation
            values[1] = 90 - values[1];

            // PID controller for azimuth and elevation
            ////////////////////////////////
            values[0] = azimuthPID.getValue(values[0]);
            values[1] = elevationPID.getValue(values[1]);
            onCompassChangeListener.onChange(values);
```

```
        }
    }

public void onSensorChanged(SensorEvent event) {
            switch(event.sensor.getType()){
            case Sensor.TYPE_ACCELEROMETER:
                acc_values = event.values.clone();
                break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                mag_values = event.values.clone();
                OnCompassMeasure();
                break;
            }
}
```

La clase encargada de arrancar y mover toda la actividad de la R.A. es **ARView**. Hereda de **ARActivity** que al mismo tiempo hereda de **SherlockFragmentActivity** para poder utilizar las ventajas del Framework. Cuando arranca se encarga de ir instanciando todos los complementos que participan en el sistema. Para empezar se instancia el *LocationService* para controlar la localización. Después se encarga de ir añadiendo las capas de información que aparecerán por pantalla (*radar, info, etc.)*. Acto seguido carga todas las preferencias, entre otras cosas, y por fin lanza la primera llamada para cargar la información.

```
public class ARView extends ARActivity {
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
   try {
startService(new Intent(this,LocationService.class));
        LocationService.registerInitListener(locationListenerStart);
        pointerObject = this;
        refreshed = false;
```

```
            showMenu = true;

            // Hide the window title and notifications bar.
            requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        //Operaciones con action bar
            ActionBar actBar = getSupportActionBar();
            actBar.setDisplayHomeAsUpEnabled(true);

            // Create our Preview view and set it as the content of our activity.
            mPreview = new CamPreview(this);
            // setting the layers which we use as graphical containers
            getLayers().setBaseLayer();
            getLayers().setResourceLayer();
            getLayers().setInfoLayer();
            getLayers().setExtraLayer();
            getLayers().setExtraLayerPicturesAndComments();

            mFocus = new DrawFocus(this);
            mRadar = new DrawRadar(this);
            mPause = new DrawPauseButton(this);
            getLayers().addInfoElement(mRadar, null);
            getLayers().addInfoElement(mPause, null);
            getLayers().addInfoElement(mFocus, null);

        loadConfig(false);

            ARGeoNode.setRadar(mRadar);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_O
N);
            compassManager = new ARCompassManager(this);
            tagManager = new ARTagManager(this, getLayers(), getLocation(),
        cam_altitude);
```

```
tagManager.setOnLocationChangeListener(onTaggingLocationListener);
tagManager.setOnTaggingFinishedListener(onTaggingFinishedListener);

        // Hecate-clean: Realizar capturas de pantalla
        screenshotManager = new ScreenshotManager(getBaseContext());
        screenshotManager.setCam(mPreview, frameReadyListener);
        showResources();
} catch (Exception e) {  Toast.makeText(
        getBaseContext(),
        "There was an error loading the AR environment elements: "
        + e.getMessage(), Toast.LENGTH_LONG).show();
        Log.e("ARView", "", e);
}
}

protected void onResume() {
        super.onResume();
getLayers().addBaseLayer(mPreview);
        showMenu = true;
        getLayers().cleanResouceLayer();
        getLayers().cleanExtraLayer();
        ARGeoNode.clearClicked(getResourcesList());
        compassManager.setOnCompassChangeListener(compassListener);
        if (!(idGPS < 0))
                idGPS =
LocationService.registerLocationListener(locationListener);
}
```

Este método define la vista que contiene el *XML* con las opciones. Al crearse la actividad lee este método por defecto y carga la configuración que se establezca. Para la R.A. aparece en la parte superior derecha los botones de configuración.

```
@Override  public boolean onCreateOptionsMenu(Menu menu) {
```

```
        MenuInflater inflater = getSupportMenuInflater();
        inflater.inflate(R.menu.menu_navegacion, menu);
         return super.onCreateOptionsMenu(menu);
    }
```

Este es el método que captura los eventos lanzados al marcar los botones de configuración.

```
    @Override public boolean onOptionsItemSelected(MenuItem item) {
    ...
    }
```

Este *Listener* es el que recibe las actualizaciones de las nuevas posiciones del GPS. Comprueba que la posición sea diferente a la anterior y además esté fuera del perímetro anterior.

```
    ILocationServiceListener locationListener = new ILocationServiceListener() {
      public void updateCurrentLocation(Location loc) {
          float[] location = { (float) loc.getLatitude(),  (float) loc.getLongitude(), 0
};
    setLocation(location);
          if (mUserStatus != null)
          mUserStatus.setLocationServiceActive(true);
          if (AltitudeManager.isLocationServiceAltitude()) {
                cam_altitude = (float) loc.getAltitude();
                LocationUtils.setUserHeight(cam_altitude);
          if (mUserStatus != null)
                      mUserStatus.setAltitudeLoaded(true);
                } else if (tagManager.getSavingType() == -1)
                      requestAltitudeInfo();
                if (tagManager != null) {
                      tagManager.setUserLocation(location);
                      tagManager.setCamAltitude(cam_altitude);
                }

          if(isNecessaryReloadSources())
```

```
                    loadResources();
        }
        };
```

En caso de ser correcto realiza una petición a *LoadResources* para que ejecute el *ThreadToUpdate* encargado de hacer una petición al servidor.

```
        private void loadResources() {
                getManagerHecateGeoSocial().getThreadtoUpdate(handlerEnd,
strCategories, LocationService.getCurrentLocation()).start();
        }
```

Es el encargado de recibir la notificación una vez obtenidos los nuevos puntos, luego llama a *showResources* para procesarlos.

```
        private Handler handlerEnd = new Handler() {
                public void handleMessage(Message msg) {
                        if (ARView.this.isFinishing())
                                return;
                        showResources();
                } };


        private int idLocationListenerStart = -1;
        ILocationServiceListener locationListenerStart = new ILocationServiceListener() {
        private AlertDialog  alertDialog;
                public void updateCurrentLocation(final Location loc) {
                 ARView.this.runOnUiThread(new Runnable(){
                   public void run(){
                        pgDia = new ProgressDialog(ARView.this);
                        pgDia.setIndeterminate(true);
                        pgDia.setMessage(getString(R.string.loading_elements));
                        pgDia.show();
                         try {
                                getManagerHecateGeoSocial().getLayerNodes(loc);
                        float[] location = new float[3];
```

```
                    location[0] =  (float) loc.getLatitude();
                    location[1] = (float) loc.getLongitude();
                    location[2] = (float) loc.getAltitude();
                    setLocation(location);
                    requestAltitudeInfo();
                    if(idLocationListenerStart!=-1){

                    LocationService.unRegisterLocationListener(idLocationList
                enerStart);
                            idLocationListenerStart = -1;
                    }
                    showResources();
                } catch (Exception e) {
                    try{
                     alertDialog = new AlertDialog.Builder(ARView.this).create();
                           alertDialog.setMessage(e.getMessage());
                           alertDialog.show();
                    }catch(Exception ex){}
            } finally{
                     pgDia.cancel();
            }   };   });  } };
```

El último método importante *showResources()* es el encargado de leer los nodos descargados, filtrarlos por distancia, convertirlos en **ARGeoNode** y enviar a la clase radar para mostrar el resultado en él.

```
    private void showResources() {
            try {
                    refreshed = false;
                    ARGeoNode.clearClicked(getResourcesList());
                    getLayers().cleanResouceLayer();
                    setResourcesList(null);
                    ArrayList<ARGeoNode> res_list;
```

```
                    res_list = ARUtils.cleanNoLocation(this,
getLayers(),getManagerHecateGeoSocial().getNodes());


                if (res_list == null || res_list.size()==0) {
                        Toast.makeText(getBaseContext(),
getString(R.string.error_no_resources_avaliables), Toast.LENGTH_LONG).show();
                        Log.e("ARView",
getString(R.string.error_no_resources_avaliables));
                    return;
                }
                if (res_list.size() > 50) {
                ArrayList<ARGeoNode> list = (ArrayList<ARGeoNode>) res_list.clone();
                res_list.clear();
                for (int i = 0; i < 50; i++)
                res_list.add(list.get(i));
                    Toast.makeText(getBaseContext(),
getString(R.string.error_too_much_objects),
                    Toast.LENGTH_LONG).show();
                    Log.e(ARView.class.getName(),
            getString(R.string.error_too_much_objects));
                }
                ARGeoNodeAzimuthComparator comparator = new
ARGeoNodeAzimuthComparator();
                Collections.sort(res_list, comparator);
                setResourcesList(res_list);
                mRadar.setResourcesList(res_list);
                ARGeoNode.setResourcesList(res_list);
                refreshed = true;
                if (altitude_status.equals(AltitudeManager.ALL_HEIGHTS))
                actionRequestHeight();
            } catch (Exception e) {
                Toast.makeText(       getBaseContext(),
getString(R.string.error_loading_ar_environment)
                + e.getMessage(), Toast.LENGTH_LONG).show();
                        Log.e("ARView", "", e);} }
```

La clase **ARActivity** antes nombrada heredando de **SherlockFragmentActivity** que almacena el método *refreshResourceDrawns.* Este método pinta cada una de los nodos que aparecen por pantalla.

La clase **ARGeoNode** ocupa un papel fundamental en la representación de los puntos y contención de la información. Contiene todos los métodos para interactuar con el punto mostrado en pantalla y activar los procesos de mostrar detalles de información. La instanciación de esta clase se realiza cuando un punto es descargado. Primero se descarga como **GeoNode** y se realiza el proceso de mapeo con esta clase. El proceso incluye operaciones para determinar su posición, cargar la imagen como una View para que pueda ser mostrada en pantalla, etc. Además, el mapeo incluye la ordenación en un listado que luego será utilizado para mostrar la información detallada.

Como dato introductorio, el detalle de cada punto es mostrado en una galería de imágenes (**ARStaticSummaryBox.class)** y permite deslizarse hacia el punto anterior o posterior con el uso del desplazamiento. Gracias a esto se pueden consultar todos los puntos del mapa y si uno no aparece en el campo de visión una flecha indica en qué dirección hay que enfocar.

Continuando con la explicación de ARGeoNode, la representación de la información implica tres clases principales. *ARView,* que gestiona las actualizaciones de la información, su ordenación y llamadas para que sea representada en pantalla. La siguiente clase es **ARGeoNode** que controla tanto el icono informativo del punto que aparecer por pantalla como las acciones para mostrar el detalle del punto. También incluye el propio punto en sí del que se obtiene toda la información que se pinta en el detalle mencionado anteriormente.

```
public class ARGeoNode implements ARNodeDrawingIF{
    private DrawResource drawn;
    private boolean isloaded = false;
    private GeoNode geoNode;
```

```
private float distance;
private float azimuth = 0;
private Point point = null;
private DrawResourceSearcher nodeSearcher;
private Activity mActivity;

private static ARSummaryBox arSummaryBox = null;
private static boolean doCenterSummary = false;
private static Semaphore sem = new Semaphore(1);
private static DrawRadar mRadar = null;
private static boolean refreshIcon = false;
private static boolean isSearchSystem = false;
```

El listado de atributos anterior resume la interacción que puede llegar a tener la clase. El ejemplo de ***mRadar*** permite que al ser seleccionado un punto aparezca en el radar con un color diferente. El atributo **arSummaryBox** es el encargado de comunicarse con la clase que muestra el detalle y al marcar un punto indicarle qué información debe pintar. El atributo **geoNode** contiene la información del punto que será leída y mostrada por pantalla. El último atributo a destacar es **nodeSearcher,** que activa en pantalla una flecha para, como se ha indicado antes en la explicación de la galería, indicar en qué dirección hay que enfocar la cámara para que aparezca en la pantalla.

La clase **DrawResource** también está instanciada en **ARGeoNode** y merece una menciona exclusiva. En ella cada punto que aparece por pantalla concentra las operaciones de representación del punto, es decir, almacena los puntos encargados de refrescar la información, localizarla en pantalla, etc.

```
public class DrawResource extends View {
...
```

Al extender de *View* permite ser representada como un componente en la pantalla. Esta clase dispone del método ***onDraw*** que recibe un *Canvas* (panel donde se

pinta) y en él se pinta la información del punto. En este caso un icono y una descripción, pero en el caso de **DrawResourceSearcher** pinta una flecha que gira a su alrededor según se mueva el dispositivo.

El código que aparece a continuación muestra los cálculos realizados en el método *onDraw* para situar los gráficos en la pantalla.

```
    @Override
protected void onDraw(Canvas canvas){
    int w = canvas.getWidth();      int h = canvas.getHeight();
    cx = ((float)w) / 2;        cy = ((float)h) / 2;

    /* Calculating if the icon should be drawn */
    if((Math.abs(azimuth)>MAX_AZIMUTH_VISIBLE) ||
(Math.abs(elevation)>MAX_ELEVATION_VISIBLE)){
        if(onBoxChangeListener != null)
            onBoxChangeListener.onChange(-1, -1, -1, -1);
        if(onShowIconListener != null)
            onShowIconListener.onShow(false);
        return;
    }
    if(onShowIconListener != null)
            onShowIconListener.onShow(true);

    /* If there is no icon, we request for one */
    if((bitmap_clicked == null) && (onGetIconListener != null))
            onGetIconListener.onGetIcon();

    /* Calculating the screen pixel that corresponds to the node */
    posx = cx +
(float)(Math.sin(Math.toRadians(azimuth))/Math.sin(Math.toRadians(MAX_AZIMUTH
_VISIBLE))) * cx;
    posy = cy +
(float)(Math.sin(Math.toRadians(elevation))/Math.sin(Math.toRadians(MAX_ELEVATI
ON_VISIBLE))) * cy;
    /* Check if the tag is into the big central region */
```

```
        boolean isCenter = false;
        if(isOnCircularArea(cx, cy, posx, posy, ARUtils.transformPixInDip(mContext,
BIG_CENTER_RANGE)))
            isCenter = true;
        /* Check if the tag is into the central region */
        if(isOnCircularArea(cx, cy, posx, posy, CENTER_RANGE))
            if(onCenterListener != null)
                    onCenterListener.onCenter();
/* Calculating the sizes of the elements */
                    if(is_clicked){
                            icon_size = MAX_ICON_SIZE;
                            bitmap = bitmap_clicked;
                            text_size = MAX_TEXT_SIZE;
                    }else if(isCenter){
                            icon_size = MED_ICON_SIZE;
                            bitmap = bitmap_normal;
                            text_size = MED_TEXT_SIZE;
                    }else{
                            icon_size = MIN_ICON_SIZE;
                            bitmap = bitmap_small;
                            text_size = MIN_TEXT_SIZE;
                    }
                    /* Checking the names painting */
                    boolean isName = false;
                    if(name_status.equals(ALL_NAMES))
                            isName = true;
                    else if((name_status.equals(CENTRAL_NAMES)) && isCenter)
                            isName = true;

        /* Painting the icon bg box */
                    Paint paint_bg = new Paint();
                    paint_bg.setARGB(200,0,0,0);
                    paint_bg.setShader(new LinearGradient(posx, posy - (icon_size/2) -
BORDER,
```

```
                                posx, posy + (icon_size/2) + BORDER, Color.WHITE,
Color.BLACK, TileMode.MIRROR));
                paint_bg.setAntiAlias(true);

                RectF backRect = new RectF (posx - icon_size/2 - BORDER, posy -
(icon_size/2 + BORDER),
                                posx + icon_size/2 + BORDER, posy + icon_size/2 +
BORDER);

                canvas.drawRoundRect(backRect, BORDER/2, BORDER/2, paint_bg);

    /* Painting the icon */
                if(bitmap != null)
                        canvas.drawBitmap(bitmap, posx - (icon_size/2), posy -
(icon_size/2), null);

                if(is_clicked || isName){
                        /* Painting the arrow */
                        Paint paint = new Paint();
                        paint.setARGB(250,255,255,255);
                        paint.setAntiAlias(true);
                        paint.setFakeBoldText(true);
                        paint.setTextAlign(Align.CENTER);
                        paint.setTypeface(Typeface.SANS_SERIF);

                        Path path = new Path();
                        path.moveTo(posx - (icon_size/4), posy + (icon_size/2) +
2*BORDER);
                        path.lineTo(posx, posy + (icon_size/2) + BORDER);
                        path.lineTo(posx + (icon_size/4), posy + (icon_size/2) +
2*BORDER);

                        canvas.drawPath(path, paint);

                        /* Painting the name bg box */
```

```
                paint.setTextSize(text_size);
                String text = name;
                float tlongitude = paint.measureText(text);
                if(tlongitude > 2*icon_size){
                        tlongitude = 2*icon_size;
                        paint.setTextAlign(Align.LEFT);
                }
                paint_bg.setShader(new LinearGradient(posx, posy +
(icon_size/2) + 2*BORDER,  posx, posy + (icon_size/2) + 4*BORDER + icon_size/8,
Color.WHITE, Color.BLACK, TileMode.MIRROR));
                paint_bg.setAntiAlias(true);
                backRect = new RectF (posx - tlongitude/2 - 2*BORDER, posy +
(icon_size/2) + 2*BORDER, posx + tlongitude/2 + 2*BORDER, posy + (icon_size/2) +
4*BORDER + icon_size/8);
                canvas.drawRoundRect(backRect, BORDER/2, BORDER/2, paint_bg);
        /* Painting the name */
                path = new Path();
                path.moveTo(posx - tlongitude/2, posy + (icon_size/2) + 2*BORDER +
(2*BORDER + icon_size/8)*3/4);
                path.lineTo(posx + tlongitude/2, posy + (icon_size/2) + 2*BORDER +
(2*BORDER + icon_size/8)*3/4);

                canvas.drawTextOnPath(text, path, 0, 0, paint);
        }
        // calculating box padding
        if(is_clicked && (onBoxChangeListener != null)){
                int left = (int)Math.max(0, (posx -
ARUtils.transformPixInDip(mContext, 125)));
                int top = (int)(posy + (icon_size/2) + 4*BORDER + icon_size/8);
                int right = (int)Math.max(0, (w - (left +
ARUtils.transformPixInDip(mContext, 250))));
                int bottom = (int)Math.max(0, (h - (top +
ARUtils.transformPixInDip(mContext, 200))));

                onBoxChangeListener.onChange(left, top, right, bottom);
        }
```

```
            age = System.currentTimeMillis();
            super.onDraw(canvas);
        }
```

Un detalle importante es que se valora si el nodo está seleccionado o no, en caso afirmativo se trae al frente del resto de nodos para ser más visible y se aumenta su tamaño para que destaque.

```
@Override public boolean onTouchEvent(MotionEvent event){
        Log.i(name, Integer.toString(event.getAction()));
        switch(event.getAction()){
            case MotionEvent.ACTION_DOWN:
                if((System.currentTimeMillis()-age) > 1000 &&
!ARCompassManager.getPauseStatus())
                break;

            float posx = this.posx;
            float posy = this.posy;
            float click_posx = event.getX();
            float click_posy = event.getY();
            if(isOnCircularArea(posx, posy, click_posx, click_posy, CLICK_RANGE)){
                is_clicked = !is_clicked;
                bringToFront();
                if(onIconClickListener != null)
        onIconClickListener.onClick(is_clicked);
            return true;
            }
            break;
}       return false;   }
```

El método anterior captura los eventos lanzados al tocar la pantalla. Si detecta que hay una intersección entre la posición que ocupa el nodo, el listener

onIconClickListener es llamado y activa el nodo como seleccionado, activando las variables correspondientes para que en el próximo refresco (ejecución de *onDraw)* pueda resaltarse el icono.

Otra opción que puede aparecer es que el nodo sea marcado como activo dado que se está consultando en el detalle que aparece en la galería. Esto también fuerza a que sea marcado como seleccionado.

La clase **ARStaticSumaryBox** es la encargada de representar por pantalla el detalle de cada uno de los puntos. Esta clase ha sido mejorada para que incluya las opciones de mostrar tanto los comentarios como las imágenes del punto seleccionado. El modo de representarse ha sido actualizado por completo para ajustarse a las necesidades del proyecto. Algunos de los detalles que se han incluido son la opción de consultar iconos con información de la accesibilidad del punto y la opción de lanzar el *Text-to-Speech*, opción que supone un gran aliciente para las personas a las que va dirigido el proyecto.

Figura 40: Diseño panel informativo

A continuación aparecen el constructor y el método que lanza la operación de mostrar el punto.

```
public ARStaticSummaryBox(Activity mActivity, RelativeLayout container,
RelativeLayout layerExtraInfoPicCom){
    super(container);
    this.mActivity = mActivity;
    this.layerExtraInfoPicCom = layerExtraInfoPicCom;
```

```
        tts = new TextToSpeech(mActivity, this);
}

public void drawSummaryBox(Context mContext, int num_node){
        setNodeClicked(num_node);
        this.mContext = mContext;
        View summary_box = getSummaryBox();
        if(summary_box == null){
                LayoutInflater factory = LayoutInflater.from(mContext);
                summary_box = factory.inflate(R.layout.ar_static_summary, null);
                int h = mActivity.getWindowManager().getDefaultDisplay().getHeight();
                summary_box.setPadding(0, h/2, 0, 0);

                setSummaryBox(summary_box);
                BoxListAdapter mAdapter = new BoxListAdapter(mContext);
                gallery = (Gallery) summary_box.findViewById(R.id.ar_static_gallery);
                gallery.setAdapter(mAdapter);
                gallery.setSpacing(2);
                gallery.setOnItemSelectedListener(onItemSelectedListener);
                }
        if((gallery != null) && (num_node != -1))
                gallery.setSelection(num_node, true);
        }
```

En el segmento de código anterior se aprecia cómo se hacen llamadas para obtener el *View*, que por defecto debe estar instanciado al ser el mismo para todos los nodos, y se carga la galería para añadirle la información a pintar. La galería utiliza la clase **BoxListAdapter** para pintar su contenido.

Las operaciones que se realizan en esta galería o en las galerías en general, implican disponer de un listado ordenado de los puntos a mostrar. El listado que se utiliza es el que desde la clase **ARView** ha sido descargado y ordenado. De ahí viene la importancia de este listado, y no solo para el radar que también muestra los puntos.

A continuación se incluye el método principal del **BoxListAdapter** cuya función es determinar el nodo a pintar e ir extrayendo los elementos de la vista para rellenarlos de información.

```
public View getView(int position, View convertView, ViewGroup parent) {

    View view;
    if (convertView == null) { // Make up a new view
        LayoutInflater inflater = (LayoutInflater)
mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
```

El *inflate* se encarga de instanciar el XML que describe la pantalla y con ello permite que sus elementos puedan ser instanciados también.

```
    view = inflater.inflate(R.layout.ar_static_label, null);
    } else {// Use convertView if it is available
        view = convertView;
    }
```

A partir de ahí la mayoría de las acciones utilizan el *findVewBy* con el fin de detectar el elemento por su identificador y así añadirle el contenido.

```
    TextView t = (TextView) view.findViewById(R.id.ar_static_title);
    t.setText(getTitle(position)); //ok
    t = (TextView) view.findViewById(R.id.ar_static_distance);
    t.setText(" "+ getDistance(position)); //ok default

    t = (TextView) view.findViewById(R.id.ar_static_description);
    t.setText(getDescription(position)!=null?getDescription(position):"" );//ok
    t = (TextView) view.findViewById(R.id.ar_static_page);
    int old_page = getPage(t.getText().toString());
    String page = Integer.toString(position+1) + "/" + Integer.toString(size);
    if(size != 1){
        if(position == 0)
```

```
                page = page + " >";
            else if(position == size-1)
                page = "< " + page;
            else    page = "< " + page + " >";      }
            t.setText(page);
            if(old_page > -1)
                setImageContainer(old_page, null);
            ImageView img = (ImageView) view.findViewById(R.id.ar_image);
            setImageContainer(position, img);
            img.setImageBitmap(getImage(mActivity, position));
            int tipoClass =ImageAdapter.TIPO_SIN;
            if(getClassPDI(position)!=null &&  getClassPDI(position) ){
                tipoClass  = ImageAdapter.TIPO_PDI;
            }else if(getClassPoint(position)!=null &&  getClassPoint(position) ){
                tipoClass  = ImageAdapter.TIPO_PUNTO;
            }
            if(tipoClass != ImageAdapter.TIPO_SIN){
                ImageAdapter auxAdapterAcces = new ImageAdapter(mContext,
tipoClass);             auxInternalGallery = (Gallery)
view.findViewById(R.id.ar_static_gallery_class_pdi_point);
            auxInternalGallery.setAdapter(auxAdapterAcces);
            auxInternalGallery.setSpacing(2);
            lstAccesib = getAccesible(position);
        }
            return view;}
```

Las imágenes y los comentarios son mostrados siguiendo el mismo procedimiento pero con algunas diferencias. En este caso la información se muestra bajo demanda. De no ser así se consumirían excesivos recursos, solo hay que tener en cuenta por cada nodo cargado en la lista pueden haber desde ninguna a varias imágenes. Si todas se cargaran en el mismo instante supondría una sobrecarga innecesaria del sistema, con todas las connotaciones negativas que eso conlleva para el terminal.

En las siguientes imágenes se pueden ver capturas reales de la aplicación mostrando el carrusel de comentarios y de imágenes asociados a un punto.

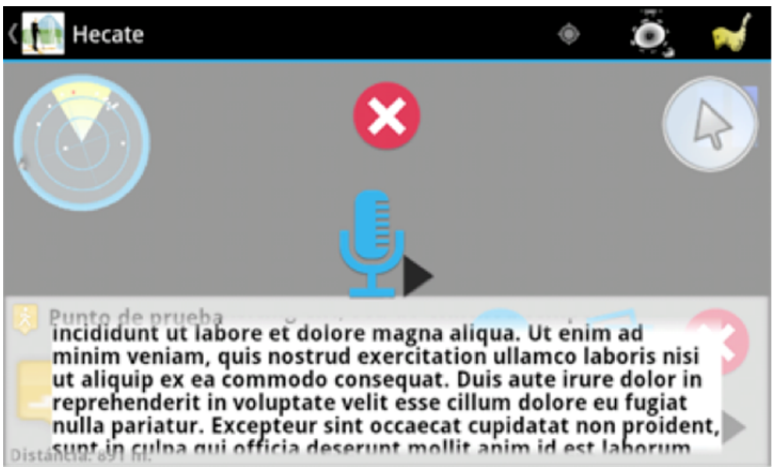

Figura 41: Carrusel de comentarios del punto

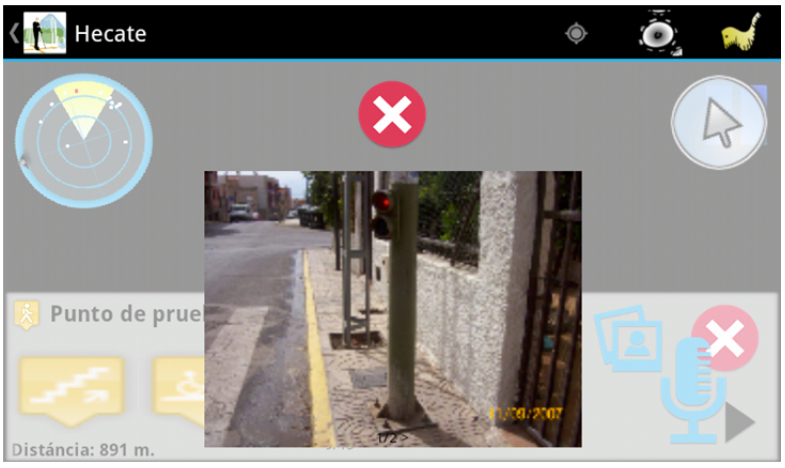

Figura 42: Carrusel de imágenes del punto

5.2 El Servidor Web

La información descrita en este punto trata de explicar el proyecto Hécate en su totalidad. Es necesario explicar esta información extra para poder entender cómo se relacionan todas las partes de la aplicación Hécate en su conjunto. En ningún caso hace referencia al apartado de R. A. descrito, sino al conjunto de elementos que componen Hécate y por lo tanto no han sido desarrollados como parte del trabajo del presente proyecto.

El servidor web es el responsable de alojar la web de Vadeo así como la Base de Datos. Obtiene consultas de los diferentes clientes, las filtra según las necesidades del servicio y las envía de vuelta a la web o el móvil, respectivamente. También es el responsable de alojar el contenido de la web, así como de procesar los distintos algoritmos de cálculo de rutas. Sus funcionalidades básicas son:

- Gestión de usuarios y moderación de contenido.
- Gestión de contenido geo-localizado.
- Gestión de mapas y rutas vía Google Maps.
- Envío de alertas mediante Twitter.
- Gestión de incidencias.
- Publicación de contenido desde GIA.
- Gestión de perfiles y personalización del contenido
- Publicar la API web.

El servidor web aloja la base de datos de Vadeo que contiene toda la información del servicio. Esto es, principalmente, los usuarios, puntos del mapa, comentarios, fotos, rutas y solicitudes.

El servidor web es el componente principal de Vadeo. Se sitúa en el centro de todas las transacciones, ya se inicien en el cliente móvil o el cliente web. En la **¡Error! No se encuentra el origen de la referencia.**, puede verse la visión general. Su papel fundamental consiste en alojar la información generada en Vadeo, ofrecer el contenido a los usuarios y gestionar las peticiones de éstos.

Esta arquitectura centralizada, permite alojar en un mismo lugar la lógica de negocio para ambos clientes, así como realizar las peticiones necesarias al servicio de mapas externo. Esto ofrece varias ventajas, la principal y más importante es que gran parte de la lógica necesaria para soportar el cliente de Vadeo web es la misma que la necesaria para el cliente de Vadeo Móvil. De hecho, el acceso a base de datos es el mismo. Lo que cambia es la forma de representar los datos solicitados. Es por este motivo que se hace uso del patrón de diseño Modelo Vista Controlador. Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos:

- Modelo: es la representación específica de la información con la cual el sistema opera. El modelo abstrae el acceso a la base de datos.
- Vista: presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Controlador: responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo.

En el servidor de Hécate se ha adoptado este patrón, adaptándolo a las necesidades del proyecto. En la siguiente figura puede verse la distribución y la forma en que actúan los componentes de acuerdo al patrón MVC.

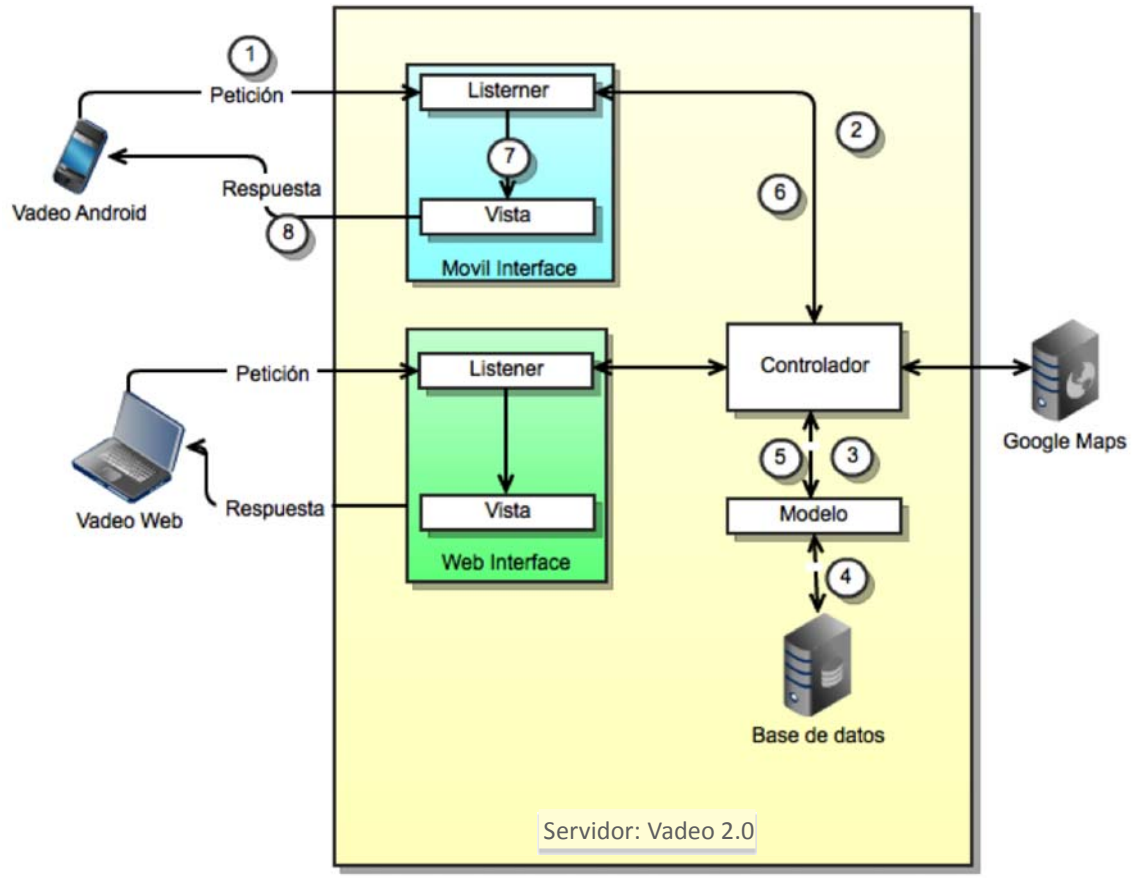

Figura 43: Arquitectura genera, subsistemas generales implicados

El servidor web se compone principalmente de cuatro módulos que se describen a continuación:

- Interfaz de acceso web: punto de entrada para el usuario web. El sub-módulo Listener recibe las peticiones HTTP que el usuario envía mediante el navegador Web, las procesa y las envía al Controlador. La respuesta del Controlador es enviada al sub-módulo Vista, que da formato al contenido en forma de página web HTML y lo envía al usuario.

- Interfaz de acceso móvil: punto de entrada para el usuario móvil. El sub-módulo Listener recibe las peticiones HTTP que el usuario envía mediante el dispositivo móvil, las procesa y las envía al Controlador. La respuesta del Controlador es enviada al sub-módulo Vista, que da formato al contenido en forma de cadena JSON y lo envía al usuario.

- Controlador: es el núcleo principal. Implementa toda la lógica de negocio. Obtiene las peticiones del módulo Listener correspondiente, las procesa, pide los datos necesarios al Modelo, los transforma según las necesidades y las devuelve al Listener.

- Modelo: recibe peticiones del Controlador y las transforma en consultas a la base de datos. Obtiene los datos necesarios de la base de datos, los empaqueta en uno o varios objetos y los devuelve al Controlador.

5.2.1 Modelo de datos

En esta sección se presenta el modelo de datos usado, es decir, cómo se ha modelado la realidad que compone el servicio de Vadeo en entidades propias de la Ingeniería del Software. En primera instancia se define la base de datos implementada, sus principales tablas y campos. A continuación se presenta el diagrama de clases, incluyendo las entidades del servicio y cómo se relacionan. Por último, se detalla la vida de los objetos más relevantes del servicio, es decir, qué estados atraviesan durante su uso.

5.2.2 Base de Datos

La base de datos se ha de diseñar para que sirva tanto para la aplicación Web como la para la aplicación Móvil en Android. Ambas aplicaciones comparten información que representan de un modo diferente. Entre esos datos se encuentran puntos de interés, obstáculos, imágenes o incluso comentarios.

El diseño realizado se divide en diferentes partes bien diferenciadas. La primera corresponde con la representación de los Puntos del mapa, en esta estructura se almacenarán todos los datos relativos a los puntos que pueden suponer un obstáculo para el usuario en su trayecto. Para ello se han creado cinco tablas: la tabla Puntos con la información referida a los mismos, en esta tabla existen dos campos, estado y tipo, que van referidos a otras dos tablas, *estado_punto* y *tipo_punto*, las cuales incluyen la explicación de los estados y tipos existentes asociados a un identificador. Las solicitudes de eliminación de un obstáculo se almacenarán en la tabla solicitudes. Cuando alcance un número de solicitudes elevado, el punto se marcará como bloqueado y no se mostrará.

La segunda parte es la referente a los PDIs, la forma de proceder es muy similar al caso de los obstáculos. Se tendrá una tabla PDI donde almacenar toda la información relativa a los PDIs y las tablas *tipo_pdi* y *estado_pdi*, donde se definen los tipos de PDIs disponibles así como los diferentes estados por los que puede. Además se han añadido las tablas *pdi_accesibilidad*, *accesibilidad_evaluación* y *accesibilidad_atributo*. Estas tablas almacenan la calificación de accesibilidad y los diferentes atributos que desde la herramienta GIA se han definido para un PDI. En la primera se almacena una descripción personalizada de qué características hay disponibles y en qué estado. En la segunda y tercera se indica el atributo que se está tratando (rampas, ascensores, accesos, vestuarios…) y la evaluación que ha recibido que corresponde con un esquema de colores (azul, naranja o verde) según del nivel de accesibilidad que ofrece.

El tercer bloque se corresponde a las Fotos, e incluye todas las tablas referentes a las fotos almacenadas en el sistema. Existe la tabla FOTOS con todos los detalles de cada foto que además se relaciona con los Puntos y PDIs. La tabla ESTADO_FOTOS

indica los posibles estados por los que puede pasar una foto. La tabla SOLICITUDES almacena las solicitudes de los usuarios que desean eliminar una foto. La foto podrá ser manualmente eliminada por el administrador o, según el número de solicitudes, automáticamente marcada como bloqueada y no se visualizará.

En cuarto lugar están los COMENTARIOS, donde se incluye la tabla comentarios, que guarda información de los comentarios; esta tabla incluye una relación con todas las tablas de aquellos elementos que pueden recibir comentarios: Puntos y PDIs. De nuevo, la tabla ESTADO_COMENTARIOS indica los posibles estados por los que puede pasar un comentario.

En quinto y último lugar se encuentra el bloque USUARIOS con las tablas pertenecientes al control de usuarios. La tabla usuarios posee la información referente a los mismos y la tabla *estado_usuarios* contiene el listado de estados que puede tener un usuario. La tabla tipo_usuario indica el tipo de usuario que accede al sistema (Administrador, ayuntamiento, asociación, usuario estándar). La tabla *Preferencias_usuario* almacena el perfil del usuario, qué información le interesa mostrar y qué información le interesa bloquear en el cálculo de rutas.

5.2.3 Diagrama de clases

Un análisis inicial del diagrama indica los objetos protagonistas del servicio, es decir, los que están presentes en la mayoría de los flujos de trabajo. Es fácilmente reconocible el objeto *Punto del Mapa* como eje central del servicio. Esto es correcto, pues toda la información del servicio se basa en puntos geo-localizados que representan un lugar físico.

Un Punto del Mapa puede ser creado por un Usuario estándar o un Usuario de la plataforma GIA. Así mismo, un Punto puede contener fotos, comentarios, solicitudes de eliminación e incidencias. Los puntos se especializan en dos categorías: Puntos de Interés, lugares o edificios en los que se puede desarrollar actividades de diferente índole; y Puntos, elementos que podemos encontrar en el entorno urbano que no son lugares.

Los Puntos De Interés pueden tener asociado atributos de accesibilidad, dónde se define qué características hay disponibles en el edificio tales como rampas, ascensores o baños adaptados.

Toda la información está basada en ubicaciones físicas existentes y por tanto necesita ser referenciada. Por este motivo el objeto Punto del mapa requiere un objeto Coordenada, el cual almacena la latitud y longitud del punto.

El segundo objeto más destacable del diagrama de clases es el objeto Usuario. Representa un usuario web o móvil que accede a la plataforma. Un usuario puede contener preferencias personales, así como almacenar rutas y participar en la definición de puntos.

Gran parte del contenido de Vadeo se plasma en forma de comentarios. El objeto Comentario se asocia con puntos y PDIs.

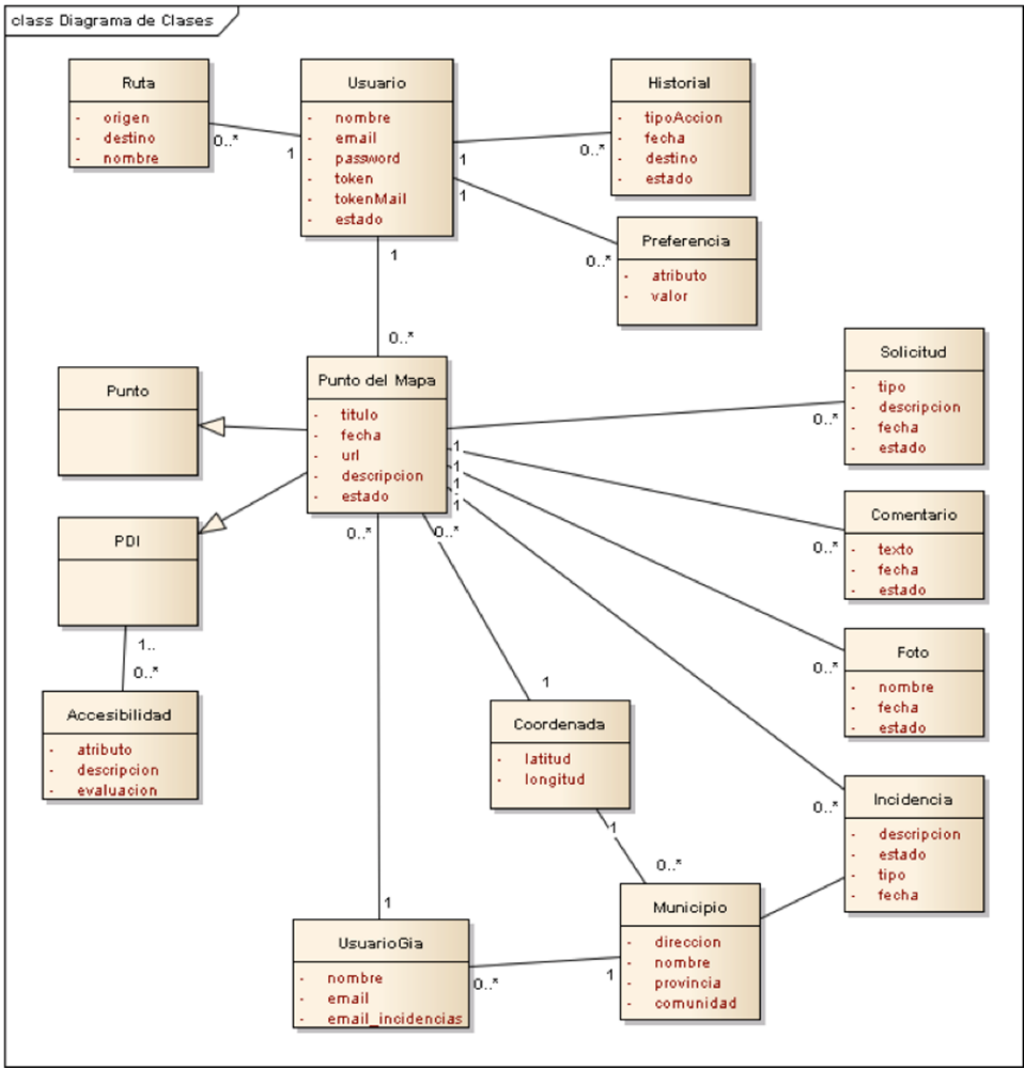

Figura 44: Diagrama de clases

6 Conclusiones

El proyecto descrito en este documento presenta una aplicación de Realidad Aumentada que se incorpora como funcionalidad de la aplicación Vadeo Móvil 2.0 del proyecto Hécate. Esta tecnología permitirá al usuario disfrutar, rápida e intuitivamente, de un medio para consultar la accesibilidad de los espacios públicos de su entorno.

La aplicación de Realidad Aumentada en su conjunto está concluida y operativa para ser incorporada al proyecto Hécate. Los objetivos que se marcaron en un principio se han cumplido en su totalidad. Con la aplicación de Realidad Aumentada un usuario es capaz de localizar su posición, realizar consultas al servidor, descargar la información de accesibilidad de su entorno y consultarla en una interfaz gráfica que incorpora herramientas de accesibilidad. El objetivo de optimización de los recursos también se ha cumplido, agregando Frameworks que reducen el tráfico e integrando toda la información en la aplicación de Vadeo Móvil, sin necesidad de utilizar otras aplicaciones o navegadores.

Algunas mejoras extras, que no estaban en los objetivos iniciales del proyecto, se han añadido durante el proceso de desarrollo. Ha sido en esa etapa, una vez conocida al completo su estructura y código, cuando se ha comprobado que podían incorporarse sin grandes inconvenientes y aportando grandes mejoras para el usuario. Respecto a la accesibilidad se ha optado por incorporar una opción para pausar el contenido y poder elegir fácilmente un punto en la pantalla. En aspectos de diseño y para resolver problemas de fraccionamiento por versiones de Android, se incorporó la librería SherlockActionBar, que simula el diseño de las versiones más actuales en todas las versiones de Android, consiguiendo el mismo estilo en cada una de las versiones del sistema operativo**.**

Durante la elaboración del proyecto son algunos los problemas que han ido surgiendo en las diferentes etapas. Empezando desde la elección de la aplicación de R.A. hasta desglosar el código para añadirlo a Vadeo. Otros de los problemas que surgieron se centraron en el diseño de la interfaz gráfica y la usabilidad de la aplicación

para con los usuarios a los que va destinada, que presentan algún grado de discapacidad. La disponibilidad del código fuente, de herramientas disponibles como el *Text-to-Speech* y del cuidado diseño de la interfaz, han sido suficientes para obtener soluciones óptimas.

En la actualidad, el proyecto está a la espera de comprobar su funcionamiento dentro de Vadeo. Dentro del proyecto Hécate está programada una fase de pruebas con el objetivo de testear la aplicación con usuarios reales con distintas discapacidades con el fin de evaluar la eficiencia y usabilidad de la aplicación. En ese momento se podrán estimar ajustes en la interfaz gráfica para decidir mejoras que puedan agregarse en futuras versiones.

Personalmente, la elaboración de este proyecto me ha permitido descubrir aspectos desconocidas para mí en el mundo de la informática. Empezando por el desarrollo del enfocado a un segmento de la sociedad que puede tener algún problema físico-motriz que les impida utilizar un dispositivo móvil normalmente. Estas deficiencias provocan que simples acciones como leer un texto, o incluso tocar un icono puedan ser un inconveniente. Realizar las aplicaciones más accesibles ha permitido obtener una nueva visión en el mundo del desarrollo. No siempre se puede desarrollar para todos los públicos, pero con esta experiencia puedo apreciar que simples detalles añadidos pueden abrir las puertas a un gran sector de usuarios.

Otro aspecto ha sido la posibilidad de desarrollar una aplicación utilizando nuevas tecnologías tanto a nivel de dispositivos como de Realidad Aumentada. Siempre he sentido curiosidad por su funcionamiento o sus aplicaciones al mundo real. Poder desarrollar el proyecto de final de carrera resolviendo esas dudas y que además pueda ayudar a la gente a mejorar su vida hace que me sienta realmente satisfecho con el esfuerzo realizado.

Para terminar, considero una gran oportunidad haber podido desarrollar este proyecto en colaboración con del Departamento de Tecnologías para la Salud y el Bienestar del instituto ITACA. Espero poder aplicar los conocimientos adquiridos tanto de colaboración como uso de herramientas de desarrollo o incluso nuevas tecnologías en próximos proyectos.

7 Glosario de términos

- Apache Software Fundation (ASF): es una organización no lucrativa creada para dar soporte a los proyectos de software bajo la denominación Apache. Es una comunidad descentralizada de desarrolladores que trabajan cada uno en sus propios proyectos de código abierto. Los proyectos Apache se caracterizan por un modelo de desarrollo basado en el consenso y la colaboración y en una licencia de software abierta y pragmática.

- API (Aplication Programming Interface): representa un interfaz de comunicación entre componentes software.

- ADC2: competición para desarrolladores Android patrocinada por Google.

- Epipolar: tecnología utilizada para capturar una misma imagen desde dos puntos diferentes.

- Framework: conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular. Además sirve como referencia para resolver problemas similares.

- Freemium: modelo de negocios que ofrece servicios básicos gratuitos, mientras que cobra por más avanzados o especiales.

- GPLv3: licencia de publicación de software orientada a proteger la libre distribución. Esta versión v3 incluye nuevas restricciones que impiden la gestión de derechos o DRM en software que lleve esta licencia publicada.

- GPS: Sistema Global de Posicionamiento por satélite que permite determinar la localización o posición de un objeto en todo el mundo.

- IOS: sistema operativo desarrollado por la compañía Apple para dispositivos móviles.

- ISO: organismo encargado de promover el desarrollo de normas internacionales de fabricación. Se centra en buscar la estandarización de normas de productos y seguridad para las empresas a nivel Internacional.

- Kernel: consiste en la parte más importante del sistema operativo, encargado de la comunicación entre el Hardware y las aplicaciones.

- Máquina Virtual: Una máquina virtual se comporta como un software que emula una computadora permitiendo la ejecución de programas en tiempo real.

- MIT: es una institución de educación superior privada situada en Cambridge, Massachusetts, Estados Unidos.

- MVC: es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

- Proxy: es un programa o dispositivo que realiza una acción en representación de otro, esto es, si una hipotética máquina A solicita un recurso a una C, lo hará mediante una petición a B; C entonces no sabrá que la petición procedió originalmente de A. Esta situación estratégica de punto intermedio suele ser aprovechada para soportar una serie de funcionalidades: proporcionar caché, control de acceso, registro del tráfico, prohibir cierto tipo de tráfico etc.

- R.A.: es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real.

- SDK: es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto

- Streaming: es la distribución de multimedia a través de una red de computadoras de manera que el usuario consume el producto al mismo tiempo que se descarga

8 Bibliografía

[1] Colaboradores de Cinu. Personas con discapacidad [en línea]. Naciones Unidas, centro de información [fecha consulta: 28 de agosto del 2012]. Disponible en: <http://www.cinu.org.mx/temas/desarrollo/dessocial/integracion/p_dis.htm>

[2] El INE. Nota prensa Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia (EDAD) año 2008 [en línea]. INE, Instituto Nacional de Estadística, 2012 [fecha consulta: 4 de septiembre del 2012]. Disponible en: <http://www.ine.es/prensa/np524.pdf>

[3] Tipos de parálisis [en línea]. Paralisiscelebral, Tipos de parálisis cerebral, 2012 [fecha de consulta: 24 de agosto del 2012]. Disponible en: <http://www.paralisiscerebral.com/ique-es-la-paralisis-cerebral/7-tipos-de-paralisis-cerebral.html>

[4] Escrito por Aeoe (Asociación Española de la Ovulación y el Embarazo). La espina bífida [en línea]. Asociación Española de la Ovulación y el Embarazo, 2009 [fecha de consulta 25 de agosto del 2012]. Disponible en: <http://aeoe.blogdiario.com/1260785160/la-espina-bifida/>

[5] Colaboradores de Wikipedia. Distrofia muscular de Duchenne [en línea]. Wikipedia, La enciclopedia libre, 2012 [fecha de consulta: 20 de septiembre del 2012]. Disponible en: <http://es.wikipedia.org/w/index.php?title=Distrofia_muscular_de_Duchenne&oldid=58814649>

[6] Vivir con discapacidades [en línea]. Fundación Homero, en un mundo de contradicciones: A problemas comunes soluciones colectivas, 2008 [fecha de consulta 30 de agosto del 2012]. Disponible en: <http://libreopinion.com/members/fundacionhomero/lasdiscapacidades.html>

[7] Tecnología Asistiva [en línea] [fecha de consulta 21 de septiembre del 2012] Disponible en: <http://www.aedin.org/tecnologia-asistiva>

[8] Colaboradores de Wikipedia. Realidad Aumentada [en línea]. Wikipedia, La enciclopedia libre, 2012 [fecha de consulta: 18 de agosto del 2012]. Disponible en: <http://es.wikipedia.org/wiki/Realidad_aumentada>

[9] Carmen Moreno Ortega. Realidad Aumentada, ¿qué es y para qué sirve? [en línea]. Viadeo blog, 2011 [fecha de consulta: 05 de octubre de 2012]. Disponible en: <http://blog.viadeo.com/es/2011/11/17/realidad-aumentada-ejemplos-que-e/>

[10] Colaboradores de Wikipedia. Realidad Aumentada, Software. Wikipedia, La enciclopedia libre, 2012 (fecha de consulta 2 de septiembre del 2012). Disponible en: <http://es.wikipedia.org/w/index.php?title=Realidad_aumentada&oldid=59631870#Software>

[11] Layar - Características [en línea]. Layar, 2012 [fecha de consulta: 10 de febrero del 2012]. Disponible en : <http://www.layar.com/features/>

[12] Componentes del proyecto Wikitude. Documentación [en línea]. Wikitude, 2012 [fecha de consulta: 10 de febrero del 2012]. Disponible en: <http://devzone.wikitude.com/web/forum/documentation>

[13] Comunidad de desarrolladores de Mixare. Uso de Mixare [en línea]. Mixare, 2012 [fecha de consulta: 12 de febrero del 2012]. Disponible en: <http://www.mixare.org/usage/>

[14] Suzy Deffeys. Descripción [en línea]. AndAR, Android Augmented Reality, 2012 [fecha consulta: 12 de febrero 2012]. Disponible en: <http://code.google.com/p/andar/>

[15] Sergio Bellón, Jorge Creixer y Angel Serrano. Presentación [en línea]. Look! Framework de Realidad Aumentada para Android, 2012 [fecha consulta 12 de febrero 2012]. Disponible en: <http://www.lookar.net/presentacion/>

[16] RobertoCalvo y Raúl Román. ARViewer SDK [en línea]. LibreGeoSocial: Augmented Reality FLOSS, 2011 [fecha consulta: 15 de febrero del 2012]. Disponible en: <http://www.libregeosocial.org/node/24>

[17] Colaborador del blog de android-so con pseudónimo *burn15*. Qué es Android? Historia y características del sistema operativo [en línea]. Android-so, blog sobre Android, 2011 [fecha consulta: 4 de agosto del 2012]. Disponible en: <http://android-so.com/que-es-android-historia-y-caracteristicas-del-sistema-operativo>

[18] Aurora Rodriguez, colaboradora de Androideity. Arquitectura de Android [en línea]. Androideity, 2011. [fecha de consulta: 03 de octubre del 2012]. Disponible en: http://androideity.com/2011/07/04/arquitectura-de-android/

[19] Colaboradores de Wikipedia. *Android* [en línea]. Wikipedia, La enciclopedia libre, 2012 [fecha de consulta: 6 de octubre del 2012]. Disponible en <http://es.wikipedia.org/w/index.php?title=Android&oldid=60272666>.

[20] Colaboradores de Wikipedia. Desarrollo de Programas para Android [en línea]. Wikipedia, La enciclopedia libre, 2012 [fecha de consulta: 20 de septiembre del 2012]. Disponible en: <http://es.wikipedia.org/w/index.php?title=Desarrollo_de_Programas_para_Android&oldid=57994319>.

[21] Colaborador de Tarracotadroid, Tomas. Android NDK [en línea]. Tarracotadroid, comunidad de Usuarios Android de Tarragona, 2011 [fecha consulta 01 de octubre del 2012].Disponible en: < http://www.tarracodroid.com/index.php/tutoriales/dificultad-media/49-android-ndk>

[22] Colaborador de JavaHispano, Roberto Montero. Modelos de negocio en el mercado de aplicaciones Android [en línea]. JavaHispano, tu lenguaje, tu comunidad, 2012 [fecha consulta: 10 de agosto del 2012]. Disponible en: <http://www.javahispano.org/android/2012/3/17/modelos-de-negocio-en-el-mercado-de-aplicaciones-android.html>

[23] Staff Layar, Olga. APIs & Web Services[en línea]. Layar, Foro Developers, 2012 [fecha consulta: 06 de octubre del 2012]. Disponible en: http://devsupport.layar.com/entries/20057863-is-there-a-layar-player-sdk-for-android

[24] Definición de usabilidad - Qué es, Significado y Concepto [en línea]. Definición.DE, 2012 [fecha consulta: 12 de noviembre del 2012]. Disponible en: <http://definicion.de/usabilidad/>

[25] María Medina. Ciclo de vida de una Activity [en línea]. Telekita, diario de un proyecto Android, 2012 [fecha consulta: 5 de septiembre del 2012]. Disponible en: <http://telekita.wordpress.com/2012/02/03/ciclo-de-vida-de-una-activity/>

[26] Equipo desarrollo de ActionBarSherlock. Usos de ActionBarSherlock [en línea]. ActionBarSherlock, 2012 [fecha de consulta: 8 de junio del 2012]. Disponible en: <http://actionbarsherlock.com/>

9 Apéndice A

9.1 Especificación de requisitos: Casos de uso

En el capítulo 3 se describe la funcionalidad de la aplicación desde varios puntos de vista y también se introduce a los casos de uso mostrando algunos de ellos.

A continuación se detallan los distintos casos de uso para que aparezcan continuados y así facilitar la comprensión de estos.

- Módulo RA (Realidad Aumentada)
    - Lanzar Vista Realidad Aumentada (R.A.)
    - Obtener coordenadas posiciones
    - Comprobar si hay que recargar
    - Descargar y mostrar información
    - Definir perímetro
    - Mostrar Opciones Configuración
    - Definir filtros
    - Consultar información Punto R.A.
    - Leer con text-to-speech
    - Consultar imágenes punto
    - Consultar Comentarios punto
    - Leer Comentarios con text-to-speech

ID	1	**Nombre**	Lanzar Vista R.A.
Descripción	El usuario selecciona la vista de R.A. del menú principal de la aplicación.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Prioritario		
Datos de entrada	Ninguno		
Precondiciones	El usuario solicita cambiar la vista a R.A. El GPS debe de estar activado		
Flujo Normal	1. El usuario se encuentra en el menú principal de la aplicación 2. El usuario selecciona R.A. 3. La aplicación cambia la vista a R.A.		
Flujo Alternativo	No hay		
Post-condiciones	Se muestra la vista de R.A. con las opciones (elementos) por defecto.		
Excepciones	No		

ID	2	**Nombre**	Obtener coordenadas posición
Descripción	Tras cargarse la vista de R.A. se lanza un proceso para leer continuamente posiciones del GPS.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Prioritario		
Datos de entrada	Aviso de activación de servicio y/o señal de GPS.		
Precondiciones	La vista de R.A. ha sido lanzada correctamente.		
Flujo Normal	1. Se termina de lanzar la vista de R.A. 2. La aplicación lanza un servicio para registrar las entradas del GPS 3. En el momento que se activa el GPS registra sus posiciones.		
Flujo Alternativo	Cada vez que el GPS recibe una señal se activa este proceso. El GPS está configurado para lanzar señal al salir de un perímetro o cada un cierto tiempo determinado.		
Post-condiciones	Al obtener la señal se envía para que sea analizada.		
Excepciones	El GPS no devuelve ninguna señal aun estando activo. Se detiene la espera hasta futuras comprobaciones después de un tiempo.		

ID	3	**Nombre**	Comprobar si hay que recargar
Descripción	El sistema comprueba la señal GPS obtenida para determinar que es una nueva localización y que se precisa de nuevos datos.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Secundario		
Datos de entrada	Coordenadas GPS		
Precondiciones	Se debe de haber leído unas coordenadas nuevas.		
Flujo Normal	1. El sistema ha determinado que hay unas coordenadas nuevas 2. Se comparan las coordenadas para comprobar que ha habido desplazamiento 3. Si hay desplazamiento se envía la orden de descargar nueva información		
Flujo Alternativo	No hay		
Post-condiciones	Se envía la confirmación y las coordenadas para que se descarguen los nuevos puntos relacionados con esas coordenadas.		
Excepciones	No		
Módulo	Capas	**Componente**	Elementos

ID	4	**Nombre**	Descargar y mostrar información
Descripción	El sistema lanza una consulta al servidor con las nuevas coordenadas y los filtros que se hayan establecido.		
Actores	Usuario anónimo, usuario registrado		
Incluye	-		
Nivel de prioridad	Prioritario		
Datos de entrada	Coordenadas GPS y parámetro que confirma o descarta la recarga.		
Precondiciones	Las coordenadas leídas han de ser favorables para una nueva recarga.		
Flujo Normal	1. El sistema comprueba que el parámetro confirma la recarga. 2. Lanza una petición al servidor con las nuevas coordenadas y los parámetros de filtrado. 3. Espera una respuesta del servidor. 4. Al recibir la respuesta filtra la información y la representa por pantalla. 5. Si no recibe nada lanza un aviso y no representa nada en la pantalla		
Flujo Alternativo	No hay		
Post-condiciones	En caso favorable el usuario recibe por pantalla la información representada de los nuevos puntos.		
Excepciones	No		
Módulo	Capas	**Componente**	Elementos

ID	5	**Nombre**	Definir perímetro
Descripción	El usuario determina el parámetro perímetro utilizado en el proceso de obtención de información del servidor.		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar opciones configuración		
Nivel de prioridad	Secundario		
Datos de entrada	Lectura del valor actual del parámetro.		
Precondiciones	El usuario debe de haber accedido a las opciones de configuración.		
Flujo Normal	1. El usuario accede a las opciones de configuración 2. El usuario accede a la opción de perímetro. 3. Es usuario modifica el parámetro de perímetro.		
Flujo Alternativo	No hay		
Post-condiciones	En caso favorable el usuario recibe por pantalla la información representada de los nuevos puntos.		
Excepciones	No		
Módulo	Capas	**Componente**	Elementos

ID	6	**Nombre**	Mostrar Opciones Configuración
Descripción	El usuario lanza las opciones de configuración para elegir una		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar opciones configuración		
Nivel de prioridad	Secundario		
Datos de entrada	Ninguno.		
Precondiciones	El usuario debe de estar en la vista de R.A.		
Flujo Normal	1. El usuario se encuentra en la vista de R.A. 2. Selecciona el botón de configuración.		
Flujo Alternativo	No hay		
Post-condiciones	Aparecen en pantalla las opciones disponibles para elegir y editar los parámetros de filtrado.		
Excepciones	No		
Módulo	Capas	**Componente**	Elementos

ID	7	**Nombre**	Definir filtros
Descripción	El usuario determina los filtros de elementos que serán utilizados en el proceso de obtención de información del servidor.		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar opciones configuración		
Nivel de prioridad	Prioritario		
Datos de entrada	Lectura del valor actual de los parámetros.		
Precondiciones	El usuario debe de haber accedido a las opciones de configuración.		
Flujo Normal	1. El usuario accede a las opciones de configuración 2. El usuario accede a la opción de elementos 3. Es usuario modifica marca o desmarca tantos como sea necesario.		
Flujo Alternativo	No hay		
Post-condiciones	En caso favorable el usuario recibe por pantalla la información representada de los nuevos puntos.		
Excepciones	No		
Módulo	Capas	**Componente**	Elementos

ID	8	**Nombre**	Consultar información Punto R.A.
Descripción	La vista de R.A. muestra la capa con los elemento en cuestión: Punto de Interés (PDI), obstáculo, incidencia o información técnica. El usuario selecciona uno de ellos y la aplicación muestra una ventana sobre la vista con los detalles del elemento: título, tipo, accesibilidad; y las opciones a consultar: fotografías, comentarios, lectura texto.		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar vista de RA		
Nivel de prioridad	Prioritario		
Datos entrada	Identificador del elemento		
Precondiciones	La aplicación muestra la capa correspondiente con al menos un elemento.		
Flujo Normal	1. El usuario selecciona el elemento sobre la vista cuando aparece en su campo de visión aumentada 2. La aplicación busca el elemento entre los descargados 3. La aplicación muestra una cuadro con el resumen de los detalles del elemento: • Título y/o descripción • Tipo de elemento • Accesibilidad 4. Si el usuario desea ampliar esa información, pulsa sobre los botones correspondientes para: • Consultar fotografías disponibles en otra ventana • Consultar comentarios disponibles en otra ventana • Reproducir por voz todo el texto que aparece.		
Flujo Alternativo	Si los datos recibidos están corruptos o no se reciben: 2. Mostrar mensaje al usuario.		
Post-condiciones	Un nuevo cuadro aparece sobre la vista de RA con la información del elemento seleccionada.		
Excepciones	La aplicación no puede mostrar la información		

ID	9	Nombre	Leer con text-to-speech
Descripción	La vista de R.A. muestra la descripción de uno de los puntos y la aplicación reproduce por el altavoz todo el texto que hay en la descripción.		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar vista de RA y descripción del punto.		
Nivel de prioridad	Secundario		
Datos entrada	Identificador del elemento		
Precondiciones	La aplicación muestra la capa correspondiente con al menos un elemento y la descripción de uno de los puntos.		
Flujo Normal	1. El usuario ha selecciona un elemento sobre la vista cuando ha aparecido en su campo de visión aumentada 2. La aplicación ha mostrado en pantalla la descripción del elemento 3. El usuario selecciona el botón de reproducir 4. Se carga el Text-to-speech que reproduce el texto.		
Flujo Alternativo	Si no se puede reproducir el texto aparece un mensaje de alerta con el error.		
Post-condiciones	Se reproduce por el altavoz el texto		
Excepciones	La aplicación no puede reproducir la información		

ID	10	Nombre	Consultar imágenes punto
Descripción	La vista de R.A. muestra la descripción de uno de los puntos y a continuación se carga otra ventana con las imágenes		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar vista de RA y descripción del punto		
Nivel de prioridad	Secundario		
Datos entrada	Ninguno		
Precondiciones	La aplicación muestra la capa correspondiente con al menos un elemento y la descripción de uno de los puntos		
Flujo Normal	1. El usuario ha selecciona un elemento sobre la vista cuando ha aparecido en su campo de visión aumentada 2. La aplicación ha mostrado en pantalla la descripción del elemento 3. El usuario selecciona el botón de imágenes 4. Se carga otra pantalla donde aparecerán las imágenes 5. Se consulta si las imágenes han sido descargadas previamente y si es la primera vez se descargan		
Flujo Alternativo	Si no hay imágenes o no se pueden descargar aparece un mensaje de error		
Post-condiciones	Aparece una nueva pantalla con una galería de las imágenes disponibles		
Excepciones	La aplicación no puede descargar las imágenes		

ID	11	**Nombre**	Consultar Comentarios punto
Descripción	La vista de R.A. muestra la descripción de uno de los puntos y a continuación se carga otra ventana con las comentarios		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar vista de RA y descripción del punto		
Nivel de prioridad	Secundario		
Datos entrada	Ninguno		
Precondiciones	La aplicación muestra la capa correspondiente con al menos un elemento y la descripción de uno de los puntos		
Flujo Normal	1. El usuario ha selecciona un elemento sobre la vista cuando ha aparecido en su campo de visión aumentada 2. La aplicación ha mostrado en pantalla la descripción del elemento 3. El usuario selecciona el botón de comentarios 4. Se carga otra pantalla donde aparecerán las comentarios 5. Se descargan los comentarios relacionados al punto del servidor		
Flujo Alternativo	Si no hay comentarios o no se pueden descargar aparece un mensaje de error		
Post-condiciones	Aparece una nueva pantalla con una galería de las comentarios disponibles		
Excepciones	La aplicación no puede descargar los comentarios		

ID	12	**Nombre**	Leer Comentarios con text-to-speech
Descripción	La vista de R.A. muestra la descripción de uno de los puntos, a continuación se ha cargado otra ventana con una galería de comentarios y se reproduce el comentario actual por el altavoz		
Actores	Usuario anónimo, usuario registrado		
Incluye	Mostrar vista de RA, descripción del punto y la galería de comentarios		
Nivel de prioridad	Secundario		
Datos entrada	Texto comentarios actual.		
Precondiciones	La aplicación muestra la capa correspondiente con al menos un elemento y la descripción de uno de los puntos		
Flujo Normal	1. El usuario ha seleccionado un elemento sobre la vista cuando ha aparecido en su campo de visión aumentada 2. La aplicación ha mostrado en pantalla la descripción del elemento 3. El usuario selecciona el botón de comentarios 4. Se carga otra pantalla donde aparecerán las comentarios 5. Se descargan los comentarios relacionados al punto del servidor 6. El usuario selecciona el botón de reproducción de texto 7. Se lee todo el texto del comentario		
Flujo Alternativo	Si no se pueden reproducir aparece un mensaje de error		
Post-condiciones	Aparece una nueva pantalla con una galería de las comentarios disponibles		
Excepciones	La aplicación no puede leer los comentarios		

10 Apéndice B

10.1Ciclo de vida de las aplicaciones Android

El ciclo de vida de una APP en Android incorpora diferentes estados que deben conocerse para un mejor aprovechamiento del sistema. Para ello la información obtenida de *Teletika [25]* ayuda a su mejor entendimiento.

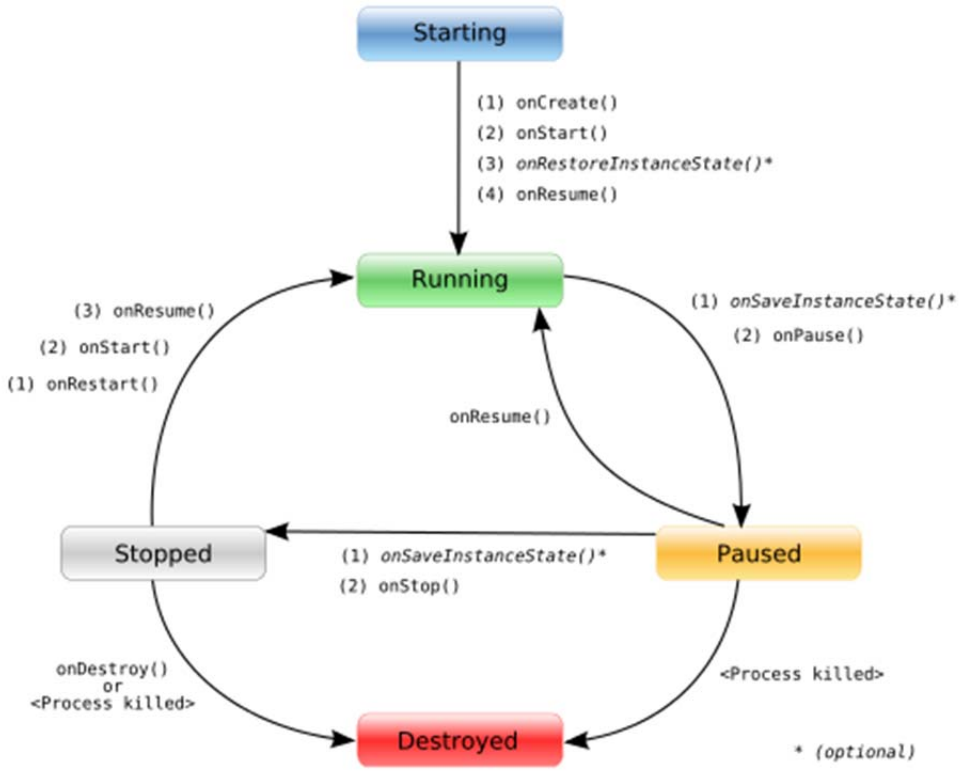

Figura 45: Ciclo de vida de una aplicación

10.1.1 Estados en el ciclo de vida

Activa (*Running*): está la primera en la pila de ejecución, el usuario ve la actividad y puede interactuar con ella.

Pausada (*Paused*): ha pasado a segundo plano en la pila de ejecución, pero aun sigue disponible para futuras llamadas. En este caso, el sistema es el único que puede terminar esta actividad si sus recursos quedan limitados y otras actividades los requieren.

Parada (*Stopped*): ha pasado a segundo plano y otra actividad se está ejecutando en primer plano, de nuevo el sistema puede eliminarla en caso de ver sus recursos limitados.

Destruida (*Destroyed*): ya no está disponible en la pila de ejecución, sus recursos han sido liberados y en el caso de ser llamada de nuevo deberá empezar un nuevo ciclo de vida.

10.1.2 Métodos utilizados en el ciclo de vida

onCreate(): se llama al crear la actividad y en él se prepara la interfaz gráfica de la pantalla. Tras esta función, el proceso sobre el que se ejecuta la Actividad no puede ser destruido por el sistema. El siguiente método que se llama es *onStart*().

onRestart(): se llama cuando una actividad que se había parado vuelve a estar activa, justo antes de que comience de nuevo. Tras ella se llama a *onStart*() y su proceso no puede ser destruido ni durante ni tras su ejecución.

onStart(): se ejecuta justo antes de que la aplicación sea visible al usuario. El siguiente método de ciclo de vida llamado será *onStop*() u *onResume*(), dependiendo de la situación.

onResume(): se ejecuta en el momento en que la actividad se encuentra en la parte superior de la pila, justo antes de que el usuario pueda interactuar con ella. El siguiente método será onPause().

onPause(): se llama cuando la actividad va a ser pasada a un segundo plano de ejecución y almacenada en la *Pila*, por tanto se llama cuando se llame al *onRestart*() de otra. En este método se debe aprovechar para liberar todo aquello que consume recursos (para música, detener procesos…) o guardar datos de manera persistente. No podrá contener tareas lentas ya que hasta que no termine este método no podrá ejecutarse el *onResume()* de la nueva actividad. El método siguiente será *onResume*() u *onStop*().

onStop(): se ejecuta cuando la actividad se hace invisible al usuario. Puede ser porque otra actividad este en primer plano y entonces el siguiente método será *onRestart*() o porque la actividad haya sido destruida, llamado a continuación a *onDestroy*().

onDestroy(): se llama antes de destruir la actividad. Durante la destrucción de la actividad se perderán todos los datos asociados a ella por lo que en este método podrá ser utilizado para controlar la persistencia de datos. Se llamará cuando se ejecuta el método de finalización *finish*() sobre la actividad o porque el sistema elimina la actividad para conseguir más recursos.