UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Una aplicación de Realidad Aumentada en el ámbito Universitario
Autores: Colman, Marisa Roxana - Negri, Gabriel Alejandro
Director: Díaz, Javier
Codirector: Harari, Ivana
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

Hoy, en que el mundo es cada más móvil, los usuarios desean acceder a la información desde cualquier lugar en que se encuentren. Con este argumento, los teléfonos inteligentes son lo más codiciado actualmente en tecnología.
Las principales tendencias tecnológicas en el mercado móvil se centran en Cloud, Contexto y Movilidad. Se esperan nuevas experiencias para interactuar con dispositivos móviles, además de la inclusión de tecnologías como la Realidad Aumentada (RA) y Servicios Basados en Localización (LBS) que harán necesario un replanteo de los modelos de negocios actuales para considerar estas circunstancias.
RA sobre SBL serán los líderes en tecnología móvil integrada en el mercado. La proliferación de teléfonos con GPS y brújula digital ya ha dado lugar a una serie de aplicaciones de RA basado en plataformas de localización.
La combinación de RA con LBS permite contenido gráfico relacionado con la posición del usuario, convirtiéndolo en una de la interfaz de usuario más intuitiva disponible actualmente en el móvil y también hace que el consumo de información del contexto sea mas entretenida.

Palabras Claves

dispositivos móviles, realidad aumentada, geolocalización, android, punto de interés, google maps

Trabajos Realizados

Se brindaron introducciones teóricas sobre LBS, Realidad Aumentada y la plataforma Android.
Aplicando estos conceptos se implementó un prototipo de arquitectura cliente-servidor sobre la plataforma Android que permite visualizar e interactuar con recursos geolocalizados.
El resultado es un prototipo funcional que permite la consulta entidades académicas y eventos en el ámbito universitario.

Conclusiones

Los móviles modernos tienen capacidades que los hacen ideales para desarrollos de aplicaciones interactivas.
Debemos ver a las prestaciones LBS y RA no sólo como categorías en si mismas, sino considerar que brindan valor agregado a servicios existentes, incrementando la utilidad y valor para sus usuarios.
Estas tecnologías, embebidas en Android, marcan un enorme potencial a futuro y su campo de aplicación resulta uno de los más variados.

Trabajos Futuros

Entre los trabajos a futuro podemos nombrar

- Implementar nuevas fuentes de alimentación automática de eventos para las entidades.
- Acoplar distintos elementos de feedback de usuario para mejorar la percepción de éste.
- Analizar la inclusión de elementos sensibles al contexto para una mejor la experiencia de usuario.

Del prototipo desarrollado pueden desprenderse distintas implementaciones para distintos y muy variados ámbitos de aplicación.

Fecha de la presentación: NOVIEMBRE 2011

Una aplicación móvil de realidad aumentada en el ámbito universitario

Marisa Roxana Colman
Gabriel Alejandro Negri

Tesina de grado para obtener el grado de
Licenciatura en Sistemas

Facultad de Informática,
Universidad Nacional de La Plata

Director: Lic. Javier Díaz
Codirector: Lic. Ivana Harari

RESUMEN

Hoy, en que el mundo es cada más móvil, los usuarios desean acceder a la información desde cualquier lugar en que se encuentren. Con este argumento, los teléfonos inteligentes son lo más codiciado actualmente en tecnología.

Las principales tendencias tecnológicas en el mercado móvil se centran en Cloud, Contexto y Movilidad. Se esperan nuevas experiencias para interactuar con dispositivos móviles, además de la inclusión de tecnologías como la Realidad Aumentada (RA) y Servicios Basados en Localización (LBS) que harán necesario un replanteo de los modelos de negocios actuales para considerar estas circunstancias.

La RA sobre LBS serán los líderes en tecnología móvil integrada en el mercado. La proliferación de teléfonos con GPS y brújula digital ya ha dado lugar a una serie de aplicaciones de RA basado en plataformas de localización.

La combinación de RA con LBS permite contenido gráfico relacionado con la posición del usuario, convirtiéndolo en una de la interfaz de usuario más intuitiva disponible actualmente en el móvil y también hace que el consumo de información del contexto sea mas entretenida.

AGRADECIMIENTOS

...

Marisa.-

Llegar a esta instancia no hubiese sido posible sin el apoyo incondicional de mis padres, a Susana y Julio, gracias por la paciencia y confianza en todos estos años.
A Victoria, mil gracias y mil disculpas por las largas noches y fines de semana encerrados.
Agradecer también a todas y cada una de las personas que con su apoyo aportaron para la culminación de este trabajo.

Gabriel.-

ESTRUCTURA DEL TRABAJO

La elaboración del presente informe esta dividida en dos secciones principales:

- Una *Investigación teórica*, que cubrimos en los capítulos 1 a 4, donde exponemos las
- Un *Desarrollo tecnológico*, que describimos en los capítulos 5 a 7.

En la *Investigación teórica* recorrimos los fundamentos teóricos de los temas a desarrollar necesarios para la implementación del prototipo propuesto inicialmente.

En el **Capítulo 1** introducimos los *Dispositivos móviles*, comenzamos con una introducción a la evolución de los teléfonos móviles seguido de una descripción general de su arquitectura. Describimos también detalles de plataformas actuales y particularidades de desarrollo.

En el **Capítulo 2** nos adentramos en la *Realidad Aumentada*, introduciendo el tema completamente, recorriendo por completo todas las actividades que se requieren para implementar un sistema de Realidad Aumentada, para luego concluir con la popularidad de la RA para dispositivos móviles.

En el **Capítulo 3** hablamos sobre los *Servicios basados en localización*. Cuales son las principales características y componentes que forman parte de un sistema basado en localización. Clasificamos las aplicaciones según el área en que se apliquen, y ejemplificamos con aplicaciones existentes en el mercado.

En el **Capítulo 4** describimos la plataforma *Android*. Detallamos las principales características de este sistema operativo móvil, su diseño, arquitectura y funcionamiento. Introducimos la plataforma de desarrollo, herramientas de desarrollo, anatomía de las aplicaciones y sus principales componentes.

Con respecto a la segunda sección de este trabajo, el *Desarrollo tecnológico*, se describe completamente la implementación tanto en la pieza del servidor de la aplicación, como en la pieza de la aplicación cliente para el móvil.

En el **Capítulo 5**, describimos la *Especificación del prototipo*. Definimos la arquitectura y componentes de la aplicación, especificando a través de casos de uso la funcionalidad a implementar y llegamos al modelo de datos a utilizar.

En el **Capítulo 6**, nos referimos a la *Implementación del Administrador Web*, presentamos el Framework Groovy on Grails y implementamos la parte servidor del prototipo.

En el **Capítulo 7**, hacemos lo propio con a la *Implementación del Cliente Móvil*. Implementamos la parte cliente del prototipo describiendo la plataforma Android y decisiones de diseño que se tomaron a lo largo del desarrollo.

Por último, en el **Capítulo 8** presentamos nuestras *Conclusiones* surgidas de este trabajo y los *Resultado obtenidos*, para luego terminar con las tareas a modo de *Trabajo futuro* que pueden desprenderse del resultado de este trabajo.

ÍNDICE GENERAL

1 - DISPOSITIVOS MÓVILES 1
1.1 - ARQUITECTURA DE SISTEMAS MÓVILES 2
1.1.1 - Arquitectura de hardware 2
1.2 - LA EVOLUCIÓN DE LOS DISPOSITIVOS MÓVILES 4
1.2.1 - Primera generación 4
1.2.2 - Segunda generación 4
1.2.3 - Tercera generación 5
1.2.4 - Cuarta generación 5
1.3 - PLATAFORMAS MÓVILES 6
1.3.1 - Licenciado 6
1.3.2 - Propietario 6
1.3.3 - Código libre 7
1.3.4 - Cuadro comparativo de plataformas móviles 7
1.4 - DESARROLLO EN DISPOSITIVOS MÓVILES 8
1.5 - TIPOS DE APLICACIONES MÓVILES 10
1.5.1 - SMS 10
1.5.2 - Sitios Web para móviles 10
1.5.3 - Widgets móviles Web 11
1.5.4 - Aplicaciones Web móviles 11
1.5.5 - Aplicaciones nativas 11
1.5.6 - Juegos 12
1.5.7 - Resumen de tipos de aplicaciones para móviles 12

2 - LA REALIDAD AUMENTADA 13
2.1 - APLICACIONES 15
2.2 - COMO FUNCIONA 17
2.2.1 - Realidad Aumentada basada en marcadores 18
2.2.2 - Realidad Aumentada basada en localización 18
2.3 - ARQUITECTURA DE SISTEMAS RA 19
2.3.1 - Captura de contexto 19
2.3.2 - Identificación de contexto 19
2.3.2.1 - Seguimiento 20
2.3.2.2 - Método 6DOF 20
2.3.2.3 - Técnicas de seguimiento hibridas 21
2.3.3 - Aumento de la realidad 22
2.3.3.1 - Problemas de registro 23
2.3.4 - Visualización 24
2.3.4.1 - Head Mounted Displays 24
2.3.4.2 - Handheld Displays 25
2.3.4.3 - Spatial Displays 25
2.3.5 - Dispositivos de entrada 25
2.4 - HISTORIA 26
2.5 - REALIDAD AUMENTADA EN DISPOSITIVOS MÓVILES 29
2.5.1 - Limitaciones móviles 29
2.5.2 - Aplicaciones para móviles 29
2.5.2.1 - Layer 30
2.5.2.2 - Wikitude 31
2.6 - ESTÁNDARES DE REALIDAD AUMENTADA 32
2.6.1 - Especificación ARML 32
2.6.2 - KHARMA 32

3 - SISTEMAS BASADOS EN LOCALIZACIÓN 34
3.1 - GIS y LBS 35
3.2 - COMPONENTES DE LOS SERVICIOS BASADOS EN LOCALIZACIÓN 36
3.3 - CARACTERÍSTICAS DE LOS SERVICIOS BASADOS EN LOCALIZACIÓN 37
3.4 - CONCEPTOS BÁSICOS DE LBS 38
3.4.1 - Acciones y objetivos asociados al usuario 38
3.5 - TECNOLOGÍA DE POSICIONAMIENTO LBS 40

3.5.1 - GPS 40
3.5.2 - Posicionamiento basado en red 40
3.5.3 - WI-FI 41
3.6 - APLICACIONES BASADAS EN LOCALIZACIÓN 42
3.6.1 - Navegación 42
3.6.2 - Emergencia 42
3.6.3 - Información 42
3.6.4 - Publicidad y facturación 43
3.6.5 - Seguimiento 44
3.6.6 - Juego y ocio 44
3.6.7 - Realidad Aumentada 45
3.6.8 - Redes Sociales 46
3.7 - ESTÁNDARES 47
3.8 - LOS SISTEMAS BASADOS EN LOCALIZACIÓN Y LA PRIVACIDAD 48
3.9 - LIBRERÍAS DE LOCALIZACIÓN 48

4 - SISTEMA OPERATIVO ANDROID 49
4.2 - VERSIONES SDK DE ANDROID 51
4.2.1 - Características 52
4.3 - ARQUITECTURA ANDROID 53
4.4 - INTENTS E INTENTSFILTERS 55
4.4.1 - Objeto Intent 55
4.5 - ANDROID SDK 56
4.6 - COMPONENTES DE UNA APLICACIÓN ANDROID 57
4.6.1 - Activación de componentes: Intents (Intención) 57
4.6.2 - Ciclo de vida de las actividades 58
4.6.3 - Política de eliminación de procesos en Android 59
4.7 - TIPOS DE APLICACIONES ANDROID 61
4.8 - ESTRUCTURA DE UNA APLICACIÓN ANDROID 62
4.9 - HERRAMIENTAS DEL SDK DE ANDROID 65
4.9.1 - Dalvik Virtual Machina 67
4.10 - CUADRO COMPARATIVO DE ANDROID, IPHONE Y WINDOWS MOBILE 68
4.8.1 - Entornos de desarrollo Android e iPhone 69

5 - ESPECIFICACIÓN DEL PROTOTIPO 71
5.1 - CONCEPTOS DE LA APLICACIÓN 72
5.1.1 - Descripción de Conceptos 73
5.2 - LISTA DE REQUERIMIENTOS 75
5.2.1 - Requerimientos del cliente móvil 75
5.2.2 - Requerimientos del administrador Web 75
5.3 - CASOS DE USO 77
5.3.1 - Casos de Uso del cliente móvil 77
5.3.2 - Casos de Uso del Administrador Web 77
5.3.3 - Descripción de los casos de uso Cliente móvil 78
5.3.4 - Descripción de los casos de uso Administrador Web 80
5.4 - ARQUITECTURA DEL SISTEMA 82
5.5 - DIAGRAMAS DE CLASES 84
5.5.1 - Descripción de las clases 85

6 - IMPLEMENTACIÓN DEL ADMINISTRADOR WEB 87
6.1 - EL FRAMEWORK GROOVY ON GRAILS 88
6.2 - CREANDO EL PROYECTO 89
6.3 - CREANDO LAS ENTIDADES 90
6.4 - CREANDO EL CONTROLADOR Y LAS VISTAS 91
6.5 - CONFIGURACIÓN DE LA APLICACIÓN 98
6.6 - FUNCIONALIDADES ADICIONALES 100
6.7 - ESTADÍSTICAS EN LOS EVENTOS 101
6.8 - COMUNICACIÓN ENTRE CLIENTE Y SERVIDOR 102
6.8.1. Implementación de servicios Web con REST en Grails 103
6.9 - SUMINISTRO AUTOMÁTICO DE EVENTOS 106
6.9.1 - Integración con Facebook 106
6.9.2 - Definición de API de eventos 110

7 - IMPLEMENTACIÓN DEL CLIENTE MÓVIL 113
7.1 - CREANDO EL PROYECTO ANDROID 114
7.1.1 - La actividad principal 116
7.1.2 - Google APIs 116
7.1.3 - Implementación de la actividad 117
7.1.4 - Creando la vista de la actividad 118
7.1.5 - Menús de las actividades 119
7.2 - UBICANDO AL CLIENTE 121
7.2.1 - Mostrando la ubicación del usuario 122
7.3 - COMUNICACIONES CON EL SERVIDOR 124
7.3.1 - Representación del modelo en el cliente 125
7.3.2 - Recuperando las entidades y eventos 126
7.4 - VISUALIZANDO ENTIDADES Y EVENTOS 129
7.4.1 - Información de las entidades 133
7.4.2 - Información de los eventos 136
7.4.3 - Listar las actividades de un evento 136
7.4.4 - Filtrar los resultados 137
7.4.5 - Noticias de las entidades 138
7.5 - BUSCADOR DE EVENTOS 143
7.6 - COMO LLEGAR A UN DESTINO 145
7.7 - FEEDBACK DE LOS EVENTOS 148
7.7.1 - Compartiendo eventos 152
7.8 - IMPLEMENTACIÓN DE REALIDAD AUMENTADA 154
7.8.1 - Captura de video 156
7.8.2 - Datos de ubicación 158
7.8.3 - Datos de los sensores 158
7.8.4 - Superposición gráfica 159
7.8.5 - Interacción con los elementos 161
7.9 - PREFERENCIAS DE LA APLICACIÓN 163
7.9.1 - URL del servidor 165
7.10 - FUNCIONALIDADES ADICIONALES 166
7.10.1 - SplashScreen de la aplicación 166
7.10.2 - Sonidos de la aplicación 167
7.10.3 - Visualizar dirección actual 168

8 - CONCLUSIONES Y TRABAJO FUTURO 169
8.1 - Conclusiones 169
8.2 - Resultados obtenidos 170
8.3 - Trabajo futuro 170

REFERENCIAS BIBLIOGRÁFICAS 173
GLOSARIO 175
APÉNDICE A - PREPARACIÓN DEL AMBIENTE DE DESARROLLO 178
APÉNDICE B - ESTADÍSTICAS DE IMPLEMENTACIÓN 183

ÍNDICE DE FIGURAS

Figura 1. Arquitectura de sistemas móviles 2
Figura 2. Arquitectura de hardware de un dispositivo móvil moderno 3
Figura 3. Tipos de aplicaciones móviles 8
Figura 4. Milgram-Virtuality Continuum 13
Figura 5. Arquitectura de Brügge 17
Figura 6. Tareas principales de Realidad Aumentada 19
Figura 7. Movimientos en 6DOF 20
Figura 8. Arquitectura de la plataforma Layer 31
Figura 9. Arquitectura KHARMA 33
Figura 10. Composición de LBS 34
Figura 11. Arquitectura de sistemas LBS 36
Figura 12. Arquitectura del sistema Android 53
Figura 13. Ciclo de vida de una actividad Android 58
Figura 14. Emulador corriendo sobre Windows XP 65
Figura 15. Interacción Android DDMS 66
Figura 16. Estructura de Android ADB 66
Figura 17. Arquitectura básica del sistema AR-DROID 71
Figura 18. Diagrama de conceptos de AR-DROID 73
Figura 19. Arquitectura extendida de AR-DROID 82
Figura 20. Estructura de diseño de Gralis 88
Figura 21. Arquitectura de sincronización por API de eventos 110
Figura 22. Diagrama de clases básico de la actividad principal AR-DROID 116
Figura 23. Diagrama de intercambio de información AR-DROID 126
Figura 24. Diagrama de clases básico del módulo de RA 154

ÍNDICE DE TABLAS

Tabla 1. Cuadro comparativo de plataformas móviles 7
Tabla 2. Comparativa de tipos de aplicaciones móviles 12
Tabla 3. Requerimientos de Realidad Aumentada y Realidad Virtual 13
Tabla 4. Acciones fundamentales en LBS 39
Tabla 5. Versiones de Android 52
Tabla 6. Cuadro comparativo de Android, iPhone 4 y WP7 68

1 - DISPOSITIVOS MÓVILES

Hoy, en que el mundo es cada más móvil, los usuarios desean acceder a la información desde cualquier lugar en que se encuentren; esto es, a través de dispositivos PDA, teléfonos móviles o incluso en automóviles.

Un Smartphones(*) (cuya traducción en español sería "teléfono inteligente") es una evolución del teléfono móvil tradicional que cuenta con ciertas características y prestaciones que lo acercan más a un ordenador personal que a un teléfono tradicional. Entre dichas características, se puede encontrar una mejora en la capacidad de proceso y almacenamiento de datos, conexión a Internet, dispositivos de seguimiento, métodos de entrada avanzados y diversas aplicaciones de usuario como navegadores, aplicaciones ofimáticas, reproductores multimedia, etc.

A pesar de estas importantes mejoras con respecto a sus predecesores móviles, el reducido tamaño de los Smartphones conlleva limitaciones de hardware que los mantienen claramente diferenciados de los ordenadores convencionales [1]. Estas limitaciones se reflejan principalmente en pantallas más pequeñas, menor capacidad de procesamiento, restricciones de memoria RAM y persistente, y necesidad de adaptar el consumo de energía a la capacidad de una pequeña batería.

Aun así, los teléfonos inteligentes son lo más codiciado actualmente en tecnología, y ello no ha pasado desapercibido para las grandes compañías, las empresas ha llevado algunas de sus mejores aplicaciones al mercado de los teléfonos móviles. Además de incluir nuevas.

Muchas veces se asume que la arquitectura de Internet es exactamente igual a la arquitectura móvil es a la de Internet, aunque esta lejos de ser verdad. La arquitectura móvil es una arquitectura única y completa. Al igual que sucede con Internet, se compone de piezas diferentes donde todas deben trabajar de forma integrada. Sin embargo, en la tecnología móvil, las partes que la componen son diferentes, y porque los dispositivos móviles tener acceso a Internet, significa que esta última puede ser una parte de la arquitectura de redes móviles.

1.1 - ARQUITECTURA DE SISTEMAS MÓVILES

Para decirlo de otra manera, pensemos en Internet como una gran nube. Cuando queremos para sacar algo de ella, usamos una herramienta, como una pieza de software o dispositivo. Estas herramientas pueden ser los dispositivos móviles.

Entonces vemos el sistema móvil como un sistema de capas [2], donde cada capa depende de las otras para funcionar:

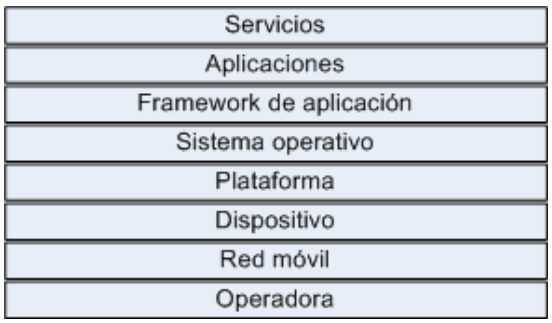


Figura 1. Arquitectura de sistemas móviles

La capa de base en la arquitectura móvil es el operador. Son los que fundamentalmente hacen el trabajo de toda arquitectura móvil: instalan torres de comunicación, gestionan la red celular, hacen que los servicios (como Internet) estén disponible para los suscriptores móviles. El papel del operador en la arquitectura es crear y mantener de manera segura un conjunto específico de servicios de tecnología inalámbrica a través de una red celular.

Las operadoras operan las redes inalámbricas. La tecnología celular, de manera simplista, es una radio que recibe una señal de una antena. El tipo de la radio y la antena determinan las capacidades de la red y los servicios que puede habilitar en ella. La gran mayoría de las redes de todo el mundo utilizan el estándar GSM(*) (Global System for Mobile), pero también existen CDMA(*) (Code Division Multiple Access) y su híbrido CDMA2000(*) 2.5G, que aunque ofrece una mayor cobertura, limitan la cantidad y capacidad de servicios que pueden transferirse sobre ella.

Los dispositivos son los teléfonos celulares, aunque existen tipos de dispositivos inalámbricos que se basan en redes de los operadores, pero no hacen llamadas telefónicas. De ahora en más al referirnos a dispositivos móviles son referiremos a los teléfonos móviles. La mayoría de estos dispositivos son los teléfonos multifunciones, que constituyen la mayoría del mercado. Los teléfonos Smartphones constituyen una pequeña porción del mercado en todo el mundo. Como los dispositivos de próxima generación son una realidad, la distinción entre términos teléfonos y teléfonos inteligentes tiende a desaparecer.

1.1.1 - Arquitectura de hardware

Hoy día un dispositivo móvil ofrece un gran número de posibilidades, como si de una computadora se tratará. Todo esto combinado con una gran potencia de procesamiento y volumen de almacenamiento que permiten la ejecución no sólo de sencillas aplicaciones sino de aplicaciones realmente funcionales. Todo esto sin olvidar otras funcionalidades incorporadas a los móviles, como son el GPS (*), pantallas táctiles, capturadotas de video, sensores, capacidades multimedia, etc.

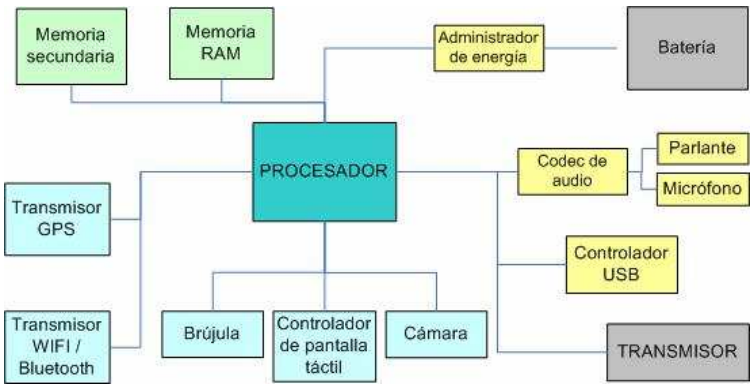


Figura 2. Arquitectura de hardware de un dispositivo móvil moderno

Las capas superiores de la arquitectura de sistemas móviles, las describiremos más adelante, ya que son las que más interesantes para el presente trabajo.

1.2 - LA EVOLUCIÓN DE LOS DISPOSITIVOS MÓVILES

Cada historia tiene un comienzo, y la evolución de los dispositivos móviles no es diferente. La tecnología móvil ha pasado por muchas evoluciones diferentes para llegar a donde esta hoy en día.

En la industria, a menudo se refieren a esta evolución como "generaciones" [3], que se refiere a la madurez y las capacidades de las redes celulares actuales.

1.2.1 - Primera generación

La primera generación (o 1G) es la tecnología de la telefonía móvil de primera generación, los primeros teléfonos celulares. Estas fueron normas de redes móviles, analógicas, que se introdujeron en los años 80 y continuó hasta que fue reemplazado por los teléfonos móviles digitales 2G.

Una de las normas fue NMT(*) (Nordic Mobile Telephone), usada en los países nórdicos, Europa del Este y Rusia. Otra fue AMPS(*) (Advanced Mobile Phone System) utilizada en los Estados Unidos. Antecedentes de la tecnología 1G es el teléfono móvil de radio.

Los teléfonos celulares se hicieron populares y recibieron la demanda del público durante el período 1983 a 1989. Donde los dispositivos eran instalados generalmente en coches, pues los primeros modelos de los dispositivos de primera generación no fueron diseñados para ser portátiles.

Motorola DynaTac

1.2.2 - Segunda generación

Durante la década de 1990, la tecnología sobre la que los teléfonos celulares funcionaban fue llamada 2G. Estados Unidos y Europa utilizan en ese momento proveedores digitales de telefonía celular y redes. Teléfonos celulares 2G tenían una red más rápida que la que funciona en las señales de radio. 2G sustituyó a la red de las frecuencias analógicas y finalmente, la adaptación de las redes modernas hechas las frecuencias analógicas obsoletas.

Los teléfonos celulares 2G eran más pequeños, alrededor de 100 a 200 gramos. Estos fueron de mano y eran portátiles. Normalmente no podían transferir datos, como correo electrónico o software, que no sea datos digitales de voz, y otros datos básicos auxiliares, como fecha y hora. Sin embargo, aparece el servicio SMS(*) como una forma de transmisión de datos para algunas de las normas. Por todos los avances que involucraron a los dispositivos móviles, sus baterías, y su procesamiento, la popularidad del teléfono celular creció rápidamente.

La tecnología 2G se puede dividir en las normas TDMA(*) y CDMA(*), basada en función del tipo de multiplexación que utilizan. Las normas que se implementaron en esta generación son:

- GSM (TDMA), originaria de Europa pero que se utilizan en todo el mundo.
- IDEN (TDMA), propietaria de la red utilizada por Nextel en los Estados Unidos y Telus en Canadá.
- ES-136 (TDMA), también conocida como D-AMPS, utilizado en el continente americano.
- IS-95 también (CDMA), conocida como cdmaOne, que se utiliza en América y partes de Asia.
- PDC (TDMA), utilizada exclusivamente en Japón

1.2.3 - Tercera generación

La tecnología 3G es la última de las de las generaciones en comunicaciones móviles. Esta diseñada para que dispositivos móviles que incluyen una experiencia multimedia verdadera, con características de mayor velocidad de transferencia para dar cabida a aplicaciones como las basadas en Web. Los servicios asociados con 3G proporcionan la posibilidad de transferir tanto voz como datos, que incluyen la descarga de información, intercambio de correo electrónico, mensajería instantánea y video en línea

Las tecnologías de 3G son una respuesta a la especificación de la Unión Internacional de Telecomunicaciones en el año 2000. Originalmente, la 3G se supone que es una única y un estándar unificado para todo el mundo, pero en la práctica, el mundo de la 3G se ha dividido en tres:

- UMTS (Universal Mobile Telephone System), basado en la tecnología CDMA, es la solución generalmente preferida por los países que utilizan GSM, centrada en Europa. UMTS está gestionado por la organización 3GPP, también responsable de GSM, GPRS(*) y EDGE(*).
- CDMA2000 es otro estándar 3G importante, que es una consecuencia de la norma anterior 2G CDMA IS-95. Sus zonas de influencia principales son Japón y Corea. CDMA2000 es administrado por 3GPP2, que es una organización independiente de 3GPP(*).
- TD-SCDMA es un estándar menos conocido, que fue desarrollado en la República Popular de China por la empresa Datang y Siemens.

Las redes 3G tienen una velocidad de transferencia potencial de hasta 3 Mbps. Por comparación, el teléfonos más rápido de 2G podía alcanzar hasta 144kbps. Los teléfonos 3G, son como mini-ordenadores portátiles y puede adaptarse a aplicaciones de banda ancha como la videoconferencia, streaming de vídeo desde Web, y enviar y recibir de inmediato mensajes de correo electrónico con archivos adjuntos.

1.2.4 - Cuarta generación

La tecnología móvil 4G es el nombre dado a la próxima generación de dispositivos móviles. No existe aún un estándar de la industria de que es lo que constituye la red 4G [4], así que por ahora no es más que un término de marketing.

No importa lo que la tecnología está detrás de él, la idea de 4G estará destinada para ofrecer velocidad. En promedio, 4G se supone que va a ser de cuatro a diez veces más rápido que las actuales redes 3G.

Proveedores actuales como Sprint dice que su red WiMax 4G ofrece velocidades de descarga que son diez veces más rápido que una conexión 3G, con velocidades que alcanzan un máximo de 10 megabits por segundo. En cambio Verizon, por su parte, con su red LTE, asegura que puede alcanzar velocidades que varían entre 5 y 12 Mbps.

Teléfono Google G1

1.3 - PLATAFORMAS MÓVILES

Primero debemos definir que una plataforma móvil, al igual que sucede con las computadoras de escritorio, es la capa que existe para proporcionar acceso a los dispositivos. Para ejecutar cualquier aplicación o servicio en cada uno de los dispositivos móviles, se necesita una plataforma. Esto es un lenguaje básico de programación, en el que todo el software está escrito. Como todas las plataformas de software, estas son divididas en tres categorías: Licenciado, Propietario, y Código libre.

En los últimos años hemos visto un notable crecimiento en las prestaciones de las plataformas de software diseñadas para dar soporte a los dispositivos móviles, así como en las prestaciones que estos sistemas aportan. Hoy en día, estas plataformas se aproximan más a lo que son los sistemas operativos de computadoras de sobremesa que a simples interfaces de acceso a los recursos de los dispositivos móviles.

1.3.1 - Licenciado

Las plataformas licenciadas se venden a fabricantes de dispositivos para la distribución exclusiva en los dispositivos. El objetivo es crear una plataforma común de desarrollo de interfaces de programación de aplicaciones (*APIs*) que funcionan de manera similar a través de varios dispositivos con el menor posible esfuerzo necesario para adaptarse a las diferencias de hardware, aunque esto no es del todo cierto.

Las aplicaciones licenciadas que existen hoy son:

- Java Micro Edition (Java ME): Anteriormente conocido como J2ME, Java ME es, por mucho, la plataforma de software más predominante en cualquier tipo en dispositivo móvil. Java ME combina una JVM y un conjunto de APIs de Java para desarrollar aplicaciones para dispositivos de recursos limitados, como son los móviles.
- Windows Phone: Anteriormente llamado Windows Mobile, es una versión licenciada y compacta del sistema operativo Windows, combinado con un conjunto de aplicaciones básicas para dispositivos móviles que se basa en el API Win32 de Microsoft.
- Entorno de ejecución de binarios para Wireless (BREW): BREW es una plataforma creada por Qualcomm(*) para dispositivos móviles. Las características técnicas incluyen: Una SDK(*), un modelo basado en APIs independientes que cubren toda la funcionalidad básica de un terminal, y finalmente es multiplataforma con soporte a gran cantidad de fabricantes.

1.3.2 - Propietario

Las plataformas propietarias son diseñadas y desarrolladas por los fabricantes de dispositivos para el uso exclusivo en sus dispositivos. No están disponibles para otros fabricantes de dispositivos de la competencia. Incluyen las plataformas:

- BlackBerry: BlackBerry mantiene su propia plataforma propietaria basada en Java, que se utiliza exclusivamente por sus dispositivos.
- Palm: Palm utiliza tres diferentes plataformas propietarias. Su primer y más reconocida es la plataforma Palm OS basado en el lenguaje C/C++, la que fue inicialmente desarrollada para su línea de Palm Pilot. Con la aparición teléfonos inteligentes, Palm creó webOS, que es una plataforma basada en Linux y su ejecución se hace totalmente en el marco del navegador WebKit,
- iPhone: Apple utiliza una versión compacta de Mac OS X como plataforma para su línea de dispositivos iPhone, iPad e iPod, que ciertamente esta basada en Unix. Actualmente esta en la versión 4G. Provee todas las herramientas y recursos para construir aplicaciones nativas con un buen soporte para multimedia y gráficos.
- Symbian: La plataforma Symbian es el software de código abierto basado en Symbian OS, el sistema más utilizado operativo abierto para teléfonos móviles. La Fundación Symbian es una organización sin fines de lucro, fundada por Nokia, Motorola, Sony Ericsson, NTT DoCoMo, Texas Instruments, Vodafone, Samsung, LG y AT & T.

1.3.3 - Código libre

Plataformas de código abierto son las plataformas móviles que están disponibles gratuitamente para los usuarios que pueden descargar, modificar y editar. Estas plataformas son las más recientes en móviles, pero aun así cada vez están ganando más confianza con los fabricantes de dispositivos y desarrolladores. Una de estas plataformas es Android, desarrollada por la Open Handset Alliance(*), encabezada por Google. La Alianza busca desarrollar una plataforma de código abierto para móviles basado en el lenguaje de programación Java. Describiremos en detalle la plataforma Android más adelante.

Nombramos en esta instancia la existencia de otras plataformas libres, basadas en Linux:

- OpenMoko. Hace uso del núcleo Linux con un entorno gráfico implementado con el X.Org, GTK+. OpenMoko es desarrollado por First International Computer (FIC) bajo la GPL.
- MeeGo: Esta plataforma es capaz de funcionar en teléfonos móviles, además de una variedad de dispositivos, como netbooks, sistemas en vehículos, televisores, etc. MeeGo esta basada en el gestor de paquetes RPM, y cuenta con el auspicio de la Linux Foundation.
- LiMo: LiMo es una fundación conformada por varias empresas del mercado de los dispositivos móviles. Esta desarrollado para asistir en el diseño, desarrollo y despliegue de dispositivos móviles. Esta compuesto por un sistema modular, una arquitectura de plugins, sobre un SO con un entorno seguro en tiempo de ejecución.

Podemos notar que la plataforma esta muy ligada al sistema operativo en plataformas propietarias. Las plataformas licenciadas, al igual que las de código libre, disponen de una especificación que permite a los fabricantes de dispositivos hacer equipos compatibles con tales plataformas.

Trataremos los sistemas operativos y Frameworks(*) de aplicación más adelante, cuando describamos la plataforma móvil Android. Compararemos el sistema operativo de Google contra sistemas como son iPhone, de Apple, y su SDK para desarrolladores, o Windows Phone, de Microsoft, con su Framework de aplicación basado en .NET.

1.3.4 - Cuadro comparativo de plataformas móviles

Como conclusión a la presentación de las plataformas, donde nombramos las más populares, presentamos esta tabla comparativa que pretende dar una visión general de todas ellas para poder extraer algunas ideas importantes.

PLATAFORMA	CÓDIGO LIBRE	API/SDK LIBRE	LICENCIA	BASADO EN
SYMBIAN	*SI (EN 2010)*	*SI*	*ECLIPSE PUBLIC LICENSE (EPL)*	*SYMBIAN OS*
ANDROID	*SI*	*SI*	*APACHE*	*LINUX*
LIMO	*SI*	*SI*	*GPL*	*LINUX*
OPENMOKO	*SI*	*SI*	*GPL*	*LINUX*
PALM	*NO*	*SI*	*PROPIETARIO*	*LINUX*
IPHONE	*NO*	*SI*	*PROPIETARIO*	*DARWIN*
LIMO	*SI*	*SI*	*GPL*	*LINUX*
MEEGO	*SI*	*SI*	*GPL2*	*LINUX*
RIM	*NO*	*SI*	*PROPIETARIO*	*BLACKBERRY OS*
WINDOWS MOBILE	*NO*	*SI (API WIN32)*	*MICROSOFT EULA*	*WIN32*
BREW	*NO*	*SI*	*COMERCIAL*	-
JAVA ME	*SI*	*SI*	*GPL*	*JRE*

Tabla 1. Cuadro comparativo de plataformas móviles

1.4 - DESARROLLO EN DISPOSITIVOS MÓVILES

El desarrollo de aplicaciones móviles conlleva una variedad de consideraciones de acuerdo al propósito y escenario para el que van a ser utilizadas. Hace algunos años se tenia la creencia que desarrollar aplicaciones móviles era igual a desarrollar una aplicación tradicional pero en "pequeño".

Debemos empezar por conocer los tipos de aplicaciones que podemos implementar para los dispositivos móviles.

Figura 3. Tipos de aplicaciones móviles

El desarrollo de aplicaciones móviles difiere del desarrollo de software tradicional en muchos aspectos, lo que provoca que las metodologías usadas para estos entornos también difieran de las del software clásico. Esto es porque el software móvil tiene que satisfacer una serie de requerimientos y condicionantes especiales [5] que lo hace más complejo:

- Canal radio: consideraciones tales como la disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad deben tenerse especialmente en cuenta en este entorno de comunicaciones móviles.
- Movilidad: aquí influyen consideraciones como la migración de direcciones, alta latencia debido a cambio de estación base o la gestión de la información dependiente de localización. Sobre esta última, de hecho, se pueden implementar un sinfín de aplicaciones, pero la información de contexto asociada resulta muchas veces incompleta y varía frecuentemente.
- Portabilidad: la característica portabilidad de los dispositivos implica una serie de limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas, o del teclado.
- Fragmentación de la industria: la existencia de una considerable variedad de estándares, protocolos y tecnologías de red diferentes añaden complejidad al escenario del desarrollo móvil.
- Capacidades limitadas de los terminales: incluimos factores como la baja potencia de cálculo o gráfica, los riesgos en la integridad de datos, las interfaces de usuario poco funcionales en muchos aspectos, la baja capacidad de almacenamiento, la duración de las baterías o la dificultad para el uso de periféricos en movilidad. Factores todos que, por otro lado, están evolucionando en la dirección de la convergencia de los ultra portátiles (netbooks) con los dispositivos inteligentes constituyendo cada vez menos un elemento diferencial.
- Usabilidad: las necesidades específicas de amplios y variados grupos de usuarios, combinados con la diversidad de plataformas tecnológicas y dispositivos, hacen que el diseño para todos se convierta en un requisito que genera una complejidad creciente difícil de acotar.
- Time-to-market: en un sector con un dinamismo propio, dentro de una industria en pleno cambio, los requisitos que se imponen en términos de tiempo de lanzamiento son muy estrictos y añaden no poca dificultad en la gestión de los procesos de desarrollo.

El diseño de sistemas de software móvil es, por tanto, bastante más complejo que el tradicional visto en los otros proyectos de desarrollo, forzando a los investigadores a reconsiderar el uso de las metodologías actuales de desarrollo de software. Como se ha visto, el uso de metodologías ágiles es el medio más apropiado para el desarrollo de tecnología en móviles, aunque las características especiales de los terminales y de las redes de telefonía móvil demandan algunos ajustes sobre las actuales metodologías ágiles.

Ha pasado ya mucho tiempo desde el estreno de plataformas abiertas para el desarrollo de

aplicaciones móviles como Java o Symbian. La disponibilidad de herramientas y la facilidad para construir y comercializar software para el móvil ha hecho que su número se dispare. A eso se ha añadido, tal como se menciona más arriba, la aparición de de las metodologías ágiles adaptadas para el desarrollo de aplicaciones para el escritorio.

Los móviles son un medio diferente y se rige por un conjunto diferentes de reglas. Grandes productos móviles son capaces de adaptarse e incluso ir más allá de la estrategia tradicional para identificar nuevas soluciones y una forma única para hacer frente a los retos y los beneficios que el medio móvil tiene para ofrecer.

1.5 - TIPOS DE APLICACIONES MÓVILES

Como hemos mencionado anteriormente, el ecosistema móvil es grande y complejo. Se tiene un número de opciones en los que el medio se utiliza para hacer frente los objetivos, cada uno con sus pros y sus contras. Algunos desarrollos se apresuran a crear aplicaciones funcionales, pero accesible a menos usuarios. Otros se dirigen a un mercado más grande, pero son mucho más complejos y costosos.

Cuando se enfrenta el desarrollo de un nuevo sistema móvil, se tiene que decidir en qué tipo de contexto la aplicación que desea para presentar su contenido o información. En otras palabras, qué tipo de aplicación es el más adecuado a nuestro problema o necesidad. A continuación se tratan los diferentes tipos de aplicaciones móviles y se definen cada una de estas opciones [6].

1.5.1 - SMS

La aplicación móvil más básica que podemos crear son las de tipo SMS. A pesar de que podría parecer extraño considerar las aplicaciones de mensajes de texto, son sin embargo un diseño con experiencia. Dada la ubicuidad de los dispositivos que soportan SMS, estas aplicaciones pueden ser herramientas útiles cuando se integra con otros tipos de aplicaciones móviles.

Normalmente, el usuario envía una palabra clave a un código corto de pocos dígitos que resulta en la devolución de información o un enlace a contenidos de calidad.

Los usos más comunes de las aplicaciones SMS son la descarga de contenido móvil, tal como tonos de llamada e imágenes o para interactuar con bienes y servicios reales. Los mensajes SMS también pueden utilizarse para compra de tiempo en un estacionamiento o zona de pago.

Las ventajas de las aplicaciones SMS son:

- Trabajan en cualquier dispositivo móvil casi instantáneamente.
- Son útiles para el envío de alertas a tiempo al usuario.
- Se pueden incorporar en cualquier Web o de aplicación móvil.
- Pueden ser fáciles de configurar y administrar.

Las desventajas de las aplicaciones SMS son:

- Están limitados a 160 caracteres.
- Ofrecen una experiencia limitada basada en texto.
- Las implementaciones pueden resultar costosas.

1.5.2 - Sitios Web para móviles

Como es de esperar, un sitio Web para móviles es un sitio Web diseñado específicamente para móviles dispositivos, que no debe confundirse con la visualización de un sitio hecho para los navegadores en un móvil. Estos se caracterizan por su sencilla arquitectura "drill-down"(*), o la simple presentación de los enlaces de navegación que le llevará a una página de un nivel más profundo.

Estos sitios Web móviles suelen tener un diseño simple y son típicamente de naturaleza informativa, que ofrece pocos -si alguno- de los elementos interactivos se puede esperar de un sitio de escritorio.

Como mejores navegadores móviles comenzaron a ser introducidos en plataformas de dispositivos como el iPhone y Android, la calidad de sitios Web para móviles comenzaron a mejorar de manera espectacular, y mejoró su utilización.

Las ventajas de sitios Web para móviles son:

- Son fáciles de crear, mantener y publicar.
- Se pueden utilizar todas las mismas herramientas y técnicas que en sitios de computadoras de escritorio.
- Casi todos los dispositivos móviles pueden ver sitios Web para móviles.

Las desventajas de sitios Web para móviles son:

- El soporte puede ser difícil a través de múltiples dispositivos.
- Ofrecen a los usuarios una experiencia limitada.
- Se pueden cargar las páginas lentamente, debido a la latencia de la red.

1.5.3 - Widgets móviles Web

En gran parte en respuesta a la mala experiencia proporcionada por la Web móvil a través de los años, ha existido un creciente movimiento para establecer Widget(*). Este es un componente de interfaz de usuario con un bloque independiente de código HTML que es ejecutado por el usuario de una manera particular.

Básicamente, los Widgets Web para móviles son pequeñas aplicaciones Web que no se puede ejecutar por sí mismos. Puede crear atractivas experiencias de usuario que aprovechan las funciones del dispositivo y, en muchos casos, requieren ejecutarse mientras el dispositivo está en línea.

Las ventajas de Widgets Web para móviles son:

- Son fáciles de crear, a partir de HTML, CSS, JavaScript y el conocimiento.
- Pueden ser fácil de implementar a través de múltiples dispositivos.
- Ofrecen una experiencia de usuario mejorada y un diseño más rico, y algunos no requieren conexión.

Las desventajas de Widgets Web para móviles son:

- Por lo general requieren una plataforma de Widgets compatibles para ser instalado en el dispositivo.
- Generalmente no se puede ejecutar en cualquier navegador Web para móviles.
- En algunos casos, se requiere el aprendizaje adicional técnicas propias de Widgets.

1.5.4 - Aplicaciones Web móviles

Las aplicaciones Web móviles son aplicaciones móviles que no necesitan ser instaladas o compiladas en el dispositivo de destino. Usando XHTML, CSS y JavaScript, son capaces de proporcionar una rica experiencia de la aplicación para el usuario final.

Este tipo de aplicaciones permiten a los usuarios interactuar con el contenido en tiempo real, donde un clic o toque realiza una acción dentro de la vista.

El desarrollo y uso de este tipo de aplicaciones explotó con la llegada del iPhone, y más tarde los dispositivos con Android. Esto es porque los fabricantes instalaron un movimiento para mejorar los navegadores de sus dispositivos, dotándolos de soporte estándar casi completo para XHTML, CSS, JavaScript y Ajax para aprovechar este nuevo medio de aplicaciones.

Las ventajas de las aplicaciones Web para móviles son:

- Son fáciles de crear, a partir de HTML, CSS, JavaScript y contenido.
- Son fáciles de implementar a través de múltiples dispositivos.
- Ofrecen una mejor experiencia de usuario y un diseño rico, aprovechando las características del dispositivo.
- El contenido es accesible en cualquier navegador Web para móviles.

Las desventajas de las aplicaciones Web para móviles son:

- La experiencia óptima podría no estar disponible en todos los terminales.
- Puede resultar un reto (aunque no imposible) el apoyo a través de múltiples dispositivos.
- No siempre se cuenta con el apoyo nativo de aplicaciones, como el modo en línea, ubicación, operaciones de búsqueda, sistema de ficheros de acceso, cámaras, etc.

1.5.5 - Aplicaciones nativas

El medio de aplicación más antigua y común. Estas aplicaciones nativas o de la plataforma se construyen específicamente para los dispositivos que ejecutan la plataforma de que se trate. El más común de todas las plataformas es Java ME.

Con la llegada de los Smartphones, las SDK de cada plataforma consiguen una implementación mucho más específica. Una capa del sistema operativo y APIs añadidos permiten a los desarrolladores implementar más fácilmente las tareas complejas de la API en lugar de escribir métodos a partir de cero.

La creación de aplicaciones utilizando una plataforma de aplicaciones significa decidir los dispositivos destino, teniendo medios de ensayos y certificación, y un método para distribuir la aplicación

a los usuarios. La gran mayoría de las aplicaciones nativas están certificadas, se venden y distribuyen a través de un portal del operador o una tienda de aplicaciones.

Con las aplicaciones nativas que se desarrollan sobre la capa superior de la plataforma, se puede aprovechar la mayoría de las características del dispositivo.

Las ventajas de las aplicaciones nativas incluyen:

- Ofrecen la mejor experiencia de usuario, ofreciendo un diseño rico y aprovechan las funciones del dispositivo incluso sin conexión.
- Son relativamente fácil de desarrollar para una sola plataforma.
- El cobrar por las aplicaciones es más fácil.

Los contras de las aplicaciones nativas incluyen:

- No son fácilmente portadas a otras plataformas móviles.
- El desarrollo, prueba y soporte de múltiples plataformas de dispositivo es costoso.
- Requieren certificación y distribución de un tercero que no tiene control.
- Generalmente se necesita compartir los ingresos con una o más terceras partes.

1.5.6 - Juegos

El de los juegos es el medio más popular de todos los medios disponibles para móviles dispositivos. Técnicamente, los juegos son en realidad aplicaciones nativas que utilizan la SDK de la plataforma para crear experiencias de inmersión. Pero los tratamos de forma diferente de las aplicaciones nativas por dos razones: no pueden ser fácilmente duplicada como aplicaciones Web, y la migración a otras plataformas móviles es un poco más fácil que las típicas aplicaciones nativas.

Los juegos son relativamente fáciles de portar, es que la mayor parte de un juego esta en los gráficos y de hecho utiliza muy poco de la API dispositivo. La mecánica del juego es lo único que tiene que adaptarse a las diversas plataformas.

Las ventajas de aplicaciones de juegos son los siguientes:

- Proporcionan una manera simple y fácil de crear una experiencia de inmersión.
- Pueden ser portado a múltiples dispositivos con relativa facilidad.

Los contras de aplicaciones de juegos son los siguientes:

- El desarrollo puede ser costoso para un juego original.
- No puede ser portado a la Web móvil.

1.5.7 - Resumen de tipos de aplicaciones para móviles

Presentamos la siguiente tabla a modo de resumen, para poder hacer una comparación de los distintos medios para el desarrollo de aplicaciones móviles.

	DISPOSITIVO DE SOPORTE	COMPLEJIDAD	EXPERIENCIA DE USUARIO	LENGUAJE	APOYO ONLINE	CARACTERÍSTICAS DEL DISPOSITIVO
SMS	*TODOS*	*SIMPLE*	*LIMITADA*	*NINGUNO*	*NO*	*NO*
SITIOS WEB	*TODOS*	*SIMPLE*	*LIMITADA*	*HTML*	*NO*	*NO*
WIDGETS	*ALGUNOS*	*MEDIA*	*BUENA*	*HTML*	*LIMITADA*	*LIMITADA*
APLICACIONES WEB	*ALGUNOS*	*MEDIA*	*BUENA*	*HTML, CSS, JS*	*LIMITADA*	*LIMITADA*
APLICACIONES NATIVAS	*TODOS*	*ALTA*	*EXCELENTE*	*VARIOS*	*SI*	*SI*
JUEGOS	*TODOS*	*ALTA*	*EXCELENTE*	*VARIOS*	*SI*	*SI*

Tabla 2. Comparativa de tipos de aplicaciones móviles

2 - LA REALIDAD AUMENTADA

La Realidad Aumentada (RA) es una variación de la Realidad Virtual (RV). La tecnología de RV sumerge completamente al usuario dentro de un entorno virtual. Mientras está inmerso, el usuario no puede ver el mundo real que le rodea. Por el contrario, RA permite al usuario ver el mundo real, con objetos virtuales superpuestos o mezclados con el mundo real. Por lo tanto, se dice que RA complementa la realidad, en lugar de reemplazarla por completo. La RA ideal es cuando el usuario tiene la sensación de que los objetos virtuales y reales coexistieron en el mismo espacio.

La idea básica de la realidad aumentada consiste en superponer gráficos, audio y otras mejoras sensoriales en un entorno real en el mundo en tiempo real. El objetivo de la Realidad Aumentada (RA) es completar la información y el significado de un objeto o lugar del mundo real. La RA permite mediante el agregado de gráficos, comentarios de audio, datos de localización, contexto histórico, u otras formas de contenido hacer que la experiencia de un usuario sobre un objeto o lugar sea más significativa.

Actualmente hay dos definiciones comúnmente aceptadas sobre el término Realidad Aumentada: Uno de ellas fue dada por Ronald Azuma mediando el año 1997 [7]. La definición de Azuma dice que la realidad aumentada presenta básicamente tres elementos:

1. Combina lo real y lo virtual: La información digital es combinada con la realidad.
2. Funciona en tiempo real: La combinación de lo real y lo virtual se hace en tiempo real.
3. Esta registra en tres dimensiones: En general la información aumentada se localiza o "registra" en el espacio. Para conservar ilusión de ubicación real y virtual esta tiende a conservar su ubicación o moverse respecto a un punto de referencia en el mundo real.

Un poco antes, en 1994, P. Milgram y F. Kishino definieron la realidad de Milgram-Virtuality Continuum [8]. Estos describen como un continuo que abarca desde el entorno real a un entorno virtual puro. Entre medio hay Realidad Aumentada (más cerca del entorno real) y Virtualidad Aumentada (está más cerca del entorno virtual).

Figura 4. Milgram-Virtuality Continuum

El área entre los dos extremos, donde lo real y lo virtual se mezclan, es la llamada Realidad Mixta. A su vez, se dice que esta consiste tanto en la RA, donde aumenta el número de elementos reales y disminuye los virtuales, y la Realidad Virtual donde el crecimiento es a la inversa.

A diferencia de la RV, la RA no crea una simulación de la realidad; en su lugar, necesita un verdadero objeto o espacio como base para incorporar en él datos contextuales virtuales para profundizar la comprensión de una persona sobre un tema. Un sistema de RA genera una vista compuesta para el usuario.

Las tecnologías de RV y RA están muy relacionadas, pero existen algunas diferencias:

- La realidad virtual permite la inmersión del usuario en un mundo artificial que sustituye completamente al mundo real del usuario.
- La realidad aumentada mantiene el mundo real del usuario enriqueciéndolo con la presencia de elementos virtuales.

Los requerimientos generales para sistemas de RA se pueden resumir, comparándolos con los requisitos para sistemas de RV, para los tres subsistemas básicos que necesitan.

SUBSISTEMAS	REALIDAD VIRTUAL	REALIDAD AUMENTADA
GENERADOR DE ESCENAS	*MÁS AVANZADO*	*MENOS AVANZADO*
DISPOSITIVOS DE PANTALLA	*ALTA CALIDAD*	*BAJA CALIDAD*
SEGUIMIENTO Y DETECCIÓN	*MENOS AVANZADOS*	*MÁS AVANZADOS*

Tabla 3. Requerimientos de Realidad Aumentada y Realidad Virtual

La RA intenta crear la sensación de que los objetos virtuales están presentes en el mundo real. Para conseguir este efecto, RA fusiona el software RV con elementos del mundo real. Sabemos que la RA

es más eficaz cuando se agregan elementos virtuales en tiempo real. Debido a esto, RA comúnmente implica agregar objetos digitales a una imagen de vídeo en tiempo real. Este es el ejemplo más común de RA, denominado Realidad Aumentada Visual, que significa una superposición de imágenes 2D en vídeo digital. También es posible añadir objetos 3D a la imagen de video, pueden llegar a ser prestados de manera que parecen pertenecer a una escena que contiene objetos reales en 3D. La adición de un objeto 3D al video en tiempo real, es una de las demostración más impresionante de la tecnología RA.

2.1 - APLICACIONES

Los contextos de aplicación de RA van desde sistemas de navegación, a diseño de automóviles, desde medicina hasta entretenimiento y educación. Si bien la lista de aplicaciones es larga, exponemos los que hoy en día tienen más exploración y son viables a corto plazo:

- Turismo y Patrimonio: La RA permite, por ejemplo, visualizar monumentos o edificios en 3D, pudiendo ser recorridos e inspeccionados en detalle. También se puede aplicar en ambientes exteriores, mediante móviles, brindándole al usuario reconstrucciones virtuales de construcciones en ruinas, completando así los faltantes y mostrando en la pantalla el original en el entorno actual. También podemos añadir información adicional e interactiva a las propias construcciones. Por ejemplo, el dispositivo mostraría en la pantalla información adicional, imágenes del interior, audios explicativos, etc.
- Marketing: Mediante este tipo de sistemas, podríamos llevar cabo campañas publicitarias caracterizadas, interactivas e innovadoras.
- Educación: El campo educativo es un terreno cada vez más amplio para la RA. Se podría disponer de libros de texto como los que se usan hoy día, pero que sus páginas contengan marcadores concretos. Así, y a través del PC, se podría acceder vía Web a contenidos extra que proporciona la tecnología: videos documentales, modelos 3D de del cuerpo humano, volúmenes y superficies matemáticas, etc.
- Psiquiatría: Si disponemos de un sistema de visualización subjetiva, la RA puede crear cualquier tipo de elemento alrededor del usuario de forma controlada.
- Entretenimiento: Lo más popular de este ámbito son los videojuegos. Ya se pueden disfrutar de varios que se comercializan actualmente en el mercado. A través de la televisión podemos ver como mascotas virtuales recorren el salón de casa.

La RA también podría combinarse con espectáculos de música o baile, en los cuales los actores y bailarines interactuaran con creaciones virtuales acordes con las coreografías, enriqueciendo la experiencia al verlo a través de una pantalla.

También podríamos encontrar aplicaciones en el sector de la industria, en la presentación de prototipos o proyectos de construcción, simulaciones,… y en definitiva casi en cualquier sector que se nos ocurra. Esta gran versatilidad es lo que hace de la RA una de las apuestas más importantes en la innovación de los sistemas de visualización.

Además de agregar objetos a un entorno real, la RA también tiene el potencial para eliminarlos. El trabajo actual se ha centrado en la adición de objetos virtuales a un entorno real. Sin embargo, se pueden utilizar superposiciones gráficas para quitar u ocultar partes del entorno real que ve el usuario. Este tipo de RA se ha utilizado en el cine, por ejemplo. Hacer esto de forma interactiva en un sistema de RA significará más complejidad, pero para esta eliminación de escena en video a tiempo real, no es necesario ser demasiado realista para ser eficaz.

Hasta ahora, vemos que los investigadores se han centrado en la mezcla imágenes reales y virtuales y gráficos. La RA podría aplicarse a todos los sentidos, no sólo a la vista. Veamos el caso de que RA podría ampliarse para incluir sonido: Un ejemplo podría ser que el usuario utilice audífonos equipados con micrófonos para el exterior. Los auriculares añadirían sonido sintético, mientras que los micrófonos externos podrían detectar sonidos entrantes del entorno. Esto daría al sistema una oportunidad para

ocultar o encubrir sonidos reales. Si bien esto no sería tarea fácil, podría ser posible. Otro sentido que podría aumentarse es el tacto: guantes con dispositivos que proporcionan retroalimentación táctil puede aumentar las sensaciones virtuales en el medio ambiente.

La RA es identificada como una de las diez primeras nuevas tecnologías más importantes para el período 2008-2012 por Gartner Research [9] y se espera que sea utilizado por más del 30% de la fuerza de trabajo móvil para el año 2014. Gartner define la RA como una tecnología innovadora que causa un cambio importante en la forma aceptada de hacer las cosas, incluyendo modelos de negocio, procesos, flujos de ingresos, la dinámica de la industria y el comportamiento del consumidor.

2.2 - COMO FUNCIONA

Una amplia gama de tecnologías pueden ser utilizadas para implementar RA. Muchos de los primeros proyectos incluían un casco o un dispositivo similar del campo de la visión, lo que corresponde con un objeto real o el espacio que el usuario está observando. Los dispositivos portátiles pueden utilizar las señales GPS para ofrecer a los usuarios datos adicionales del contexto, agregando medios audiovisuales, o datos basados en texto acerca de objetos o lugares.

La RA no es más que un archivo de texto o multimedia, pero también es una tecnología diseñada para "ver" un verdadero objeto o lugar y proporcionar al usuario información adecuada en el momento adecuado. Se ha diseñado para difuminar la línea entre la realidad que atraviesa el usuario y el contenido que ofrece la tecnología.

En el año 2002, Brügge presentó un resumen de las arquitecturas de los mayores sistemas de software de RA [10]. Este estudio incluyó 18 arquitecturas de donde se extrajo una arquitectura de referencia que contiene los componentes comunes de tales sistemas. El diagrama de componentes y dependencias es el siguiente:

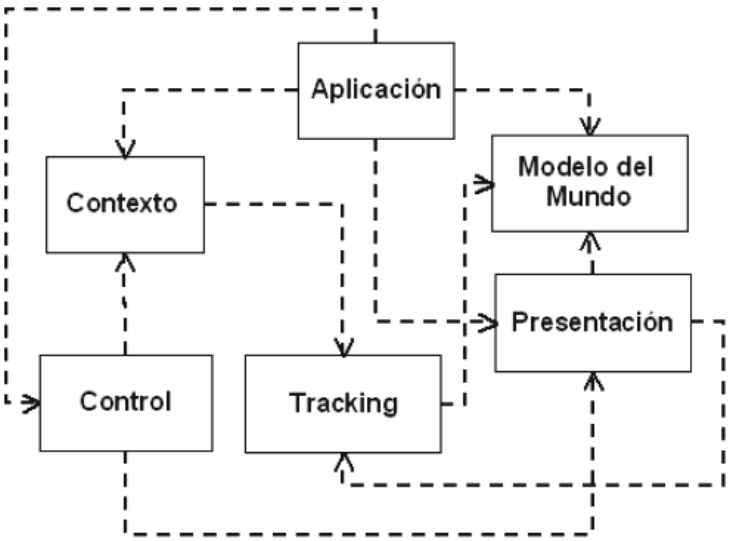


Figura 5. Arquitectura de Brügge

Definimos brevemente cada componente a continuación.

- Aplicación: maneja la lógica y contenidos específicos del sistema.
- Tracking: determina la posición de los usuarios y objetos.
- Control: procesa las entradas para el usuario.
- Presentación: se encarga de la representación gráfica.
- Contexto: recoge diferentes datos de contexto.
- Modelo del Mundo: almacena información sobre los objetos virtuales y reales.

Para poder añadir información a una imagen es necesario saber qué hay en esa imagen. El ordenador puede intentar reconocer formas en la imagen capturada por la cámara (hoy en día el ordenador de un avión puede identificar otros aviones por la temperatura de sus motores, en este caso usando una cámara de infrarrojos) o puede presuponer lo que hay en la imagen a partir de las coordenadas y la dirección de la cámara. Esta última opción es la que usan la mayoría de las aplicaciones que se han hecho populares últimamente.

La RA es una tecnología utilizada desde hace años para usos tan diversos como marcar un fuera de juego en la televisación de partidos de fútbol, superponer esquemas eléctricos sobre los circuitos reales o mostrar a los pilotos de combate información sobre los objetivos que ven a través de su casco.

Estas primeras aplicaciones nombradas anteriormente nos permiten distinguir los dos tipos básicos de realidad aumentada:

- La basada en ciertos marcadores presentes en la escena a mostrar.
- La basada en la localización de la escena real a mostrar y la orientación de la cámara.

2.2.1 - Realidad Aumentada basada en marcadores

La RA basada en marcadores encarna el trabajo de reconocer patrones, como puede ser un código de barras, QR(*) o un símbolo, en la imagen de video que se recibe desde una cámara. Cuando se reconoce un patrón en particular, en su posición se superpone una imagen digital en la pantalla.

Este es el primer tipo de realidad aumentada modera, que tuvo sus orígenes en algo muy sencillo: etiquetas.

Código de barra 2D o QR

El proceso de formación del objeto virtual en RA basada en marcadores es el siguiente:

1. La cámara filma el mundo real y manda las imágenes, en tiempo real, para que el software analice y determine la presencia de patrones para incluir objetos virtuales.
2. El software estará programado para crear determinados objetos virtuales dependiendo de la concordancia con patrones en la imagen que fue tomada por la cámara.
3. El aparato de salida, que puede ser un televisor o un monitor de computadora, o la pantalla de dispositivo móvil, exhibe el objeto virtual en sobreposición con el real, como si ambos fueran la misma escena.
4. Comúnmente, en muchas aplicaciones actuales, a los objetos virtuales se le agregan elementos para interactuar con ellos.

La complejidad en este tipo de RA suele ser el software, que mediante algoritmos de reconocimiento de imágenes debe determinar los patrones en la imagen captada. Para agilizar el proceso y permitir la interactividad, la cual requiere de gráficos en tiempo real, es conveniente que la correspondencia entre patrones, rasgos del contexto, y posición tridimensional y la perspectiva de la impresión de los objetos virtuales, sea preparada con anticipación. Esto es, debemos contar con una base de datos y un algoritmo eficiente para evitar muchos cálculos en tiempo de ejecución.

2.2.2 - Realidad Aumentada basada en localización

Sabemos que una de las maneras más populares en que las personas interactúan con la RA es a través de aplicaciones móviles basadas en localización. Con solo sostener un teléfono en nuestro campo de visión, podemos visualizar puntos de interés cercanos gracias a la información tomada a través de varios sensores del teléfono, entre ellos el GPS y brújula de orientación. Debido a las limitaciones del GPS, sin embargo, este tipo de experiencia es más difícil de reproducir en un espacio interior que en el exterior.

Es importante señalar que la orientación en este tipo de RA no trabaja con los objetos captados de la cámara para identificar su naturaleza. La dificultad para todo se reduce a la adquisición de tres dimensiones LLA (estructura básica compuesta por: latitud, longitud y altitud) y su comparación y alineación con las referencias que el sistema tenga almacenadas. Lo que necesitamos es una técnica que rápidamente puede comparar la posición y orientación actual con las estructuras almacenadas, para así minimizar la latencia.

2.3 - ARQUITECTURA DE SISTEMAS RA

Podemos expresar hasta ahora, que en cualquier sistema de realidad aumentada se requieren dispositivos que identifiquen el escenario real y lo clasifiquen así como que visualicen tanto entorno real como información digital.

La tecnología RA aprovecha las tecnologías derivadas de la visualización para construir aplicaciones y contenidos con las cualidades que estas áreas han madurado en las últimas décadas. Del procesamiento de imágenes toma la cualidad de resaltar aspectos en las imágenes captadas por la cámara de video (posiciones tridimensionales, patrones fiduciarios para el reconocimiento, etc.). De los gráficos por computadora toma la síntesis de objetos tridimensionales y sus transformaciones.

En todo sistema de RA son necesarias cuatro tareas principales para poder llevar a cabo el "aumento" de la realidad. Explicaremos estas tareas se con mayor detalle posteriormente, pero básicamente son [11]: (1) Captura de contexto; (2) Identificación de contexto; (3) Aumento de la realidad; y (4) Visualización.

Figura 6. Tareas principales de Realidad Aumentada

Describimos a continuación las cuatro tareas necesarias que están presentes en cualquier sistema de RA.

2.3.1 - Captura de contexto

Una de las tareas más significativas en todo sistema de RA es la de identificar la escena que se quiere aumentar. En el caso de los sistemas que utilicen reconocimiento visual, es indispensable disponer de algún mecanismo que permite tomar la escena para luego ser procesada. Entonces general inicia con el registro de las señales del mundo real (generalmente video, aunque bien pudiera ser audio).

Los dispositivos de captura de imágenes son dispositivos físicos que recogen la realidad para luego esta poder ser aumentada. Básicamente, estos dispositivos se agrupan en dos grupos:

- Dispositivos see-through: estos dispositivos realizan simultáneamente la tarea de capturar la escena real como la de mostrar la información aumentada al usuario. Estos dispositivos acostumbran a trabajar en tiempo real, haciéndolos no sólo más costosos en presupuesto sino también en complejidad. Dentro de este grupo encontramos aquellos dispositivos conocidos como head-mounted. Sabemos que estos dispositivos see-through llevan años siendo utilizados, por ejemplo, en los Head Up Displays (HUDs) utilizados por los aviones de combate para visualizar información sobre altura, velocidad, identificación de objetivos, y otros datos sin necesidad de retirar la vista de la zona frontal de la cabina.
- Dispositivos video-through: dentro de este grupo encontramos los dispositivos donde el modulo que realiza la captura de video es independiente al modulo de visualización. En este se encontrarían las cámaras de video o los terminales móviles.

2.3.2 - Identificación de contexto

Aparte de la tecnología de captura de imagen, el más importante desafío tecnológico en RA en general es el seguimiento y registro.

Los sistemas AR requieren mucha precisión en la posición exacta y el seguimiento de la orientación para la alineación, o registro, de información virtual con los objetos reales que deben ser aumentados. En el caso de dispositivos móviles en general, no se puede esperar contar con el seguimiento de cualquier tipo de las infraestructuras en el medio ambiente. En estas circunstancias, no existe actualmente una solución de seguimiento perfecto, ni podemos esperar encontrarla en un futuro próximo. Aun así, la tecnología de seguimiento ha mejorado constantemente desde los primeros días de la RA.

2.3.2.1 - Seguimiento

El seguimiento se puede lograr con una variedad de diferentes tecnologías que se basan en diferentes principios físicos: Mecánico, magnético, los enfoques de seguimiento acústicos y ópticos. No todas las aplicaciones de RA requieren un seguimiento preciso. Las aplicaciones modernas, como las que emplean los teléfonos celulares, sólo necesita preocuparse por la alineación de contenido virtual con la captura de imagen desde la cámara de la escena.

Los dispositivos empleados para enfrentar este problema son:

- Sensores: Se hace uso de dispositivos de hardware de corto alcance que permiten determinar una ubicación en el espacio. Existen sensores ópticos (infrarrojos), magnéticos y de movimiento.
- Visión de máquina: Se hace uso de una combinación de hardware y software para identificar objetos o patrones y calcular su distancia respecto al observador. Por ejemplo se puede usar una cámara de video para identificar un libro real y dibujar sobre éste un texto virtual.
- sistemas a campo abierto: Para aplicaciones de realidad aumentada en campo abierto se usa el Sistema de Posicionamiento Global (GPS) para determinar la posición del usuario.

2.3.2.2 - Método 6DOF

Un método común, ampliamente utilizado, consiste en determinar la posición del en algún sistema de coordenadas mundial, y se lo refiere a un modelo informático del entorno actual. La determinación de la posición y orientación de un objeto se refiere a menudo como una posición de "seis grados de libertad" (6DOF, del acrónimo "Six degrees of freedom") de seguimiento, para los seis parámetros detectados: la posición en X, Y, Z, y la orientación angular de guiñada, cabeceo y alabeo.

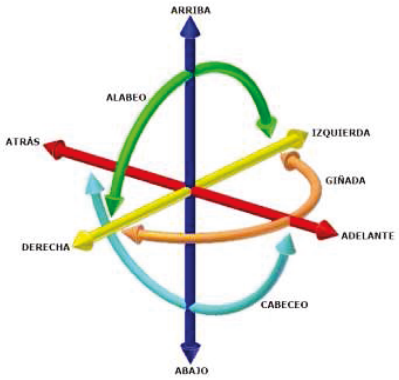
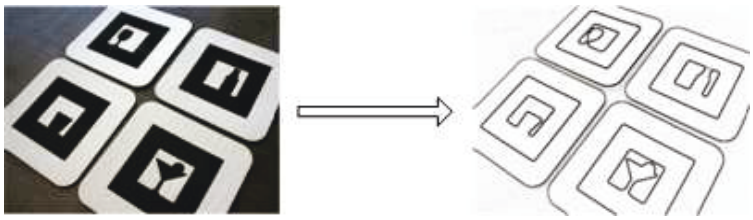


Figura 7. Movimientos en 6DOF

Antes de describir las técnicas de posicionamiento y seguimiento, indicamos que no existe una técnica general para todos los sistemas de realidad aumentada, sino que los requisitos dependen estrictamente de las necesidades del sistema.

La posición absoluta y la orientación de la visión del usuario y los objetos físicos no necesariamente tienen que ser conocidas. En uno de los más directos enfoques de la identificación del contexto, se analiza la captura de video en busca de puntos de referencia específicos (por ejemplo, marcadores artificiales) en el medio. Si se detectan dichos marcadores en la imagen, las anotaciones virtuales pueden ser directamente insertadas en coordenadas de píxeles sin tener que establecer la exacta relación geométrica entre el marcador y la cámara. La utilización de marcadores ha sido en muchas ocasiones el enfoque más clásico a fin de saber con exactitud la posición de la cámara.

En primer lugar los puntos de interés, o marcadores fiduciarios se detectan desde las imágenes de la cámara. Esta etapa utiliza algoritmos populares en el tratamiento de imágenes(*), como Corner detection, Blob detection o Thresholding u otros filtros de procesamiento para imágenes que nos permitan detectar patrones en ella.



Filtro Corner detection sobre imagen con marcadores 2D.

Si no utilizamos marcadores para reconocer el contexto, el uso de dispositivos externos como el GPS y la brújula digital resultan indispensables con tal de que los objetos de la escena guarden una coherencia visual para quien esté visualizando la escena captada en la captura de vídeo. Si la ubicación exacta de los puntos de referencia en el medio ambiente son conocidos, las técnicas de visión por computador se pueden utilizar para estimar lo que cámara esta capturando.

Con el uso de dispositivos con cámaras montadas junto a la pantalla, el reconocimiento se refiere a veces como "seguimiento de circuito cerrado", en los que la exactitud del seguimiento se puede corregir con una precisión de píxeles, si la imagen de la cámara y la pantalla de gráficos coinciden. Esto está en contraste con el seguimiento a campo abierto basado en localización, que trata de alinear las anotaciones virtuales con los objetos físicos en el mundo real basándose únicamente en la actitud 6DOF detectada de la persona y el modelo informático que se conoce del medio ambiente.

Cualquier error en el dispositivo de localización o del modelo geométrico hará que la anotación este ligeramente desviada de su posición prevista en relación con la física mundo. La precisión en el seguimiento necesario para RA depende mucho de la aplicación y de la distancia a los objetos detectados. Si estamos registrando datos de edificios a la distancia, podemos permitirse un error de registro. Cuando se trata de determinar la ubicación exacta de una ventana en particular, tenemos que ser más precisos. Dado que no existen sensores independientes para determinar coordenadas 6DOF confiables en ambientes al aire libre, los sistemas móviles de RA normalmente recurren a métodos híbridos, que emplean mecanismos independientes para determinar la posición y el seguimiento de la orientación. El seguimiento de la posición vía GPS es un candidato natural para los ambientes al aire libre, ya que es funcional a nivel mundial, siempre y cuando se este en ambientes a cielo abierto.

2.3.2.3 - Técnicas de seguimiento hibridas

Existen otros tipos de técnicas para determinar el seguimiento de la posición. Las mencionamos a continuación:

- Existe una tecnología basada la re red satelital GPS, que ciertamente es una mejora de esta, el GPS asistido - conocido A-GPS(*). La tecnología se las arregla para eludir la restricción de las zonas que no permiten una visibilidad directa con los satélites. A-GPS utiliza una referencia a red mundial de estaciones de base para la emisión de la señal terrestre. En combinación con un gran número paralelos de circuitos correlación la recepción del receptor GPS móvil puede extender a mucho más allá de áreas descubiertas, tales como cañones urbanos y ambientes interiores en los que la señal sea lo suficientemente fuerte.
- La red de telefonía móvil permite determinar, con cierto grado de precisión, la posición de cualquier dispositivo móvil conectado a la red GSM. El funcionamiento es muy sencillo, ya que básicamente se trata del establecimiento de una triangulación del teléfono celular en base a su posición actual en el planeta. La tecnología es eficiente y esta disponible en cualquier lugar cubierto por servicio GSM, aunque su precisión no lo hace confiable, ya que varía entre 50 - 500 metros y en áreas rurales el error puede llegar a los 2000 metros.
- Otro sistema de seguimiento de la posición en áreas amplias consiste en calcular la ubicación de un dispositivo a partir de la calidad de la señal de red inalámbrica 802.11b(*), Wi-Fi. Obviamente, esto también requiere el despliegue de puntos de acceso en el medio ambiente. Pero en interiores o contexto sin señal GPS, este sistema de posicionamiento puede servir como un beneficio adicional. La resolución a alcanzar depende de la densidad de puntos de acceso desplegados para formar la red inalámbrica.
- Dos medios adicionales para determinar la posición, a menudo son empleados en el marco de los sistemas de seguimiento de híbridos: los sensores inerciales y la visión basada en

enfoques. Acelerómetros y giroscopios son sensores inerciales autónomos o sin origen. Su principal problema es la deriva, por lo tanto, en la práctica, esta aproximación a la estimación de la posición sólo puede ser empleada por periodos de tiempo muy pequeños (generalmente, entre las actualizaciones obtenida de una fuente más fiable).

- En investigación, existen modelos basados en técnicas visión. Estos requieren un modelo preciso del contexto con puntos de referencia conocidos que se pueden reconocer en la imagen de video que se captura. La reconstrucción simultánea del movimiento de la cámara y la geometría de la escena es posible, pero tales cálculos son muy computacionalmente muy costosos.
- Otro método poco extendido es la tecnología de identificación por radio frecuencia. A través de lectores RFID(*) en los dispositivos y marcas colocadas en el medio ambiente.
- Otra alternativa, más sencilla y que no requiere hardware especial, es recuperar el contenido a través de las cámaras en los dispositivos. Un ejemplo sencillo sobre este tipo de sistema de seguimiento son los llamados códigos QR. Aunque poco utilizados, son una buena alternativa para seguimiento en interiores.

El seguimiento de orientación también se beneficia en gran medida de los enfoques híbridos. La base tecnologías disponibles para la orientación de detección son brújulas electromagnéticas (Detectores de metales), sensores de inclinación gravitacional (inclinómetros), y giroscopios (Mecánicos y ópticos). Por el momento, los híbridos de seguimiento basado en visión y otras tecnologías de detección mostrar el resultado de la mayor promesa. Estas soluciones híbridas se han desarrollado tanto en lo que productos comerciales como en prototipos de investigación.

En resumen, el problema del seguimiento de una persona para aplicaciones móviles de RA en general es un problema difícil sin una única y mejor solución. Los enfoques de seguimiento híbridos son actualmente la forma más prometedora para hacer frente a las dificultades que plantea por lo general la RA en ambientes interiores y exteriores.

2.3.3 - Aumento de la realidad

Una vez que se haya identificado el contexto a aumentar, la siguiente tarea que tiene lugar en los sistemas de RA es de superponer la información virtual sobre la escena real capturada. Como mencionamos anteriormente, esta información u objetos virtuales de aumento pueden ser tanto de tipo visual como auditivo o táctil, lo que por lo general, en la mayoría de sistemas de RA sólo ofrecen tipos de aumentos visuales. Podríamos definir esto como el segundo de los requerimientos prioritarios de una aplicación RA: el registro de su contenido virtual en relación con los objetos del mundo real.

En aplicaciones de RA visual, el primer concepto que hay que diferenciar es el tipo de objetos que se quiere agregar. Aquí podemos distinguir entre dos tipos básicos de información: 2D y 3D. En los sistemas de RA, excepto en aquellos que utilizan hologramas tridimensionales o técnicas similares, los dispositivos de visualización son de dos dimensiones (pantallas de ordenadores, teléfonos móviles, etc.). Esta limitación nos puede hacer pensar que sólo es posible representar información en 2D y, aunque esto es cierto, es posible simular la sensación de tridimensionalidad en un plano 2D. Para realizar la conversión de una imagen en 3D al plano bidimensional se suele utilizar la técnica de proyección de perspectiva (o proyección de puntos). Esta consiste en simular la forma en que recibimos la información visual por medio de la luz y cómo nos puede dar la sensación 3D. Este proceso consiste en la colocación de dos imágenes bidimensionales captadas desde distintos ángulos, dando la sensación de profundidad (que en realidad no existe) en imágenes 2D.

Retomemos la principal actividad de esta tarea: aumentar la realidad significa sintetizar y representar en tiempo real cualquier objeto dentro del sistema de RA. Para ello, podemos hacer uso de algún motor de representación grafica especializado para RA o incluso si se quiere tener un mayor control sobre el registro, utilizar alguna librería grafica de bajo nivel. Estos motores gráficos y librerías varían en cuanto a que plataforma móvil estemos hablando, aunque algunas están disponibles en varias de las más populares.

A continuación detallaremos algunas de los motores de representación más populares para RA.

ARToolkit

Es un motor que nos ayuda en la creación de aplicaciones de RA basada en marcadores. Utiliza los

parámetros de seguimiento de vídeo, con el fin de calcular la posición de la cámara y la orientación relativa a los marcadores detectados. Una vez que la posición de la cámara se sabe, y en presencia de un marcador reconocido, modelos 2D o 3D son superpuestos exactamente sobre el marcador real.

Existen varias variantes de ARToolkit, y esta portada a varias plataformas (entre ellas Android) que lo hacen el motor más popular en su especialidad.

iPhone ARKit

Es una biblioteca de interfaz de usuario para la visualización de los datos de localización basados en sistemas de coordenadas esféricas utilizando la interfaz de usuario de la SDK de iPhone. El tipo de realidad aumentada que abastece a ARKit es la superposición de la información a través de una vista de cámara.

Existe también una versión portada a Android, denominada AndroidARKit.

La diferencia entre un motor de representación y una librería grafica son las facilidades que brindan. Una librería grafica sólo se encarga de la representación en sí, una "Enghien" proporciona una serie de utilidades adicionales que podrían resultar deseables. Estas utilidades tienen que ver con la carga, conversión, optimización y texturizado de objetos 3D.

Detallaremos a continuación algunas de las librerías graficas más populares para:

OpenGL

OpenGL es un estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones para producir y consumir gráficos en 2D y 3D. La librería sustenta la capacidad gráfica de los dispositivos. OpenGL maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D.

Existe una variante simplificada, OpenGL ES, que esta disponible para varias de las plataformas móviles, incluyendo Symbian, Android e iPhone.

WMGL

WMGL es una biblioteca gráfica desarrollada para dispositivos con Windows Phone. Proporciona fácil utilización de interfaces para la mezcla y representación, además de varios efectos 3D.

Como contrapartida a los beneficios que obtenemos al utilizar una librería grafica, la integración de esta en un sistema de RA requiere un trato especial, ya que los dispositivos utilizados en estos sistemas tienen ciertas limitaciones de tamaño y capacidad de proceso, impuestas por la necesidad de que sean fácilmente transportables y de una alimentación independiente. Las librerías gráficos ofrecen características de representación avanzadas que suelen requerir alta capacidad de proceso, tanto de la CPU como de la GPU(*), características que no siempre corresponden con las restricciones de hardware de los dispositivos portátiles.

Con las aplicaciones que requieren un registro exacto, la información tiene que ser encontrada en profundidad en el mundo real con el fin de llevar a cabo la calibración necesaria de los entornos reales y agregados. Sin un conocimiento preciso de la geometría tanto del mundo real y la escena generada por ordenador, el registro exacto es prácticamente imposible. Para alinear correctamente videos e imágenes virtuales, varios marcos de referencia deben ser considerados.

Lorensen ha discutido los temas de calibración y registro de la imagen en el contexto en diferentes marcos de referencia para la RA. En el ejemplo de RA para médicos de Lorensen [12], se necesitaban dos sistemas de coordenadas: un sistema de coordenadas del mundo real y un sistema de coordenadas del mundo virtual. Con estos dos sistemas, la alineación del video y las imágenes generadas por computadora se realiza de forma manual. Lorensen señaló también que este procedimiento ha funcionado bien cuando las características anatómicas de la paciente eran fácilmente visibles.

A pesar de estas metodologías de trabajo que funcionan razonablemente bien para los sistemas que tienen un rango limitado de movilidad, hace falta bastante investigación que hacerse antes de que el registro de las imágenes mediante los sistemas portátiles pueda ser realmente preciso.

2.3.3.1 - Problemas de registro

Uno de los problemas básicos que actualmente limita a las aplicaciones de RA es el problema de

registro. Los objetos virtuales insertados debe estar alineados con el mundo real, o la ilusión de que coexisten dos mundos se verá comprometida. Muchas aplicaciones tienen demanda de registro exacto. Por ejemplo, aplicaciones en el ámbito de la medicina.

Como un medio interactivo, en la RA es difícil trabajar con estos errores, pues no se puede controlar los movimientos del usuario. El usuario se mueve cuando quiere, y el sistema debe responder dentro de decenas de milisegundos. Los errores de registro son difíciles de controlar de forma adecuada debido a los requisitos de alta precisión y las numerosas fuentes de error. Estas fuentes de error se puede dividir en dos tipos: estática y dinámica.

Los errores estáticos son los que causan los errores de registro, incluso cuando el usuario y los objetos en el contexto permanecen completamente inmóviles. Los errores dinámicos son los que no tienen efecto hasta que el punto de vista o los objetos comienzan a moverse. Para los actuales sistemas de RA, los errores dinámicos son, por mucho, los mayores contribuyentes a los errores de registro, pero tampoco podemos ignorar los errores estáticos.

Las principales fuentes de errores estáticos son: Distorsión óptica, errores en el sistema de seguimiento, y parámetros de visualización incorrectos (campo de visión, la posición de seguimiento a los ojos y la orientación). Los errores dinámicos se producen debido a los retrasos del sistema, o tildes. El retraso del sistema de se define como la diferencia de tiempo entre el momento en que se obtienen las medidas del sistema de seguimiento y orientación del punto de vista hasta el momento en que las imágenes generadas correspondiente a la posición y orientación aparecen en pantalla. Estos retrasos existen porque cada componente de un sistema de RA requiere cierto tiempo para hacer su trabajo. Retrasos de 100ms son bastante típicos en los sistemas existentes, aunque sistemas más simples pueden tener menos retraso.

2.3.4 - Visualización

Nos referimos a visualización como el resultado de combinar objetos reales y virtuales y como se le presentan al usuario. Dividiremos esta sección de dos partes, distinguiendo los dispositivos empleados para la visualización para luego terminar describiendo los diferentes modos de presentar la información en el dispositivo de salida, tarea que se le atribuye la mayor parte del trabajo al software del sistema.

Actualmente la RA visual se logra con el uso de dispositivos de visualización similares a los de RV. Existen tres técnicas principales para implementar la visualización en RA:

1. Head Mounted Displays (HDM)
2. Handheld Displays
3. Spatial Displays

2.3.4.1 - Head Mounted Displays

Un Head-mounted Display o HMD es un dispositivo de visualización similar a un casco, que permite reproducir imágenes creadas por ordenador sobre un "display" muy cercano a los ojos. Son en su mayoría empleados para permitir al usuario ver el mundo real donde objetos virtuales se sobreponen a los medios ópticos o tecnologías de vídeo. Fundamentalmente es dividida en dos categorías: "ver a través de óptica" (OST) y "ver a través de video" (VST):

- OST son los que permiten al usuario a ver el mundo real con sus ojos y las superposiciones virtuales se realizan mediante el uso de un elemento óptico holográfico o tecnología similar. La principal ventaja de las pantallas OST es que ofrecen una vista superior de la escena real, incluyendo una vista natural e instantánea de la escena real.
- VST son aquellos en los que el usuario tiene una visión de vídeo del mundo real con gráficos superpuestos sobre la imagen en pantalla. Las ventajas de VST incluyen la coherencia entre los puntos de vista real y virtual, y la disponibilidad de técnicas de procesamiento de imagen como la corrección de la intensidad y el matiz, y permite el control de relación de mezcla [13]. Por lo tanto, VST puede manejar problemas de oclusión más fácilmente en comparación con OST debido a las varias técnicas de procesamiento de imágenes.

2.3.4.2 - Handheld Displays

Los dispositivos de mano son una buena alternativa a la HDM para aplicaciones de RA, en particular debido a que son mínimamente invasivos, socialmente aceptables, están fácilmente disponibles y son muy móviles. En la actualidad, existen varios tipos de dispositivos de mano que pueden ser utilizados para una plataforma móvil de RA: Tablet PCs, netbooks, celulares (teléfonos inteligentes y PDA. Dispone de sensores portátiles como brújulas digitales y GPS para las unidades de seguimiento.

2.3.4.3 - Spatial Displays

En lugar de que el usuario use o lleve consigo la pantalla como HMD o dispositivos de mano, la realidad aumentada espacial (SAR), hace uso de proyectores digitales para mostrar información gráfica sobre los objetos físicos. La diferencia clave en el SAR es que la pantalla está separada de los usuarios del sistema. Debido a que la mezcla no está asociada con cada usuario, naturalmente permite la colaboración entre los usuarios. Esto lo hace un buen candidato para el trabajo colaborativo en aplicaciones de RA, ya que un sistema puede ser utilizado por varios usuarios al mismo tiempo, sin tener que utilizar estos dispositivos independientes.

Resumiendo

Cada dispositivo de visualización tiene sus ventajas y desventajas y cada uno se encuadra en los distintos tipos de aplicaciones de RA.

Sabemos que los HMD aumentan la experiencia de inversión para el usuario y brinda al usuario una libertad adicional al no ocupar sus manos, pero el tamaño físico y peso pueden resultar incómodos en condiciones de trabajo. Con dispositivos Handheld, los usuarios tienen su propio sistema de E/S con ellos, donde no hay necesidad de compartir un equipo central, pero dispositivo puede interferir con la agilidad del usuario al tener que sostenerlo con la mano. En los dispositivos Spatial no se requiere equipo adicional para interactuar y las dos manos están libres y generalmente en las aplicaciones el seguimiento de los usuarios no es necesario, pero solo un número limitado de usuarios pueden interactuar con el sistema ya demasiadas líneas podría ser una gran distracción.

No es menos importante mencionar que la utilización de dispositivos HMD o Spatials en aplicaciones de realidad aumentada involucra altos costos para su implementación. No así el caso de los dispositivos Handheld, que son básicamente (o están compuestos) del hardware habitual y económico que nos cruzamos a diario.

2.3.5 - Dispositivos de entrada

En relación con los dispositivos de salida, son necesarios dispositivos de entrada efectivos para permitir al usuario interactuar sin problemas con el texto o las imágenes virtuales que se le presentan. Los dispositivos de entrada que se han desarrollado para el uso con los sistemas informáticos portátiles son muy diversos y cambiantes para dar cabida a las necesidades del usuario. El reciente crecimiento en la popularidad de los dispositivos portátiles ha provocado un creciente interés en el diseño y evaluación de los dispositivos de entrada.

En el entorno del mundo real, el usuario esta a menudo usando una o ambas manos para realizar una tarea, por lo tanto, los dispositivos de entrada utilizados en ordenadores portátiles deben ser diseñados con este requisito en mente. Para la entrada de datos o la introducción de texto, teclados montados sobre el cuerpo, software de reconocimiento de voz, o teclados portátiles son de uso frecuente.

Los dispositivos de entrada como teclados portátiles, teclados en la muñeca, o incluso los métodos de entrada de los teléfonos celulares de hoy día son eficientes para el usuario cuando no es requisito para la tarea el disponer de las manos libres.

2.4 - HISTORIA

El concepto de Realidad Aumentada no es nuevo, y se remonta varios años atrás con la masificación de las tarjetas de gráficos y la llegada de las cámaras Web. Desde hace algún tiempo, incluso algunas revistas y publicaciones impresas colocaron anuncios en los cuales se invitaba al lector a colocar la revista o el documento cerca de una cámara Web en una PC para visualizar algún contenido virtual.

En 1950 Morton Heilig escribió sobre un "Cine de Experiencia", que pudiera acompañar a todos los sentidos de una manera efectiva integrando al espectador con la actividad en la pantalla. Construyo un prototipo llamado el Sensorama en 1962, junto con 5 filmes cortos que permitían aumentar la experiencia del espectador a través de sus sentidos (vista, olfato, tacto y oído).

Sensorama

Es necesario que demos algo a entender antes de seguir: la Realidad Virtual y Realidad Aumentada han ido prácticamente de la mano.

En 1968, Ivan Sutherland, con la ayuda de Bob Sproull, construyeron lo que sería ampliamente considerado el primer visor de tecnología Head Mounted Display (HMD) para Realidad Virtual y Realidad Aumentada [14]. Era muy primitivo en términos de Interfaz de usuario y realismo, y el HMD usado por el usuario era tan grande y pesado que debía colgarse del techo, y los gráficos que hacían al ambiente virtual eran simples "modelos de alambres". A finales de los 80 se popularizo el término Realidad Virtual por Jaron Lanier, cuya compañía fundada por él creo los primeros guantes y anteojos de Realidad Virtual.

HMD

El termino Realidad Aumentada fue introducido por el investigador Tom Caudell en Boeing, en 1992. Caudell fue contratado para encontrar una alternativa a los tediosos tableros de configuración de cables que son utilizados en algunas industrias. Apareció con la idea de anteojos especiales y tableros virtuales sobre tableros reales genéricos, es así que se le ocurrió que estaba "aumentando" la realidad del usuario. El término Realidad Aumentada fue dado al público en una publicación en 1992.

En 1972, Myron Krueger crea "Videoplace", que permite a los usuarios interactuar con objetos virtuales por primera vez.

Pasaron dos décadas para que en 1992, Tom Caudell y David Mizell introdujeran el termino "Realidad Aumentada" por primera vez [15]. Mientras tanto, la RA en Boeing (empresa constructora de aviones) ayuda a los trabajadores montar los cables en los aviones.

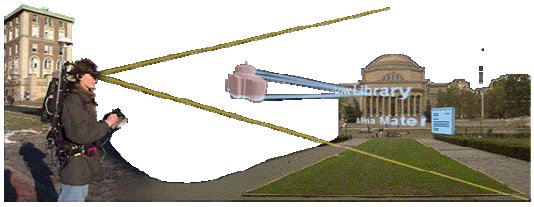
Ese mismo año, Caudell Mizell en una publicación discute las ventajas de la RA en comparación con RV, al requerir menos potencia de procesamiento de imágenes, ya que son menos las que tienen que ser prestados en pantalla, aunque reconoce los mayores requisitos de registro de estas, a fin de alinear reales y virtuales.

En 1994, Steve Mann comienza a utilizar una cámara Web en su rutina diaria y lo hace durante casi 2 años. Desde 1994 al 1996, Mann llevó una cámara portátil más una pantalla todo el día. Ambos equipos estaban conectados a su sitio Web donde se permitía a los visitantes en línea ver lo que Steve estaba viendo y le enviaban mensajes que aparecían en la pantalla de su móvil.

Paul Milgram y Fumio Kishino escribían su artículo "Taxonomía de exhibición de la realidad virtual mixta" en el que definen la Milgram-Virtuality Continuum descripta anteriormente. Milgram y Kishino describir un continuo que va desde el entorno real al entorno virtual. El continuo virtual de Milgram y la definición de RA de Azuma (1997) son comúnmente aceptadas como la definición de la Realidad Aumentada.

En 1996, Junio Rekimoto presenta los marcadores de matriz 2D en forma de códigos de barras cuadrados (conocidos como códigos QR), uno de los primeros sistemas de marcador para permitir el seguimiento de la cámara con seis grados de libertad.

En 1997, mientras que Azuma proporcionaba una definición ampliamente reconocida de AR, Steve Feiner crea el primer sistema de realidad aumentada móvil denominado MARS (Mobile Augmented Reality System) [16].



MARS

En 1999, Hirokazu Kato y Mark Billinghurst presentan ARToolKit, una biblioteca de software para la creación de aplicaciones de Realidad Aumentada, utilizando un enfoque basado en plantilla para reconocimiento y posicionamiento de objetos virtuales.

También en el 99', Jim Spohrer publica el concepto "Worldboard", una infraestructura escalable para soportar aplicaciones móviles que van desde los servicios de gama baja basados en la localización, hasta la RA en móviles de alta gama.

En 2000, Bruce Thomás presente AR-Quake, una extensión del popular juego Quake escritorio. ARQuake es una aplicación de la perspectiva en primera persona que se basa en un sistema de seguimiento 6DOF a través de GPS, una brújula digital y el seguimiento de la visión basada en los marcadores fiduciales.

El año 2001, trajo la presentación de AR-PDA por Jürgen Fruend. Un concepto para la construcción de un sistema de RA inalámbrica que incluía un prototipo especial de hardware del tamaño de la palma de la mano. También Reitmayr y Schmalstieg presentan un sistema móvil y multiusuario. Las ideas de la realidad aumentada móvil y la colaboración entre usuarios en un espacio ampliado para compartir, se combinan y se combina con un sistema híbrido.

AR-PDA

En 2002, Leonid Naimark y Foxlin Eric presentan un portátil de bajo consumo de seguimiento híbrido visual y por inercia. Este seguidor, que más tarde sería conocido como "Intersense IS-1200", se puede utilizar para el seguimiento a gran escala, como puede ser un edificio completo. Esto se logra mediante el seguimiento de un nuevo diseño de código de barras en 2-D con miles de códigos diferentes, combinando los resultados con un sensor de inercia.

En 2004, Mathias Möhring presenta un sistema de seguimiento de marcadores 3D en un teléfono móvil. Es compatible con la detección y diferenciación de los diferentes marcadores en 3D, y la correcta inyección de graficas en 3D en el flujo de video en vivo. Más tarde, el Proyecto ULTRA muestra cómo utilizar el seguimiento de características naturales en tiempo real para dispositivos móviles. El fin del proyecto era apoyar a las personas en varios dominios tales como el mantenimiento y el apoyo de máquinas complejas, construcción y producción, y educación y entretenimiento y el patrimonio cultural.

En 2006 Reitmayr presenta un sistema híbrido de seguimiento basado en modelos de RA al aire libre en el entorno urbano que permita inyectar de manera precisa y en tiempo real RA en un dispositivo de mano. El sistema combina un programa de seguimiento basado en GPS, información del giroscopio para hacer frente a movimientos rápidos, y mediciones de gravedad y campo magnético para evitar retrasos.

En 2007, Klein y Murray presentan un sistema robusto capaz de rastrear en tiempo real y en paralelo visualizar mapas con una cámara monocular en espacios de trabajo pequeños. Por su parte HIT Lab NZ y Saatchi y Saatchi liberan el primer teléfono móvil basado en RA que incluía una aplicación publicitaria para el Zoológico de Wellington.

En 2008, Wagner presenta la primera implementación 6DOF en tiempo real de seguimiento para teléfonos móviles que alcanza tasas de trama interactiva de hasta 20 Hz. Con una gran performance con el fin de ganar velocidad y reducir los requisitos de memoria.

También aparece NyARToolkit, una colección derivada de ARToolkit para máquinas virtuales, en particular las que albergan Java.

Mientras tanto varias aplicaciones de RA ven la luz:

- Mobilizy lanza Wikitude, una aplicación que combina GPS y brújula con las entradas de Wikipedia. El navegador mundial Wikitude Mundial superpone información sobre lo que se captura desde la cámara en tiempo real, para dispositivos Android.
- Sean White introduce SiteLens, un sistema móvil de RA para diseño urbano y planificación de visitas. SiteLens crea "visualizaciones situadas" que están relacionados y que aparecen en el entorno del usuario.
- SPRXmobile lanza Layar, una variante avanzada de Wikitude. Layar utiliza un mecanismo de registro único como Wikitude (GPS y brújula), y le incorpora a esto una plataforma cliente-servidor abierta. Existen varias capas de contenido para trazar con la captura en tiempo real. El 17 de agosto Layar global alcanza cerca de 100 capas de contenido.
- El 20 de octubre de 2008 aparece Wikitude AR Travel Guide, con el HTC Dream y el teléfono Android G1.

En 2009 ARToolkit es portado a Flash (FlarToolkit) por Saqoosha, por lo que empieza a aparecer la RA en el explorador.

A partir de este momento, y con disponibilidad en el mercado de dispositivos móviles de alta gama, de la mano de grandes empresas como Apple, HTC, Motorola y Samsung, corriendo sistemas operativos modernos como son Android y iPhone, la arquitectura móvil se convirtió en la preferida para los nuevos proyectos de RA. Mientras siguen apareciendo cientos de aplicaciones de RA, las existentes mejoran en contenido, fluidez y visualización.

2.5 - REALIDAD AUMENTADA EN DISPOSITIVOS MÓVILES

Los teléfonos móviles con cámara son los responsables de la popularidad actual de las aplicaciones RA. Todo lo que se necesita es un ordenador (y todos los móviles modernos son ordenadores) y una cámara. Con esto bastaría para desarrollar aplicaciones de RA basadas en marcadores. Pero lo que realmente ha hecho popular a este tipo de aplicaciones son tres pequeños chips que incorporan la mayoría de dispositivos de gama alta: el GPS, el magnetómetro y el acelerómetro.

- El GPS es una tecnología disponible desde hace años, cuyos dispositivos han disminuido tanto su tamaño y consumo que pueden ser integrados dentro de los teléfonos móviles.
- El magnetómetro permite detectar campos magnéticos, y por lo tanto puede ser utilizado como brújula.
- El acelerómetro mide las aceleraciones a las que es sometido el dispositivo.

La llegada de GPS, brújulas electrónicas, banda ancha móvil y la alta capacidad computacional de los dispositivos, han convertido a los celulares en el perfecto campo de juego para las aplicaciones de RA.

La realidad aumentada, especialmente cuando se combina con los dispositivos móviles y teléfonos inteligentes, tiene aplicaciones en casi todas las industrias y puede revolucionar la publicidad, el entretenimiento, la educación, el ejército y los servicios de emergencia, la arquitectura, la medicina, la manufactura, los negocios, las conferencias, la navegación y el turismo.

Las primeras aplicaciones de RA se basan en las PC de mesa estática con cámaras fijas o HMD, con computadoras portátiles incorporadas. Si bien las configuraciones proporcionaban un alto rendimiento y generalmente dejaban las manos libres, estas soluciones también presentaban varios inconvenientes (altos costos, atractivo social bajo y limitaciones a la destreza en los usuarios), que les impedían llegar a un amplio público de usuarios no técnicos.

Los dispositivos móviles han crecido recientemente en el poder de computación y en el procesamiento de gráficos 3D, sobre todo gracias a la introducción de procesadores de gráficos integrados, y además integrando las últimas cámaras y capacidades inalámbricas.

2.5.1 - Limitaciones móviles

Si bien en los enfoques iniciales era necesario el apoyo de hardware con más capacidad de cómputo debido precisamente a la baja capacidad de cómputo disponible en los viejos dispositivos, los avances de hardware han permitido un uso de dispositivos portátiles como plataformas de RA independientes.

A pesar de las capacidades de movilidad y el potente hardware de estos dispositivos, estos tienen también algunas desventajas inherentes:

- Aunque los nuevos dispositivos disponen de hardware integrado para la aceleración 3D, los altos consumos de energía imposibilitan que puedan compararse con la calidad gráfica que alcanza el hardware de escritorio.
- Estos dispositivos portátiles no suelen estar equipados con una unidad de procesamiento de punto flotante, y por lo tanto, sólo son capaces de realizar cálculos de punto fijo en el hardware, mientras que los cálculos de punto flotante se emulan en programas (lo que lo hace más lento). Esto hace prácticamente imposible utilizar algoritmos fuertemente basados en cálculos.
- Otro factor es el tamaño de la pantalla y el reducido campo de visión que limitan la interacción con el usuario. La técnica más utilizada por los usuarios es la denominada lente mágica. Al adoptar el enfoque de la lente mágica, el dispositivo se está empleando como una lente para aumentar una parte del mundo real. Una de las principales particulares de este enfoque es que son necesarios varios movimientos con la cámara.

2.5.2 - Aplicaciones para móviles

La RA se perfila para ser la "próxima gran cosa" en los dispositivos móviles. Las aplicaciones de RA muestran otra capa de información sobre la base del mundo real. Muchas aplicaciones utilizan la información superpuesta en la vista de la cámara a la par de información superpuesta en el mapa. Ser capaz de mirar a un lugar conocido con "nuevos ojos" es simplemente estupendo, y esto parece imaginar

lo grande que puede convertirse en el futuro.

La RA tiene innumerables aplicaciones, algunas hasta hace poco solo posibles en las películas de ciencia ficción, pero que ya son realidad. Un ejemplo es el uso de esta tecnología en proyectos educativos, como museos o centros de visitantes.

Se emplean conexiones inalámbricas con el objeto de mostrar objetos en tres dimensiones, por ejemplo, una pieza arqueológica, una planta o un animal, como un dinosaurio; también se emplea en la reconstrucción de paisajes en ruinas, mostrando el aspecto que debieron tener en el pasado; incluso, se pueden mostrar escenarios completos en los que el usuario pueda interactuar con los diferentes elementos en tres dimensiones.

El parque temático francés Futuroscope, fue uno de los primeros en utilizar la realidad aumentada en el desarrollo de sus recursos expositivos.

Desde hace un tiempo, se ha comenzado a ver cada vez más la utilización de la (RA en el área de educación. Actualmente está entre las seis tecnologías emergentes capaces de revolucionar la educación superior en Iberoamérica, según el Informe Horizon 2010.

Si bien es cierto que esta tecnología no es nueva, no ha sido hasta los últimos años que hemos podido ver y utilizar ejemplos de implementación de diferentes ámbitos de la sociedad, como es el caso de las áreas técnico-industriales, militar y del marketing. Otros sectores, como el de la educación, no han estado ajenos a las capacidades de la RA, aunque sus proyectos no estén más que en tubos de ensayo.

A continuación expondremos dos de los proyectos de RA más populares que podemos encontrar en las plataformas móviles con el fin de exponer las arquitecturas y componentes empleados y su funcionalidad.

2.5.2.1 - Layer

Layar es un navegador de RA que nació para el sistema Android. Es calificado como el primer "Navegador de Realidad Aumentada" para teléfonos móviles. Creado por la empresa SPRXmobile funciona sobre teléfonos que incorporen el hardware mínimo necesario: cámara de fotos, localizador GPS y brújula. Haciendo uso del GPS y de la brújula del móvil ubica la posición del usuario y su orientación. La pantalla del celular muestra lo que la cámara capta y, sobre esta imagen del entorno, superpone, en tiempo real, información relativa a lo que tenemos delante de nosotros.

La información está disponible en diferentes capas entre las que puede seleccionar el usuario. Por defecto, la primera capa visible es "Layar Local Search", que emplea datos de Google para señalar la posición de establecimientos de restauración, ocio o servicio en un radio de entre 25 metros y varios kilómetros; pero existen muchas más capas (más de 300 contando las realizadas por la comunidad) que permiten cosas tan diversas como conocer los bares en el entorno, obtener información inmobiliaria, consultar la Wikipedia, acceder a información sobre los transportes en la zona, etc. Entre los datos más interesantes: la aplicación es gratuita y también tiene su port oficial a la plataforma iPhone.

Layar tiene dos partes: el navegador a la experiencia de las capas de contenido, y la plataforma para servir y publicar el contenido de las capas. La plataforma es donde se definen las capas de contenido y actúa como el enlace a la editorial real que alberga sus datos. La definición de la capa de contenido es un componente clave de su capa. Esta definición da al espectador el título y el aspecto y la sensación de la capa, además de los meta-tags y la ubicación del servicio Web (donde los datos se encuentran alojado).

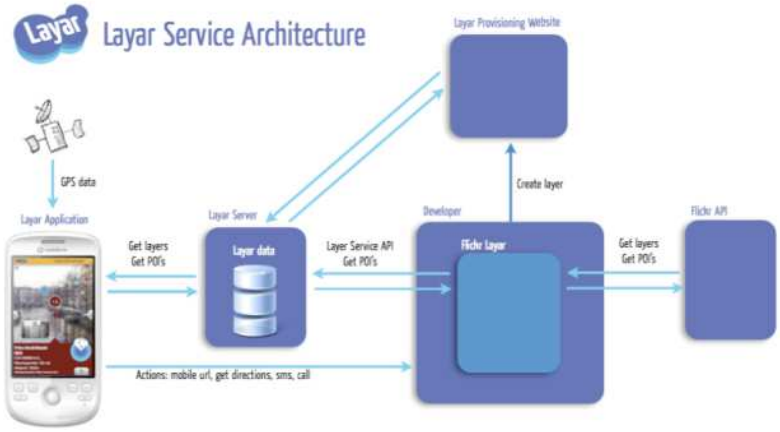


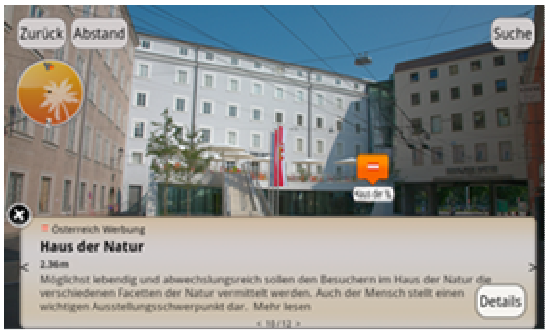
Figura 8. Arquitectura de la plataforma Layer

Como prácticamente todas las aplicaciones de este tipo, Layer también integra Google Maps para mostrar los resultados de sus capas sobre un mapa.

2.5.2.2 - Wikitude

Al igual que Layar, Wikitude es un navegador de RA que permite descubrir diversos mundos dentro del que nos rodea. Wikitude fue creado y es mantenido por la empresa austriaca Mobilizy y su primera versión fue publicada casi simultáneamente con salida del dispositivo G1 en los primeros meses de 2008. Existen multitud de contenidos disponibles para Wikitude, que funciona tanto para Android como así también para iPhone y Symbian.

Los servicios de información más importantes sobre esta aplicación se basan en la información de Wikipedia, y genera recomendaciones de los usuarios de Qype y la comunidad Panoramio para compartir fotos.



Wikitude en funcionamiento

Uno de los últimos lanzamientos de Wikitude ha sido Wikitude Drive, un navegador GPS en el cual la información de navegación se sobre impresiona sobre la visión de la cámara. De momento está solo disponible para Android y solo esta disponible en algunos países europeos. A diferencia del Wikitude World Browser, que es gratuito, su precio de salida es de 9,99 euros.

2.6 - ESTÁNDARES DE REALIDAD AUMENTADA

Aun no existen estándares o nomenclaturas a seguir a la hora de iniciar un proyecto de RA. Sin embargo, en el año 2009 se creó el que se llama "Consorcio AR", que intenta hacer frente a todos los faltantes de mejores prácticas para crear aplicaciones RA.

El Consorcio AR fue fundado en junio de 2009 para facilitar el desarrollo y avance de las tecnologías de realidad aumentada para el mercado global. Los miembros fundadores del Consorcio incluyen INT13, Metaio, Mobilizy, Neogence, Ogmento, Móvil SPRX y Tonchidot. Los objetivos del Consorcio AR incluyen proporcionar un foro para los miembros de reunirse, compartir ideas y discutir problemas que enfrenta el crecimiento de la industria, así como abrir un diálogo acerca de las nuevas normas y protocolos.

Estas empresas se han unido para formar el Consorcio RA, un grupo internacional que se centra en el desarrollo de herramientas, la tecnología, las aplicaciones y el contenido de RA. Las empresas fundadoras son todos innovadores de alto perfil en el campo, y esperamos que sus esfuerzos combinados se traduzcan en una mayor cooperación entre los miembros, más rápida penetración en el mercado, un conjunto de normas técnicas, y un fuerte enfoque en la experiencia del usuario final. e ha previsto una conferencia anual, con ubicación en rotación, ya que cada corporación fundadora tomará el papel de anfitrión.

2.6.1 - Especificación ARML

ARML (lenguaje de marcado para Realidad Aumentada) es una especificación creada por Mobiliizy, que permite a los desarrolladores de contenido crear contenido que se muestra en diversos navegadores RA móvil. Wikitude 4 es el primer navegador que soporta de forma nativa ARML, entre otros que la están adaptando. ARML se basa en un subconjunto de KML, un conjunto de etiquetas comunes de apoyados en todos los navegadores y extensiones específicas para algunos navegadores (por ejemplo, Wikitude). Esta especificación describe el conjunto completo de etiquetas con el apoyo de Wikitude 4.

El documento ARML debe ser un XML válido de codificación UTF-8(*). El texto de las etiquetas puede ser cerrado por una sección CDATA(*) para evitar caracteres sin caracteres de escape.

Los datos más importantes que contiene un documento ARML son:

- <Provider>: Identifica el proveedor de contenido o el contenido del canal (nombre, descripción, URL, logotipo)
- <Placemark>: Placemark describe un punto de interés (POI) en el navegador AR (nombre, descripción, miniatura, URL, adjuntos).
- <Point>: Las coordenadas se introducen en el formato de longitud, latitud, altitud. La altitud es opcional y debe ser expresadas en metros.

2.6.2 - KHARMA

Creado en el instituto Georgia Tech, es una plataforma abierta para la entregar experiencia de RA a dispositivos móviles [17]. Es una combinación de estándares abiertos: KML y HTML (KARML) desarrollados para servidores estándar HTTP que permiten hacer referencia a autores de panorámicas GEOSpot(*), y una infraestructura para desarrollar una argumentación de alta precisión.

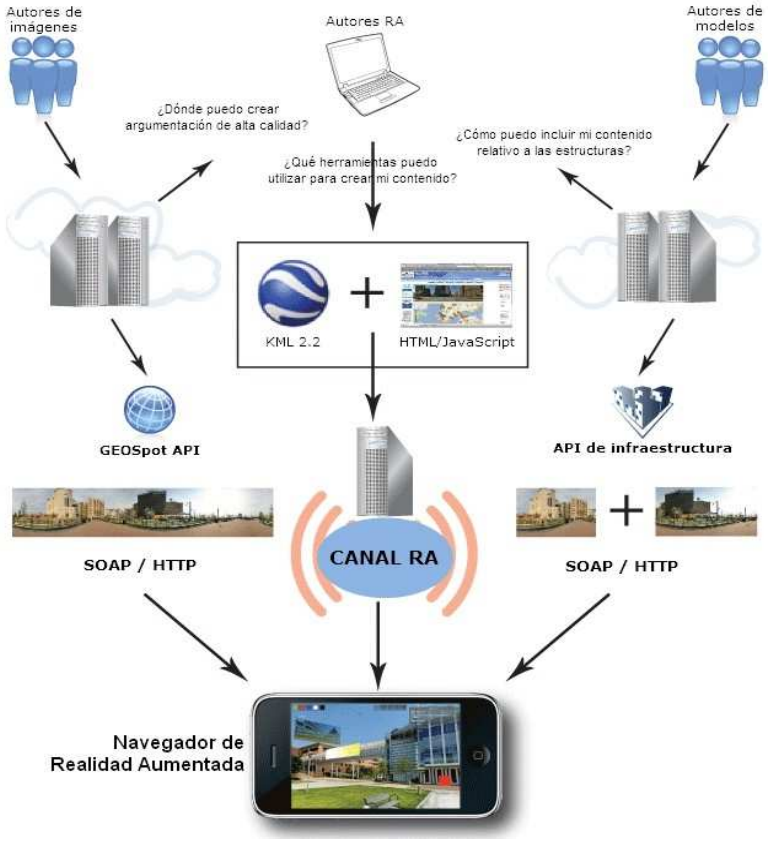


Figura 9. Arquitectura KHARMA

La arquitectura define un navegador RA móvil de código abierto permite al usuario encontrar estos puntos de interés locales a través de API de servicios basada en estándares SOAP(*), clasificándose cuando se está frente a una. A continuación, el navegador reemplaza la escena de video con el panorama y los modelos de las estructuras se superponen con el contenido del video en tiempo real.

El mismo instituto realizó una implementación de la arquitectura con una aplicación llamada igualmente a su arquitectura.

3 - SISTEMAS BASADOS EN LOCALIZACIÓN

Los servicios de localización ofrecen contenido que se modifica a medida según la ubicación del usuario. Estos servicios se envían normalmente a los dispositivos móviles, pero también se puede acceder a ellos desde otros dispositivos como ordenadores portátiles, ordenadores de mano, o cualquier dispositivo con capacidad de acceso a Internet. Las aplicaciones actuales más comunes para los servicios de localización incluyen la publicidad, noticias, redes sociales, y servicios similares.

Un número creciente de aplicaciones móviles se están aprovechando de la capacidad incorporada de geolocalización que es, cada vez más, una característica estándar en dispositivos móviles. Información sobre los edificios cercanos, puntos de referencia u otros elementos fijos es común, un uso creciente de servicios basados en localización es localizar a las personas cercanas - la gente conocida o desconocida para el usuario - que comparten intereses o experiencias. Los medios de comunicación, tales como fotos y vídeo, así como la sencillez de geotagging, serán aspectos importantes de los servicios de localización a medida que continúen desarrollándose.

Los servicios de localización son servicios que integran una localización o ubicación de un dispositivo móvil con otra información para proveer un valor agregado a un usuario [18]

Un servicio de localización, en el sentido más amplio, es cualquier servicio o aplicación que extienda el procesamiento espacial de la información, o las capacidades GIS, a los usuarios finales a través de Internet y / o red inalámbrica. [19]

Según la definición de *Brimicombe* [23], los LBS pude verse como una convergencia de 3 tecnologías: Internet, los Sistemas de Información Geográfica (GIS) y las tecnologías de las telecomunicaciones móviles.

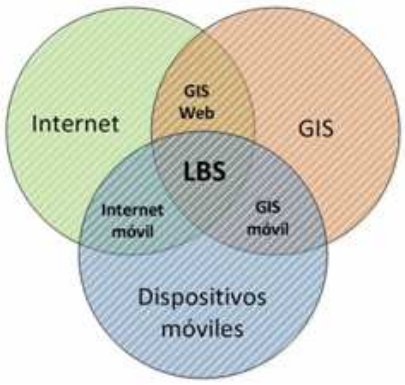


Figura 10. Composición de LBS

3.1 - GIS y LBS

Un Sistema de Información Geográfica (GIS) es una tecnología integrada por hardware , software y datos geográficos utilizada para recoger, almacenar, administrar, analizar y visualizar todos las formas de información geográficamente referenciada (georreferenciación), para resolver problemas complejos de planificación y gestión. Por lo tanto, ofrece respuestas a todo tipo de consultas relacionadas con la ciencia de información geográfica y se utiliza ampliamente en la planificación, gestión de recursos, gestión de activos, transporte, ciencias ambientales y la observación científica.

Los Servicios Basados en Localización (LBS) son en cambio la parte del servicio de información accesible a través de los teléfonos móviles u otros dispositivos PDA que utilizan la capacidad de los GIS y su base de datos. Así LBS es una combinación de GIS y sistema de telecomunicaciones móviles. Sin embargo existen diferencias [20]:

GIS:

- Desarrollado hace décadas aplicaciones profesionales de datos geográficos.
- Sistema orientado a usuarios experimentados.
- Amplia funcionalidad.
- Requiere importante recursos informáticos

LBS:

- Desarrollado hace muy poco por la revolución de los servicios para aplicaciones móviles.
- Sistemas orientado a usuarios no-profesionales
- Servicios limitados.
- Restricciones del dispositivo móvil (batería, poder computacional, etc.)

3.2 - COMPONENTES DE LOS SERVICIOS BASADOS EN LOCALIZACIÓN

Hay cinco grandes componentes en la arquitectura de un sistema que hay que reunir para lograr un LBS, el dispositivo móvil, la red de comunicación, el sistema de posicionamiento, el proveedor de servicio y el proveedor de contenidos [23].

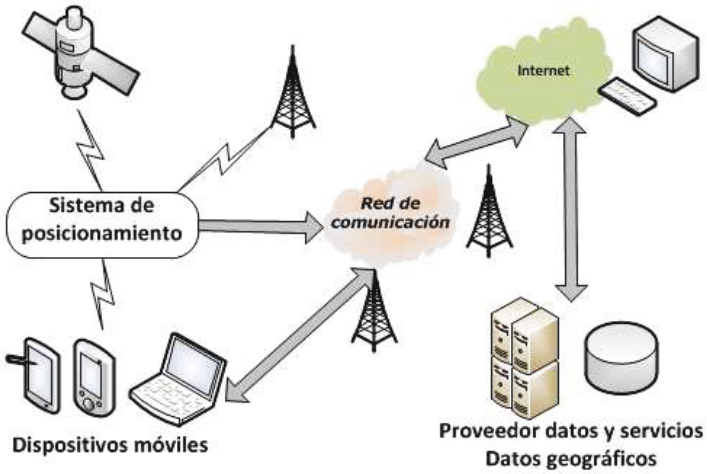


Figura 11. Arquitectura de sistemas LBS

Claramente todos estos componentes desempeñan un papel fundamental en la prestación del servicio basado en localización hacia el usuario final. El usuario final necesita el dispositivito móvil como interfaz para acceder al servicio.

Hay una necesidad de una red de comunicaciones que permite la comunicación en tiempo real entre el usuario, el proveedor de servicios y el sistema de posicionamiento. Luego el sistema de posicionamiento proporciona la ubicación del usuario, y los proveedores de contenida y datos brindan la información sobra varias entidades como las empresas locales que pueden ser de valor para los proveedores de servicios.

- Dispositivo móvil*:* Herramienta por la cual se solicita la información requerida, estos dispositivos pueden conocer la localización activamente, esto significa que puede determinar y comunicar su ubicación (ej. GPS) o pueden conocer la localización pasivamente, esto es que puede ser determinada por otros elementos del sistema al que está vinculado. Los dispositivos posibles pueden ser: laptops, smartphones, PDA's, teléfonos móviles, etc.
- Red de comunicación*:* La red móvil que transfiere los datos de usuario y la solicitud de servicio desde el dispositivo móvil al proveedor del servicio y, retorna la información solicitada al usuario. La red wireless puede ser ya sea Bluetooth, WIFI, red móvil o una combinación de estos.
- Sistema de posicionamiento*:* Los sistemas de posicionamiento permiten conocer la ubicación geográfica del dispositivo móvil del usuario. Esta posición puede obtenerse mediante el uso de los métodos de triangulación por red de comunicaciones móviles o mediante el Sistema de Posicionamiento Global (GPS).
- Proveedor de servicios*:* El proveedor de servicios ofrece una serie de servicios diferentes para el usuario basada en el interés del usuario y localización del mismo. También es responsable de procesar la solicitud de servicio.
- Proveedor de contenido y datos*:* Los proveedores de servicios no suelen almacenar y mantener toda la información que puede ser solicitada por los usuarios. Por lo tanto la información de datos geográficos y de localización se suelen solicitar por ejemplo a organismos cartográficos, páginas amarillas, empresas de transporte, etc.

3.3 - CARACTERÍSTICAS DE LOS SERVICIOS BASADOS EN LOCALIZACIÓN

Analistas e investigadores han tomado varios enfoques para clasificar las aplicaciones LBS. Una distinción importante es si son orientadas a personas o a dispositivos [18]. Las aplicaciones LBS orientadas a personas comprende todas las aplicaciones donde el servicio está basado en el usuario. Por lo tanto el enfoque del uso de la aplicación es la posición de la persona o utilizar la posición de la persona para mejorar el servicio.

Las aplicaciones LBS orientados a dispositivos son externas al usuario. De este modo, también pueden centrarse en la posición de una persona, pero no es necesario. En lugar de solo una persona pueden ser localizadas un grupo de personas (una flota) o un objeto (ejemplo: un auto). En las aplicaciones orientadas a dispositivos, la persona o el objeto localizado generalmente no controla el servicio (seguimiento de auto contra robo).

Además de esta primera clasificación de servicios, dos tipos de diseños de aplicaciones se pueden distinguir considerando si la información en entregada por la interacción del usuario o no: servicios push y pull [18].

- Servicios Push: Donde la posición del usuario o la proximidad a otro objeto activa algún evento o define alguna condición, esto implica que el usuario recibe información de su paradero sin tener que solicitarlo activamente. Esta información puede ser enviada al usuario con consentimiento previo (mensaje de alerta) o sin consentimiento (publicidad móvil, información instantánea cuando se ingresa a algún lugar).
- Los Servicios Pull en contraste, significa que el usuario activamente usa la aplicación, y, en este contexto, envía (pull) información a la red. Esta información puede ser por ejemplo una ubicación mejorada (ej. Cuando se busca el cine más cercano, asistencia en emergencias-E911).

3.4 - CONCEPTOS BÁSICOS DE LBS

Las aplicaciones basadas en localización (LBS) pueden ser categorizadas por una serie de palabras claves y preguntas relacionadas.

- Usuarios móviles: Un usuario móvil puede ser una persona o un dispositivo como un sistema de navegación para automóviles.
- Actividades móviles: Una actividad se define como una secuencia de acciones llevadas a cabo en un lugar específico durante un cierto tiempo. ¿Qué preguntas y problemas tienen los usuarios? De tales preguntas surgen las acciones de los usuarios: localización, navegación, búsqueda, identificación, control de eventos. Una pregunta adicional con respecto a las acciones es el alcance de las actividades. Según Reichenbacher [21] podemos distinguir tres tipos de alcance espacial: 1) escala macro: ¿Necesito un resumen? 2) la escala meso: ¿Qué es accesible para mí? 3) la escala micro: ¿Dónde estoy?
- Información: ¿Que se necesita para responder a las pregunta del usuario y cómo se hace? Un modelo de recuperación de información es necesaria para responder a las preguntas de los usuarios. Este modelo de proceso de información contiene un modelo de posibles preguntas, define las consultas a la base de datos geográfica y de localización de la información, y especifica las posibles respuestas.
- Interfaz de usuario: ¿Es una persona usando un PDA o teléfono móvil o algún otro dispositivo móvil? ¿Cómo puede el usuario o del sistema (de navegación) formular sus necesidades y puede que sean más concretas después de obtener una visión general?
- Visualización: ¿Como es la información retornada por el servicio basado en localización? Mapas, voz, texto, imágenes, etc.
- Tecnología: ¿Cómo son las solicitudes de servicio y los datos transferidos entre el usuario y proveedor de servicios? ¿Dónde están los datos almacenados? ¿Qué servicios se proporcionan? ¿Qué tecnología de posicionamiento se utiliza?

3.4.1 - Acciones y objetivos asociados al usuario

Las acciones resultan de un conjunto de deseos o preguntas del usuario. La pregunta más obvia es saber dónde está el propio usuario de alguien o algo (localización). Los usuarios pueden buscar a personas, objetos o eventos (búsqueda) y preguntar por el camino a una posición (navegación). Otras cuestiones pedir las propiedades de un lugar (identificación) o buscar eventos en o cerca de un lugar determinado (control-cheking).

Utilizar servicios basados en localización involucra 5 acciones fundamentales [21]: localización, navegación, búsqueda, identificación, control de eventos.

Una actividad es una secuencia de acciones llevadas a cabo por un ser humano destinados a lograr un objetivo determinado. Este objetivo podría ser la solución de un problema o una tarea. En situaciones móviles los objetivos son por ejemplo orientativos, la búsqueda de personas o encontrar el camino a un objeto. Los objetivos también pueden ser expresados por preguntas de las cuales el usuario del LBS quiere tener una respuesta.

La siguiente tabla relaciona las acciones elementales móviles con las preguntas que se trata de responder.

Acción		Pregunta	Objetivo	Servicio
	Orientación y Ubicación	Dónde estoy? Donde Esta [persona | objeto]?	Localizar personas y objetos	Entregar la posición de una persona y objeto

	<u>Navegación</u> Navegar a través de un espacio, la planificación de una ruta	Como llego [lugar | dirección]?	Encontrar el camino un destino	Entregar rutas e instrucciones de navegación al objetivo
	<u>Búsqueda</u> Búsqueda de persona y objetos	Donde está el [más cercano | más relevante] [persona | objeto]?	búsqueda de personas y objetos que reúnan los criterios de búsqueda	descubrir servicios disponibles, encontrar personas /objetos
	<u>Identificación</u> Identificar y reconocer persona y objetos	Quién es? Cuál Es?	identificar a las personas y objetos; cuantificar objetos	entregar (semántica) información sobre personas / objetos
	<u>Control de eventos</u> control de eventos; determinar el estado de los objetos	Que sucede aquí?	Saber que pasa, conocer el estado de un objeto.	Entregar información del estado de un objeto e información del evento

Tabla 4. Acciones fundamentales en LBS

3.5 - TECNOLOGÍA DE POSICIONAMIENTO LBS

Los servicios basados en localización necesitan obtener la posición del dispositivo móvil, para esto existen tecnologías que resuelven este problema. Los métodos de posicionamiento pueden categorizarse en basados en red, basados en dispositivos y en tecnologías hibridas [23].

Para el posicionamiento basado en dispositivos, el dispositivo móvil determina su ubicación a través de las señales que recibe, es decir, las mediciones de la señal y el cómputo para determinar una posición se llevan a cabo por el receptor ubicado dentro del dispositivo móvil. No hay conexión de red. El ejemplo más famoso es el GPS.

Los métodos de posicionamiento basados en red utilizan un conjunto de estaciones de la red de telefonía para localizar el dispositivo mediante la medición de la señal entre el dispositivo móvil y el conjunto de estaciones.

Con respecto a las tecnologías híbridas, existen dos enfoques: el primero se basa en que la tecnología de comunicación proporcione información adicional que mejore las prestaciones y el segundo en la combinación de la posición proporcionada por las dos tecnologías para dar un resultado más fiable. Un ejemplo es el GPS asistido (A-GPS). Aunque el GPS proporciona una posición precisa, necesitará una asistencia para solventar en cierta medida algunos problemas inherentes al sistema, tales como las situaciones en entornos donde el nivel de señal recibido es bajo o la señal directa está bloqueada (dentro de edificios o entornos urbanos). El A-GPS está conectado a la red GSM, y cuando el dispositivo móvil requiere una posición fija, los datos de asistencia de la red de referencia se transmiten para mejorar el rendimiento del receptor GPS

Otras tecnologías como Bluetooth, RFID (Radio Frequency Identification) y WLANs (Wireless Local Area Networks, WI-FI) pueden utilizarse para localización pero con un corto alcance. Con frecuencia se emplea para localización en lugares cerrados o interiores.

3.5.1 - GPS

Puede determinar en cualquier parte del mundo la ubicación de un objeto (ej: persona, auto, etc.) con gran precisión. El GPS o sistema global de posicionamiento en un sistema basado en satélites. Fue desarrollado por el departamento de defensa de los US.

El primer satélite fue lanzado en 1978 y el sistema comenzó a operar completamente en 1993. El GPS funciona mediante una red de 24 satélites que orbitan la tierra a 20.200 Km. con trayectorias sincronizadas para cubrir toda la superficie de la tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente como mínimo cuatro satélites, sin embargo, cuanto mayor sea éste, obtendremos una mayor precisión. De cada satélite se recibirá una señal indicando su posición y su reloj. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante triangulación (método de trilateración inversa), la cual se basa en determinar la distancia de cada satélite respecto al punto de medición.

La ventaja de este método es su exactitud y que su uso está muy extendido, además de no depender de ningún operador concreto. Sus limitaciones están en entornos cerrados (indoor) y zonas urbanas donde el posicionamiento puede ser inexacto o incluso inviable.

3.5.2 - Posicionamiento basado en red

Esta tecnología nos permitirá obtener la información de posicionamiento en todo el área cubierta por el operador de red móvil, incluyendo entornos urbanos o Indoor.

Entre este tipo de tecnologías cabe destacar los siguientes métodos:

- *Cell-ID (Cell Identity)* también conocido como COO (cell of origin) o Cell Global Identity(CGI). Este método consiste en encontrar la celda en la que se encuentra conectado el dispositivo móvil, esta información es utilizado para el cálculo de la posición dándonos una orientación de la zona donde se encuentra el usuario, la cual puede variar de 250 m a 5km.
- *ToA (Time of Arrival)* también conocido como tiempo absoluto de arribo (AToA - Absolute Time of Arrival). La posición del móvil es determinada por el tiempo en que tarda la señal

entre un número de torres de la red y el dispositivo móvil. Para obtener la posición se necesitan al menos 3 torres para poder hacer la triangulación. El método ToA puede proporcionar una precisión de posicionamiento dentro de un rango de 125-200m.

- *AoA (Angle of arrival)* La ubicación de un dispositivo móvil se determina midiendo el ángulo de la señal recibida a partir de dos torres de la red de telefonía. En general, el posicionamiento con AoA es capaz de alcanzar precisiones de dentro de los 300 m.

3.5.3 - WI-FI

Wi-Fi Positioning System (WPS) es un término creado por Skyhook Wireless] para describir un sistema de posicionamiento basado en WI-FI [29. Se trata de un software que contiene en sus bases de datos la posición de cualquier red Wi-fi que hayan detectado (esto se hace recorriendo las ciudades, calle por calle, localizando y registrando en la base de datos las redes que se encuentran). Estos datos pueden se pueden acceder a través de una API, y obtener el posicionamiento basado en los puntos de acceso accesibles desde un terminal.

Puede de termina la posición de un dispositivo móvil dentro de 10-20 mts.

3.6 - APLICACIONES BASADAS EN LOCALIZACIÓN

Las aplicaciones típicas LBS buscan proveer servicios geográficos en tiempo real. Algunos ejemplos típicos de esto son servicios de mapas, enrutamiento y páginas amarillas geográficas. Mostramos a continuación los distintos ámbitos de aplicación de esta tecnología.

3.6.1 - Navegación

A partir de la posición actual permite establecer rutas para llegar a destino (fija o móvil). Podemos encontrar una serie de servicios basados en navegación tales como los sistemas de navegación vehicular que asisten al conductor identificando las rutas de manera optimizada, sistemas de gestión de tráfico, estos servicios de gestión del tráfico tomar en cuenta las condiciones de tráfico actuales (por ejemplo, la congestión del tráfico o un accidente de tráfico de bloqueo) y sugerir rutas alternativas a los usuarios móviles.

Google maps navigation

Es un sistema de navegación GPS con conexión a Internet que proporciona indicaciones por voz. Funciones provistas por esta aplicación: búsqueda por voz, vista de tráfico, diferentes tipos de vista (satélite, modo coche, navegación para peatones, street view), búsqueda de cualquier tipo de empresa por la ruta en que vamos y la posibilidad de activar las capas más populares de Google Maps como las que muestran estaciones de servicio, restaurantes o estacionamientos.

Otros navegadores GPS conocidos OVI Maps-Nokia, TomTom, Garmin, etc.

3.6.2 - Emergencia

Transmite la posición exacta a fuerzas de emergencia y seguridad para una expedita asistencia/rescate. Ejemplos: E-911, SAR.

3.6.3 - Información

Los servicios de información sensible a la ubicación generalmente se refieren a la distribución digital de contenidos para dispositivos móviles basados en su ubicación, las especificaciones del tiempo y comportamiento de los usuarios. Los siguientes tipos de servicios pueden ser identificados dentro de esta categoría: sistemas de información de tráfico, páginas amarillas, sistemas de información de clima, información de transporte público, sistema de información turística para conocer puntos de interés como eventos, horarios de atención, información histórica, etc.

Google Maps

Es un servicio de mapas al que puedes acceder desde un navegador Web, desde una aplicación de escritorio como Google Hearth o a través de un dispositivo móvil. Dependiendo de la ubicación, se podrán visualizar mapas básicos o personalizados e información sobre negocios locales, como la ubicación, datos de contacto y cómo llegar hasta ellos. La opción de capas superpuestas en el mapa, por ejemplo, de tráfico, imágenes de satélite y resultados de búsquedas, entre otras.

WebParck

Es una plataforma que permite el despliegue de los servicios basados en la localización en espacios naturales. WebPark creó la posibilidad de proporcionar información a los visitantes de los espacios naturales a través del uso de los teléfonos inteligentes y el GPS. Una guía WebPark es un sitio Web para móviles que incluye contenido dinámico que cambia con los visitantes la ubicación, el tiempo y los intereses.

3.6.4 - Publicidad y facturación

Los servicios basados en la localización ofrecen una gran oportunidad al marketing directo. A través del móvil y contando con datos de localización geográfica, se puede enviar el mensaje adecuado en el momento más oportuno. Los datos de clientes relacionados con sus necesidades, actitudes y comportamientos de compra o afinidad a productos, combinados con las funciones de GPS de algunos móviles, ofrecen una combinación irresistible para el marketing directo.

También la información local puede convertirse en un excelente camino para llegar al consumidor. Por ejemplo: el móvil se usa bastante a mediodía. ¿Por qué no enviar cupones de descuento para que el destinatario coma en algún restaurante de la zona? O un patrocinador podría enviar información de los conciertos que se celebran en los alrededores un viernes cualquiera.

Hay diversos mecanismos para la aplicación de la publicidad móvil con LBS. Ejemplos de ello son banners móvil, alertas (por lo general en forma de mensajes SMS) y los anuncios de proximidad.
Los servicios de facturación pueden determinar distintas tarifas a los usuarios dependiendo su localización. Ejemplo: las empresas de telefonía pueden cobrar las llamadas dependiendo de la ubicación se encuentre el celular.

LocationPoint

NAVTEQ Location Point es una red de publicidad basada en ubicación que envía anuncios a los usuarios de smartphones, cuando se encuentran en las proximidades de un establecimiento donde pueden ir de compras o aprovechar un servicio.

NAVTEQ LocationPoint aprovecha los datos de NAVTEQ, incluidos los mapas de navegación más utilizados por los consumidores e incluye información de anunciantes para ayudar a millones de usuarios de dispositivos de navegación GPS a buscar y encontrar su camino a los anunciantes.

La publicidad en LocationPoint proporciona anuncios creativos que incluyen acciones como: *"haga clic para buscar", "clic para llamar", "haga clic para navegar" y "click para obtener cupón".*

NAVTEQ es un proveedor mundial de mapas, información basada en la localización y datos relativos al tráfico y servicios basados en la localización

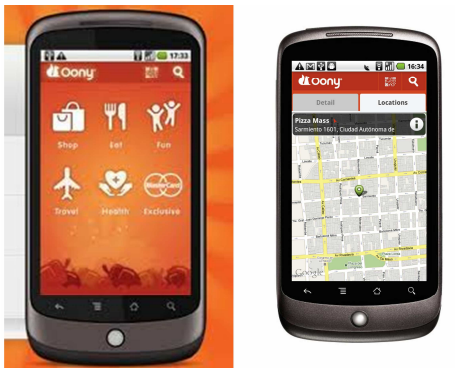
Oony

Oony es un buscador de beneficios. Se trata de una aplicación para dispositivos móviles que permite realizar búsquedas de beneficios y descuentos de acuerdo a las preferencias personales de cada usuario, sus medios de pago y su localización geográfica, para luego devolverle resultados relevantes y oportunos.

Oony funciona en todos los dispositivos móviles Android, Iphone, Blackberry, iPad y otros con Internet móvil o Wi-Fi, e incluso envía alertas por SMS, mail y Chat. El servicio se activa ingresando en http://oony.com.ar.

Mediante una simple registración en oony.com cada usuario selecciona su lista de intereses personales, compuesto por categorías y temas de interés. Luego indica sus clubes de membresía, bancos y tarjetas de crédito (sólo los nombres, no es necesario revelar datos confidenciales). El sistema le devuelve al usuario los beneficios, ofertas y descuentos existentes a su alrededor. Por otro lado Oony aprende a través de su uso para mejorar la exactitud de las ofertas seleccionadas para cada usuario, ó *Ooner*.

La funcionalidad de geo-localización es también customizable. Si un usuario se encuentra en Palermo pero desea consultar, por ejemplo, cuáles ventajas y descuentos lo esperan en San Isidro, simplemente cambia la ubicación en el sistema y vuelve a realizar la consulta.



Otras aplicaciones pueden ser Zona Galicia, del banco Galicia, es una aplicación geolocalizada para encontrar ofertas y beneficios ofrecidos por el banco en un radio de 20 metros a la redonda.

3.6.5 - Seguimiento

A través de la toma periódica de datos georreferenciados en el tiempo se consigue reconstruir rutas recorridas. Esta información puede ser analizada online (si el sistema reporta la misma) o procesada posteriormente (ej. descarga local).

Las aplicaciones incluyen seguimiento de flotas, monitoreo y envió de taxis, administración de personal, gestión móvil de cadena de suministro, manutención de la seguridad de los hijos, realizar seguimiento de personas de edad avanzadas y enfermas, seguimiento de mercancía y paquetes.

Este último ejemplo es a menudo soportado por seguimiento de objetos a medida que pasa por puntos de control fijos (ej: código de barras)

3.6.6 - Juego y ocio

En los juegos y ocios (deportes, deportes extremos) los jugadores y acciones son basados en localización.

Un juego basado en la ubicación es aquel en la que el juego de alguna manera evoluciona y progresa a través de la ubicación de un jugador. Así, los juegos basados en la localización casi siempre algún tipo de apoyo de la tecnología de localización, por ejemplo, un sistema de posicionamiento por satélite como el GPS. Los "Urban gaming" o " Street Games" son típicamente juegos multi-jugador basados en la localización que se juega en las calles de la ciudad y la acumulación de los entornos urbanos.

Pac-Manhattan

Utiliza el área de Washington Square Park y alrededores para jugar una versión de carne y hueso de Pacman. Los jugadores de Pac-Manhattan comunican su posición a través de teléfonos móviles.

GPS Mission

El funcionamiento es simple, cuando este en la calle, por la localización del GPS, se tendrá las misiones disponibles en la ciudad y al pasar de pantalla se podrán ver la ubicación del jugador en el mapa, la distancia desde su posición y la dirección que tiene que tomar.

Básicamente, para diseñar una misión, se requiere diseñar un recorrido sobre el mapa y establecer los puntos de control/pruebas donde se pueden plantear, desde sencillas preguntas que hay que responder, hasta complicados enigmas que hay que resolver. El incentivo, además del juego en sí, es poder acumular puntos para intercambiarlos por trofeos virtuales y subir posiciones en el ranking de usuarios del sitio Web.

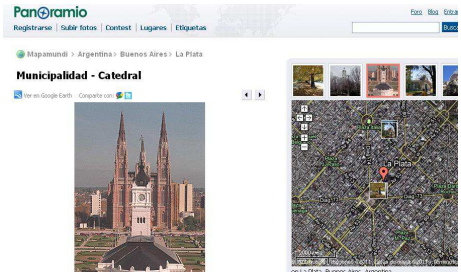
Aunque su aplicación inmediata son los juegos tipo: búsqueda del tesoro o escenificar historias de crímenes y misterio. También se puede aplicar a viajes a través de la historia local o cualquier otro tipo de visita guiada. Desde su lanzamiento, hace casi un año, los usuarios han creado misiones de todo tipo en más de 50 países.

3.6.7 - Realidad Aumentada

La realidad aumentada es una tecnología que utiliza un conjunto de dispositivos que añaden información virtual a elementos físicos ya existentes, es decir, sobreimprime los datos almacenados en una base de datos sobre las imágenes captadas de un elemento real.

Panoramio

Es un sitio Web dedicado a exhibir las fotografías de lugares o paisajes que los propios usuarios crean y georreferencian. Las imágenes que cumplen ciertos requisitos pueden ser vistas a través del software Google Earth.



Google Goggles

Es una aplicación para teléfonos Android que realiza búsquedas en Google a partir del contenido de una fotografía tomada con el móvil. En la actualidad este sistema reconoce: lugares, obras de arte, logotipos, monumentos, texto, vinos, revistas y libros

Otros ejemplos de este tipo de aplicaciones son**:** Google Earth, TwittARound**,** Wikitude ar-travel guide, layar, yelp monocle, etc.

3.6.8 - Redes Sociales

Muchas redes sociales están extendiendo la funcionalidad de sus aplicaciones para ser utilizadas en dispositivos móviles y proveer servicio de localización. La geolocalización es en estos momentos la tendencia de moda por excelencia en las redes sociales y parece que todas quieren darse prisa en adaptar sus funcionalidades para no quedarse atrás.

Facebook place (Quién. Qué. Cuándo. Y ahora: Dónde**)**

Facebook se unió a la moda de la localización geográfica al lanzar la herramienta Places para Smartphones que permite saber en tiempo real dónde están sus usuarios en todo momento. Places permite controlar los detalles y los usuarios de Facebook con los que se va compartir la información. Por ejemplo, Places puede marcar si un usuario está en el mismo lugar con otros amigos, de la misma forma que Facebook permite identificar las personas que aparecen en una foto.

Google Buzz

La red social de Google, tiene características similares a las redes sociales más importantes. Permite compartir fotos, vídeos y todo tipo de contenidos mientras el usuario está al tanto sobre lo que le interesa. Su característica más innovadora fue la total integración con Gmail, desde el primer momento y sin necesidad de instalar nada. La versión móvil de Buzz es totalmente sorprendente, añadiendo un arma definitiva: la localización. Es posible compartir fotos, vídeos, anotaciones y pensamientos con nuestra red de contactos desde el móvil, añadiendo información del lugar y el contexto en el que nos encontramos.

Otras redes que permiten localización son Twitter o Foursquare, etc.

3.7 - ESTÁNDARES

OMA (open mobile alliance)

La Open Mobile Alliance [28] es una Organización de estándares que desarrolla estándares abiertos para la industria de telefonía móvil.

La misión de la Open Mobile Alliance es proporcionar servicios ínter operables que permitan trabajar a través de países, operadoras y dispositivos móviles.

3GPP (3rd Generation Partnership Project) y 3GPP2 (3rd Generation Partnership Project 2)

Es un acuerdo de colaboración en tecnología de telefonía móvil, que fue establecido en diciembre de 1998. El objetivo del 3GPP es hacer global aplicaciones de tercera generación 3G (teléfono móvil). Los sistemas 3GPP [26] están basados en la evolución de los sistemas GSM, ahora comúnmente conocidos como sistemas UMTS (Universal Mobile Telecommunications System- sistema universal de telecomunicaciones móviles).

El 3GPP puede confundirse con 3GPP2 [27] cuyo estándar de especificación está basado en tecnología IS-95 comúnmente conocida como CDMA2000.

En el contexto de los LBS, el trabajo de 3GPP es de alta relevancia para las tecnologías de posicionamiento dentro y fuera de las redes celulares. Las especificaciones de 3GPP son continuamente mejoradas con nuevas características.

IETF (Internet Engineering Task Force)

Es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet. La IETF [25] es mundialmente conocida por ser la entidad que regula las propuestas y los estándares de Internet, conocidos como RFC (Request for Comments).

El objetivo de la IETF es hacer que Internet funcione mejor produciendo documentos técnicos relevantes y de alta calidad que influyen en la manera en que las personas diseñan, usan y manejan Internet [24].

3.8 - LOS SISTEMAS BASADOS EN LOCALIZACIÓN Y LA PRIVACIDAD

El tema de la privacidad en los sistemas basados en localización tiene gran importancia, ya que información como la ubicación/localización del usuario pueden ser utilizados sin conocimiento de este o autorización.

Es importante observar que es beneficioso en cuanto a seguridad ya que se puede conocer la ubicación de una persona en un momento determinado y puede ser útil en situaciones de emergencias. Pero la mayoría de las aplicaciones son de uso comercial exponiendo a los usuarios a un posible acceso no autorizado a los datos que pueden inducir movimientos y hábitos personales.

La capacidad de deducir el comportamiento de los ciudadanos combinado datos de diferentes fuentes representa una grave vulnerabilidad de la privacidad. Los proveedores de servicios basados en localización y también los usuarios de los servicios deben ser cuidadosos y sensibles sobre la manera en que manejan la localización de los demás.

La información de los usuarios puede ser mal utilizada aumentando así los problemas de seguridad tanto para el punto de vista de los datos personales privacidad y la seguridad nacional.

3.9 - LIBRERÍAS DE LOCALIZACIÓN

Se han desarrollado APIs (librerías software) para el aprovechamiento de la información de contexto de localización proporcionada por los métodos, técnicas y tecnologías en diferentes plataformas de software que existen para terminales móviles:

- J2ME JSR-179 (GPS)
- Symbian S60 Location API(GPS, A□GPS, E□OTD)
- iPhone Core Location API (WiFi, id celda, GPS)
- Android Location API (GPS)

4 - SISTEMA OPERATIVO ANDROID

El sistema operativo Android está basado en el kernel de Linux y ha sido desarrollado bajo el amparo de Google. En principio, nació como un sistema operativo para móviles, aunque hoy en día lo encontramos también en tablets y en Google TV. Desde su presentación en noviembre de 2007, estamos actualmente en la versión 2.3, una versión que quiere dar un punto de madurez hasta la siguiente versión: la 3.0.

Android ofrece un conjunto completo de software de código libre para dispositivos móviles: un sistema operativo, middleware y aplicaciones esenciales para móviles [35]. Es uno de los más recientes SO para móviles del mercado, está basado en una versión modificada del kernel de Linux. Las aplicaciones para Android se programan en lenguaje Java y son ejecutadas en una máquina virtual diseñada para esta plataforma, que ha sido bautizada con el nombre de Dalvik, está optimizada para requerir poca memoria y diseñada para permitir ejecutar varias instancias de la máquina virtual simultáneamente, delegando en el sistema operativo subyacente el soporte de aislamiento de procesos, gestión de memoria e hilos.

Permite un acceso fácil a prácticamente todas las funcionalidades hardware de los dispositivos en los que esté instalado. Provee a los desarrolladores de librerías que permiten la creación ágil y rápida de aplicaciones.

Android surge como resultado de la Open Handset Alliance, un consorcio de 48 empresas distribuidas por todo el mundo con diversos intereses en la telefonía móvil y un compromiso de comercializar dispositivos con este sistema operativo. El desarrollo viene de lo avalado principalmente por Google (tras la compra de Android Inc. en 2005) y entre las compañías encontramos compañías de software (Ebay, LivingImage...), operadores (Telefónica, Vodafone, T-Mobile, etc), fabricantes de móviles (Motorola, Samsumg, Acer, LG, HTC, etc) o fabricantes de Hardware (nVidia, Intel o Texas Instruments). Su objetivo es abstraer el hardware y facilitar el desarrollo de nuevas aplicaciones.

Se distribuye bajo una licencia Apache [34], versión 2, es una licencia de software libre creada por la Apache Software Foundation(ASF). La Licencia Apache permite al usuario del software la libertad de usarlo para cualquier propósito, distribuirlo, modificarlo, y distribuir versiones modificadas de ese software. La licencia Apache requiere la conservación del aviso de copyright y el disclaimer, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

Android Market, uno de los grandes laureles de Android, es una tienda de aplicaciones de donde podemos descargar aplicaciones a nuestro sistema. Buena parte del éxito de la plataforma Android se debe a su aceptación por parte los programadores y la venta de aplicaciones en el *Android Market*, la tienda de aplicaciones de Android, en la que se permite a los desarrolladores publicar, sin ningún tipo de filtro, aplicaciones tanto gratuitas como de pago.

4.1 - HISTORIA

En julio de 2005, Google compra Android Inc., una pequeña empresa de Palo Alto, California. Esto dio pie a rumores de que Google estaba planeando entrar en el mercado de los teléfonos móviles.

En Noviembre de 2007, Google anunciaba la creación de la Open Handset Alliance(OHA), un grupo de 79 empresas [31] de de hardware, software y telecomunicaciones que se unió con el objetivo de desarrollar estándares abiertos para móviles. Esta alianza incluye compañías como T-Mobile, Sprint Nextel, telefónica, Google, HTC, LG, Samsung, Motorola, Toshiba, Sony Ericsson, Nvidia, etc. Junto con la creación Open Handset Alliance se lanza la primera plataforma móvil Android construidas en la versión 2.6 del kernel de Linux.

En Agosto de 2008 se anuncia la aparición del Android Market que finalmente es lanzado al público el 22 de octubre de ese mismo año. Android Market es una tienda de aplicaciones de donde se puede descargar aplicaciones a los teléfonos móviles.

El 21 Octubre del 2008 el código fuente de Android está disponible como proyecto Open Source bajo licencia Apache 2.0, una licencia libre y gratuita, luego, el 22 de Octubre de 2008 aparece el primer teléfono que utilizaba la tecnología Android, el HTC Dream (T-Mobile G1).

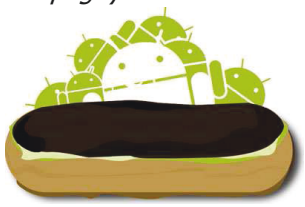
En Febrero de 2009, aparece la primera versión del SDK de Android 1.1, en Noviembre de este mimo año Motorola lanza en Estados Unidos el Droid también conocido como Milestone uno de los celulares con Android más populares hoy en día, que alcanzo ventas increíbles en solo pocos días.

En Enero del 2010 Google lanza Nexus One su primer teléfono móvil fabricado por HTC.

De donde surge el nombre de Android?

Tanto el nombre Android como *Nexus One* hacen alusión a la novela de Philip K. Dick *¿Sueñan los androides con ovejas eléctricas?*, que posteriormente fue adaptada al cine como *Blade Runner*. Tanto el libro como la película se centran en un grupo de androides llamados replicantes del modelo Nexus-6.

4.2 - VERSIONES SDK DE ANDROID

Versión	Características
1.1	La imagen del sistema Android 1.1 entregada junto con su versión de SDK, se desplegó en dispositivos móviles a principios de 2009.
1.5 *Cupkate (magdalena glaseada)* 	Versión liberada el 30 de Abril del 2009. Hubo varias características nuevas y actualizaciones en la interfaz de usuario en esta versión. • Teclado en pantalla • Grabación de video • Soporte para Widget • Soporte para Bluetooth • Aplicaciones Google • Actualizaciones en el navegador Web [http://developer.android.com/sdk/android-1.5-highlights.html]
1.6 Donut (dona o rosquilla) 	Versión liberada el 15 de Septiembre del 2009. • Cuadro de búsqueda rápida (Quick Search Box), permite una búsqueda rápida sobre múltiples fuentes como los favoritos del navegador y el historial de navegación, los contactos y la Web directamente desde la pantalla principal • Mejorada la velocidad de la cámara • posibilidad de conectarse a redes VPN, 802.1x • Nueva pantalla de indicador de batería. • Mejoras en Android Market aparecen ahora ordenadas por categorías (Aplicaciones, Juegos y Descargas). • Nuevo motor de texto a voz [http://developer.android.com/sdk/android-1.6-highlights.html]
2.0 / 2.1 Eclair (pastel francés conocido en España como *Relámpago*) 	El 26 de octubre de 2009, se libera la versión 2.0, el 3 de Diciembre del mismo año la versión 2.0.1 y el 12 de enero de 2010 se libera la SDK 2.1 • Nuevo interfaz de usuario en el navegador y soporte del navegador para HTML5. • soporte nativo de flash para la cámara, zoom digital, balance de blanco, efectos de color y modo macro • Mejoras del teclado virtual. • soporte para nuevos tamaños y resoluciones de pantalla, fondos de pantalla animados. • Contacto rápidos • Bluetooth 2.1 • Mejoras en Google Maps, que pasaba a ser multitáctil y soportar capas. Google Goggles (AR). • Mejoras de calendario. • Soporte para Microsoft Exchange. • MotionEvent mejorado para captura de eventos multi-touch. • Velocidad de Hardware mejorada [http://developer.android.com/sdk/android-2.1.html].
2.2 Froyo (abreviatura de «frozen yogurt» Yogur helado) 	La versión de la SDK 2.2 Liberada el 20 de Mayo del 2010 • Soporte para la instalación de aplicación en la memoria expandible • Optimización general del sistema Android, la memoria y el rendimiento, mejoras en la velocidad de las aplicaciones, gracias a la implementación de Dalvik JIT • Actualización del Market con actualizaciones automáticas • Soporte para Adobe Flash 10.1 • Cambio rápido entre múltiples idiomas de teclado y sus diccionarios [http://developer.android.com/sdk/android-2.2.html]
2.3 Gingerbread (pan de Gengibre)	La versión de la SDK 2.3 Liberada el 6 de Diciembre del 2010 • Actualización del diseño de la interfaz de usuario

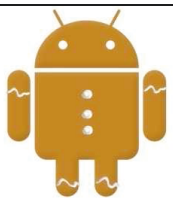
	• Teclado multi-táctil rediseñado • Soporte nativo para múltiples cámaras • Soporte nativo para más sensores (como giroscopios y barómetros) • Soporte para pantallas extra grandes • Soporte nativo para telefonía VoIP SIP [http://developer.android.com/sdk/android-2.3-highlights.html]

Tabla 5. Versiones de Android

4.2.1 - Características

Las principales funcionalidades que presenta el sistema se pueden enumerar en:

- Un Framework de aplicación que permitan la reutilización y sustitución de componentes.
- Una maquina virtual Dalvik optimizada para dispositivos móviles.
- Navegador integrado basado en el motor WebKit de código abierto.
- Biblioteca de gráficos optimizados en 2D, gráficos en 3D basada en la especificación OpenGL ES 1.0.
- SQLite para almacenamiento de datos estructurados.
- Soporte para los medios de comunicación de audio comunes, de vídeo, y formatos de imagen.
- Telefonía GSM.
- Soporte para Bluetooth, EDGE, 3G y WiFi.
- Soporte para cámara, GPS, brújula y acelerómetro.
- Entorno de desarrollo completo incluyendo emulador, herramientas de depuración, profiling de memoria y rendimiento, incluyendo un plugin para el IDE de Eclipse.

4.3 - ARQUITECTURA ANDROID

Como ya hemos comentado, el corazón de Android es el kernel Linux, donde se encuentran los drivers necesarios para el acceso al hardware, en concreto, para la gestión de la pantalla, el teclado, la cámara, la red Wi-Fi, el audio y la memoria Flash, entre otros.

En la siguiente figura se muestra la composición de la arquitectura de Android, basada en cuatro niveles, que detallamos a continuación.

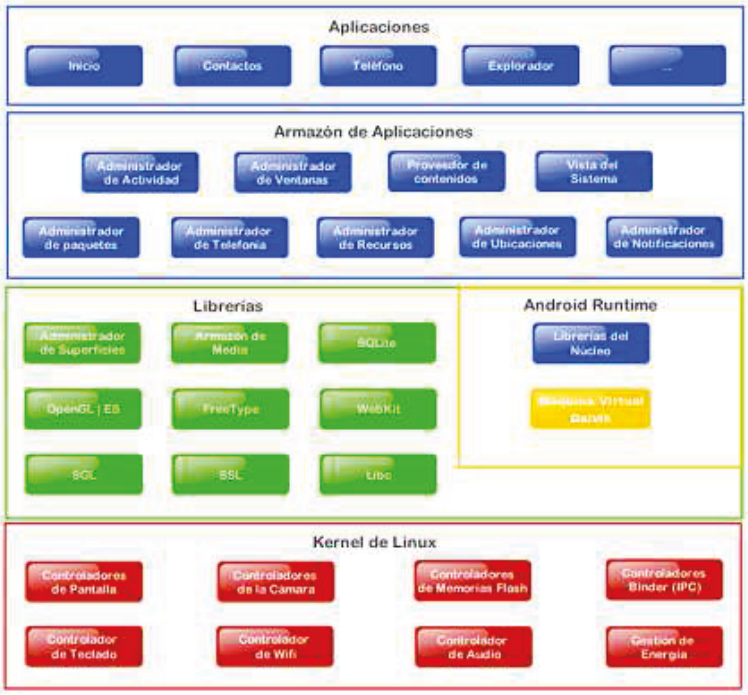


Figura 12. Arquitectura del sistema Android

En principio, el desarrollador no accederá directamente a la capa de drivers, sino que utilizará una serie de librerías que están en un nivel superior y que nos abstraen del hardware. Estas librerías, entre las que se incluyen la propia libc, están programadas en C.

En las siguientes líneas se dará una visión global de las distintas capas por la que está compuesta la arquitectura Android [35].

Aplicaciones

Android se distribuye con un conjunto de aplicaciones básicas, como un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos, y otros. Las aplicaciones están escritas en el lenguaje de programación Java.

Framework de aplicaciones

Al proporcionar una plataforma de desarrollo abierto, Android ofrece a los desarrolladores la capacidad de crear aplicaciones muy ricas e innovadoras. Toda aplicación que se desarrolladas en Android utilizan el mismo conjunto de API y el mismo *framework* que las utilizadas por el aplicaciones principales.

Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

- Activity Manager: Gestiona el ciclo de vida de las aplicaciones Android.
- *Window Manager*, gestiona las ventanas de las aplicaciones.
- *Telephone Manager*, incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.)
- *Content Providers*, permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.
- *View System*, proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, *check-boxes*, tamaño de ventanas, control

de las interfaces mediante tacto o teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
- *Location Manager*, posibilita a las aplicaciones la obtención de información de localización y posicionamiento.
- *Notification Manager*, mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada *Intent*, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.
- *XMPP Service*, colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.

Librerías

Android incluye un conjunto de bibliotecas C/C++ utilizadas por los diversos componentes del sistema Android. Estas capacidades están expuestas a los desarrolladores a través del framework de aplicaciones. Entre las librerías más importantes en este nivel se pueden encontrar:

- Librerías gráficas que incluyen *OpenGL* y *SGL* para manejo de gráficos 3D y 2D, respectivamente.
- *SQLLite* motor de base de datos relacional. Nos permite tener una base de datos local con la que gestionar todos los datos de nuestra aplicación.
- *Webkit*, con el que podemos integrar el motor de renderización Web que usan Safari, Chrome y el propio Android.
- *Surface Manager* provee gestión de las pantallas. Gestiona las ventanas pertenecientes a las distintas aplicaciones activas en cada momento
- *SSL* para mantener la seguridad sobre Internet.
- *Media Framework*. Para poder hacer uso de audio y vídeo, debemos usar esta librería.
- *FreeType*. Con ella podremos renderizar textos usando diferentes fuentes.
- La librería *libc (*standard C system library) incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.

Runtime Android

Cada aplicación Android se ejecuta en su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito para que un dispositivo pueda ejecutar varias máquinas virtuales de manera eficiente.

Kernel de Linux

Android se basa en la versión Linux 2.6 para los servicios del núcleo del sistema, como la seguridad, gestión de memoria, gestión de procesos, pila de red, y el modelo del controlador. El kernel provee una abstracción entre el hardware y el resto de las capas.

Entre los tipos de componentes más relevantes (los encontramos en la capa del Armazón de aplicaciones), podemos describir los siguientes:

- *Activity*, es el encargado de gestionar la interfaz de usuario.
- *Service*, que se ejecutan totalmente en segundo plano.
- *Broadcast receivers*, componentes se ejecutan en segundo plano que reciben datos de cambios en el sistema.
- *Content provider*, necesario para la compartición de datos entre los distintos componentes de nuestra aplicación.

Describiremos brevemente cada uno de estos componentes mas adelante.

4.4 - INTENTS E INTENTSFILTERS

Una de los elementos claves de Android es la clase Intent. Un *Intent* describe una "intención", lo que sería lo mismo que decir "lo que una aplicación desea que se haga". Podría entenderse como declarar una necesidad. Por ejemplo, una aplicación puede requerir visualizar una página Web. Para ello crearía un nuevo Intent del tipo VIEW y añadiría como datos de la intención la *URI* de una página Web.

Por otra parte hay una clase relacionada llamada IntentFilter. Mientras un Intent es un deseo de hacer algo, un *IntentFilter* es una declaración de las capacidades de una aplicación (En el caso de los navegadores, podrás registrarse para atender Intents para visualizar sitios Web). De esta forma, en el ejemplo anterior el sistema registraría la creación de un Intent y buscaría qué aplicaciones son capaces de resolverlo. Una actividad puede declarar sus *IntentFilters* tan genéricos o específicos como desee, teniendo en cuenta los datos asociados al Intent. Otro ejemplo para aclarar su funcionalidad, es donde una actividad se declara capaz de visualizar imágenes JPG pero no así para PNG.

Los *Intent* e *IntentFilters* son una manera de encapsular tareas, aislar dependencias y asegurar la reutilización de funcionalidades.

- Cada aplicación es responsable únicamente de su funcionamiento, sabiendo que para realizar cualquier tarea que no sea de su ámbito únicamente debe lanzar un *Intent*. De esta forma podemos construir aplicaciones centradas en una única actividad y cuyo trabajo puede ser reaprovechado por otras aplicaciones.
- Los *Intents* sirven como interfaz de comunicación entre aplicaciones, el equivalente a una API. Cualquier aplicación puede lanzar su *Intent* sin necesidad de conocer ningún detalle sobre la aplicación que lo va a resolver. No hay dependencia entre aplicaciones, cualquier actividad puede ser reemplazada por otra que tenga sus mismos *IntentFilter* y el reemplazo es transparente para todos los procesos que usan la funcionalidad original.
- Cuando una aplicación registra su *IntentFilter* y por tanto se declara capaz de manejar una tarea, esta aportando una funcionalidad que el resto de aplicaciones ya no necesitan implementar. Por tanto su código será reutilizado por todas las aplicaciones que lo necesiten para ejecutar el *Intent* deseado, cumpliendo así con una de las premisas básicas de la programación orientada a objetos, la reutilización.

4.4.1 - Objeto Intent

El objeto *Intent* se compone de tres elementos principales: acción, categoría y datos. Adicionalmente se pueden incluir un conjunto de elementos opcionales, que dependen y serán considerados según el "action" del Intent.

- Acción: es una cadena de texto completamente cualificada que indica la acción a realizar. Por ejemplo, android.intent.action.VIEW.
- Categoría: Son meta datos referidos al *Intent*, por ejemplo, android.intent.category.LAUNCHER nos indica que el *Intent* puede iniciar una aplicación si así es requerido.
- Datos: están definidos en forma de un objeto URI o texto plano.

La ejecución de un *Intent* es implícita: La actividad indica la tarea que desea realizar y el sistema asigna el *Intent* a la aplicación más apropiada. Otra forma de proceder es que la actividad indique explícitamente el destinatario del *Intent*, en este caso se tratará de una invocación explícita.

Esto último no es recomendable ya que estamos creando una dependencia innecesaria, y el solo hecho de desinstalar el paquete destino provocará errores en la aplicación emisora del *Intent*.

4.5 - ANDROID SDK

Android tiene un completo SDK (debuguer, APIs, bibliotecas, un emulador, documentación, código de ejemplo y tutórales) para facilitar el desarrollo de nuevas aplicaciones. Actualmente la plataforma de desarrollo esta disponible para Linux, Mac, y Windows.

A continuación se listan algunas de las características más representativas de esta plataforma, enfocadas a ventajas y desventajas.

Entre las ventajas se pueden describir:

- Plataforma libre
- Permite creación de aplicaciones por parte de terceros
- Prioridad equitativa entre aplicaciones nativas del sistema y aplicaciones de terceros
- Creación de aplicaciones rápida y fácil
- Proporciona SDK gratuito para el desarrollo de aplicaciones

Desventajas:

- Varias fueron las críticas sobre Android, sobre que no es un Sistema totalmente Abierto, y que Google quiera controlar el sistema siendo que alegan ser un sistema abierto. Puesto que no todo los fuentes y recursos del sistema está disponible.

Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, aplicaciones ofimáticas, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo. En otras palabras, Android quiere mejorar y estandarizar el desarrollo de aplicaciones para cualquier dispositivo móvil y, por ende, acabar con la perjudicial fragmentación existente hoy día.

Android es la primera plataforma "verdaderamente" abierta y amplia para dispositivos móviles. Incluye todo el software para que funcione un teléfono móvil, pero sin los obstáculos propietarios que han impedido la innovación en el teléfono móvil. El resultado en última instancia será un ritmo de innovación más rápido y mejor que proporcionará a los clientes móviles aplicaciones y capacidades hasta ahora difíciles de conseguir. Android es una estrategia importante para acercarnos al objetivo de Google de proporcionar acceso a la información a los usuarios donde quiera que estén.

Componentes de SDK

Android ofrece un plugin para Eclipse que extiende la funcionalidad de éste y facilita el desarrollo de aplicaciones para Android. Además, ofrece las herramientas que utiliza este plugin como scripts de Ant para que puedan ser utilizados también desde otros entornos como Netbeans o Intellij IDEA15. Entre las funcionalidades de este plugin se encuentra:

- Emulador de Android. Permite elegir entre distintos terminales móviles y la versión del sistema operativo.
- El acceso a herramientas de desarrollo de Android como tomar capturas de pantalla, la redirección de puertos, la posibilidad de depurar con puntos de parada o ver el estado de las hebras y los procesos corriendo en el sistema).
- Asistentes para la creación rápida de aplicaciones Android.
- Editores de código para los distintos archivos de configuración (XML) que facilitan su comprensión y desarrollo.
- Interfaces gráficas que permiten el desarrollo de componentes visualmente.

4.6 - COMPONENTES DE UNA APLICACIÓN ANDROID

El software de Android está basado en componentes independientes que interactúan entre sí. En una aplicación con audio, por ejemplo, tendremos al menos dos componentes: uno ejecutándose en segundo plano reproduciendo el audio, y otro en primer plano gestionando la interfaz de usuario. Es importante destacar que estos componentes son independientes, ya que podría haber una aplicación de terceros que también gestione ese componente de audio en segundo plano, o nosotros usar un componente que exponga un tercero.

Dentro de una aplicación de *Android* hay cuatro componentes principales: *Activitity,* Broadcast Receivers*,* Services *y Content Provider*. Todas las aplicaciones de *Android* están formadas por algunos de estos elementos o combinaciones de ellos.

Actividades (Activity)

Una Actividad se corresponde con una pantalla de la aplicación, es decir, una aplicación consistirá en un conjunto de actividades independientes que trabajan juntas, de las cuales se marca como la inicial al arrancar una aplicación. Para implementarlas se utiliza una clase por cada Actividad que extiende de la clase base *Activity*. Cada clase mostrará una interfaz de usuario, compuesta por *Views* (o Vistas). Cada Actividad es responsable de mantener su estado, de forma que puedan integrarse en el ciclo de vida de la aplicación, que es gestionado por el propio framework de aplicación. En Android se puede crear las interfaces de usuario de dos formas, desde la propia actividad usando código Java, o usando un fichero XML (que se verá más adelante) para describirla como si fuera una página HTML. Esta última es la manera más sencilla y cómoda.

Servicios (Services)

Un servicio no tiene una interfaz de grafica (UI), sino que se ejecuta en background por un período indefinido de tiempo, cada servicio extiende de la clase Service. Ejemplo: Un reproductor de música que se ejecuta en background permitiendo que se realicen otras actividades como enviar mensaje de texto (SMS) o ejecutar alguna otra aplicación mientras se escucha música.

Receptores de eventos (Broadcast Receivers)

Los receptores de eventos no realiza ninguna acción por sí mismo, recibe y reacciona ante anuncios de tipo broadcast, ejemplo: batería baja. No tienen UI, aunque pueden iniciar una actividad para atender al anuncio. Todos los receiver extienden de la clase BroadcastReceiver.

Proveedores de contenido (Content Providers)

Un proveedor de contenido permite que una aplicación ponga contenido a disposición de otra. Los datos pueden ser almacenados en un sistema de archivos, en una base de datos SQLite, o en cualquier otra forma que considere. El acceso a esos datos se hace a través del API definida por cada proveedor de contenidos. Un ejemplo de proveedor de contenidos que ofrece Google son los contactos que se tengan almacenados en nuestro teléfono.

4.6.1 - Activación de componentes: Intents (Intención)

Los Content Provider son activados al recibir una petición de un ContextResolver. Los otros tres componentes – Actividades, servicios y Broadcast Receivers, son activados por mensajes asincrónicos llamados Intents (Intención).

Es el elemento básico de comunicación entre los distintos componentes Android. Se pueden entender como los mensajes ó peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc. Un Intent es un objeto mensaje y que, en general, describe que quiere hacer una aplicación. Las dos partes más importantes de un Intent son la acción que se quiere realizar y la información necesaria que se proporciona para poder realizarla, la cual se expresa en formato URI.

Relacionado con los *Intents* hay una clase llamada *IntentFilter* que es una descripción de que *Intents* puede gestionar una Actividad. Mediante los *IntentFilters*, el sistema puede resolver *Intents*,

buscando cuales posee cada actividad y escogiendo aquel que mejor se ajuste a sus necesidades. El proceso de resolver *Intents* se realiza en tiempo real, lo cual ofrece dos beneficios:

- Las actividades pueden reutilizar funcionalidades de otros componentes simplemente haciendo peticiones mediante un *Intent.*
- Las actividades pueden ser remplazadas por nuevas actividades con *IntentFilters* equivalentes.

4.6.2 - Ciclo de vida de las actividades

En Android, cada aplicación se ejecuta en su propio proceso. Esto aporta beneficios en cuestiones básicas como seguridad, gestión de memoria, o la ocupación de la CPU del dispositivo móvil. Android se ocupa de lanzar y parar todos estos procesos, gestionar su ejecución y decidir qué hacer en función de los recursos disponibles y de las órdenes dadas por el usuario.

Android lanza tantos procesos como permitan los recursos del dispositivo. Cada proceso, correspondiente a una aplicación, estará formado por una o varias actividades independientes (componentes *Activity*) de esa aplicación. Cuando el usuario navega de una actividad a otra, o abre una nueva aplicación, el sistema duerme dicho proceso y realiza una copia de su estado para poder recuperarlo más tarde. El proceso y la actividad siguen existiendo en el sistema, pero están dormidos y su estado ha sido guardado. Es entonces cuando crea, o despierta si ya existe, el proceso para la aplicación que debe ser lanzada, asumiendo que existan recursos para ello.

Cada uno de los componentes básicos de Android tiene un ciclo de vida bien definido; esto implica que el desarrollador puede controlar en cada momento en qué estado se encuentra dicho componente, pudiendo así programar las acciones que mejor convengan.

El componente *Activity*, probablemente el más importante, tiene un ciclo de vida como se muestra en la siguiente figura:

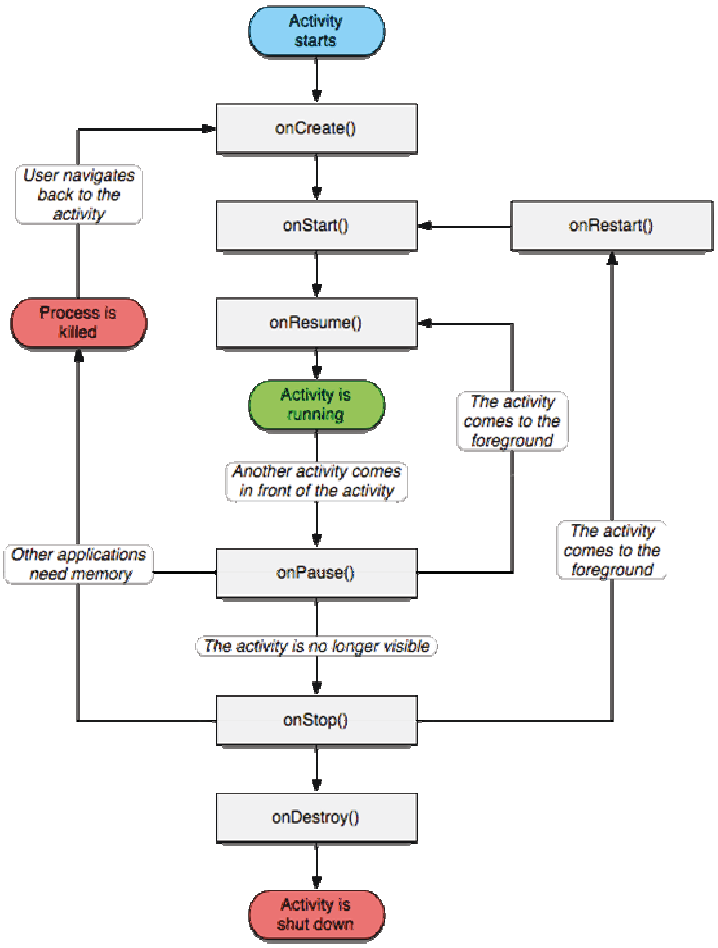


Figura 13. Ciclo de vida de una actividad Android

De la siguiente figura se puede determinar lo siguiente:

- *onCreate(), onDestroy():* abarcan todo el ciclo de vida. Cada uno de estos métodos representan el principio y el fin de la actividad.
- *onStart(), onStop():* representan la parte visible del ciclo de vida. Desde onStart() hasta onStop(), la actividad será visible para el usuario, aunque es posible que no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Pueden ser llamados múltiples veces.
- *onResume(), onPause():* delimitan la parte útil del ciclo de vida. Desde onResume() hasta onPause(), la actividad no sólo es visible, sino que además tiene el foco de la acción y el usuario puede interactuar con ella.

Estados de una actividad

- Activo (Running) cuando está en primer plano en pantalla, es visible tiene el foco. La actividad está encima de la pila de actividades.
- Pausado (Paused) la actividad ha perdido el foco pero es visible por el usuario. Se alcanza este estado cuando pasa a activa otra actividad transparente o que no ocupa toda la pantalla. Cuando una Actividad es tapada por completo pasa a estar parada.
- Parado (Stopped) Cuando la actividad no es visible. Se recomienda guardar el estado de la UI, preferencias, etc.
- Destruido (Destroyed): Cuando la Actividad termina, o es matada por el runtime de Android. Sale de la Pila de Actividades.

En la clase Activity existen métodos para ser redefinidos (override) por sus subclases que incluyen el código a ejecutar en las transacciones entre estados. Estos métodos son los siguientes:

- onCreate(): Se invoca cuando la Actividad se arranca por primera vez. Se utiliza para tareas de inicialización como crear la interfaz de usuario de la Actividad
- onStart(): Se invoca cuando la Actividad va a ser mostrada al usuario
- onResume():Se invoca cuando la actividad va a empezar a interactuar con el usuario
- onPause():Se invoca cuando la actividad va a pasar al fondo porque otra actividad ha sido lanzada para ponerse delante. Se utiliza para guardar el estado de la Actividad.
- onStop() Se invoca cuando la actividad va a dejar de ser visible. Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la actividad ser destruida directamente.
- onRestart() Se invoca cuando una actividad parada pasa a estar activa
- onDestory() Se invoca cuando la Actividad va a ser destruida. Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la actividad ser destruida directamente.

4.6.3 - Política de eliminación de procesos en Android

Cada aplicación de *Android* corre en su propio proceso, el cual es creado por la aplicación cuando se ejecuta y permanece hasta que la aplicación deja de trabajar o el sistema necesita memoria para otras aplicaciones. Una característica fundamental de *Android* es que el ciclo de vida de una aplicación no está controlado por la misma aplicación sino que lo determina el sistema a partir de una combinación de estados como pueden ser que aplicaciones están funcionando, que prioridad tienen para el usuario y cuanta memoria queda disponible en el sistema. De esta manera, *Android* sitúa cada proceso en una jerarquía de importancia basada en los componentes en ejecución y en el estado de estos.

Procesos de menor importancia se eliminan primero, luego los siguientes en importancia, y así sucesivamente. Hay cinco niveles en la jerarquía. El orden de importancia sería el siguiente:

1. Procesos activos (foreground): aquellos necesarios para lo que el usuario está haciendo en ese momento. Un proceso se encuadra en esa categoría si cumple alguna de las siguientes condiciones:
    a. Una actividad de estado activo, es decir, está en primer plano y responde a eventos del usuario.

   b. Esta ejecutándose un componente *Broadcast Intent Receiver*. Actividades, Servicios, o *Broadcast Receiver* que se está ejecutando actualmente un manejador de eventos onReceive().
   c. Esta ejecutándose un componente *Service*.
2. Procesos visibles: es aquel que contiene una Actividad que es visible al usuario mediante la pantalla pero no en primer plano (esta pausada). Este proceso solo se eliminará en caso de que sea necesario para mantener ejecutándose los procesos en primer plano.
3. Procesos de Servicio: es aquel que contiene un servicio que ha sido inicializado. No son directamente visibles al usuario, desempeñan tareas sí percibidas por este. El sistema los mantendrá a no ser que no pueda servir los dos siguientes.
4. Procesos en segundo plano (background): procesos con un componente *Activity*, que no son visibles al usuario. Estos procesos no tienen una importancia directa para el usuario en ese momento. Mientras que dichos procesos implementen bien su propio ciclo de vida, el sistema puede eliminarlos para dar memoria a cualquiera de los 3 servicios anteriores
5. Procesos Vacíos: es aquel que no contiene ningún componente activo de ninguna aplicación. La única razón para mantener dicho proceso es para mejorar sus inicializaciones posteriores a modo de caché.

Según esta jerarquía, Android prioriza los procesos existentes en el sistema y decide cuáles han de ser eliminados, con el fin de liberar recursos y poder lanzar la aplicación requerida.

4.7 - TIPOS DE APLICACIONES ANDROID

Un *Android Package* (.apk) es un archivo que contiene código de la aplicación y sus recursos, que posteriormente se instala en el dispositivo para poder ejecutar la aplicación.

Estos paquetes distribuibles pueden contener distintas funcionalidades y ser para fines muy diversos, pero en cuanto a tipo de tareas que ejecutan, se dividen en cuatro grupos

Tareas

Una tarea en *Android* es lo que el usuario ve como una aplicación y el desarrollador ve como una o más actividades donde el usuario interacciona y va pasando de pantalla en pantalla.

Dentro de las tareas, una actividad toma el papel de punto de entrada (será la primera en mostrarse cuando se ejecute la aplicación) y las demás, si hay, formarán parte de la misma tarea, a la espera de ser instanciadas.

Procesos

En *Android,* los procesos se ejecutan a nivel de *kernel* y el usuario normalmente no tiene constancia de ellos. Todo el código de la aplicación se suele ejecutar en un proceso dedicado pero también se puede especificar si sólo se quiere que se ejecute en el proceso una determinada clase o componente de la aplicación.

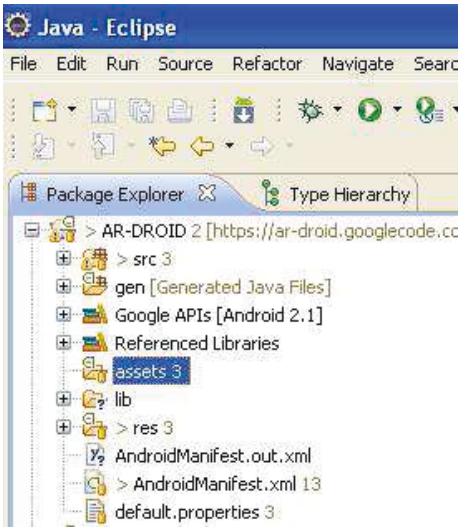
Threads

En lo referente a *threads*, *Android* evita la creación de *threads* adicionales por parte de un proceso, manteniendo la aplicación en un sólo *thread* al menos que no los cree la propia aplicación. Esto repercute de manera importante en las llamadas a instancias a actividades, *Listeners* y servicios, ya que sólo pueden ser hechas por el *thread* principal del proceso en el que están corriendo.

Por otro lado, al no crearse un *thread* por cada instancia, dichas instancias no deben realizar operaciones largas o bloqueantes cuando son llamadas, de lo contrario, bloquearían todos los demás componentes del proceso.

4.8 - ESTRUCTURA DE UNA APLICACIÓN ANDROID

Al crear una aplicación Android en Eclipse se genera automáticamente una un proyecto con la estructura de directorios necesaria como muestra la siguiente imagen. Esta estructura será común a cualquier aplicación, independientemente de su tamaño y complejidad.



Estructura de un proyecto Android

Un proyecto contiene una serie de subcarpetas y de ficheros que constituyen la anatomía completa de un proyecto Android.

Carpeta /src/

*C*ontiene el código fuente de la aplicación. El sistema de paquetes es igual que el de java.

Carpeta /res/

Esta carpeta almacena todos los ficheros de recursos necesarios para la aplicación desde imágenes, videos hasta elementos .xml que compondrán el Interfaz de Usuario y la funcionalidad del mismo.

Cada uno de los recursos utilizados en una aplicación Android debe estar localizado en la carpeta adecuada, en función de su naturaleza. De esta forma, dentro de la carpeta de recursos "/res" se pueden encontrar las siguientes subcarpetas:

- /drawable/. Contiene las imágenes (íconos) de la aplicación. Por ejemplo, imágenes con formato JPG, PING o GIF. Se puede dividir en /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para utilizar diferentes recursos dependiendo de la resolución del dispositivo.
- /layout/. *Contiene* archivos XML que describen la interfaz del usuario. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación del dispositivo.
- /menú/ .Contiene la definición de los menús de la aplicación.
- */values/.* Archivos XML que definen valores de distintos tipos. Por ejemplo, cadenas de texto, colores predefinidos, arrays de elementos, dimensiones, estilos, etc. Por convención, existirá dentro de esta carpeta un fichero XML por cada tipo distinto de recurso que se declare: "strings.xml" para las *cadenas de texto*, "colors.xml" para los colores, "dimens.xml" para las dimensiones, "styles.xml" para los estilos, etc.
- /anim/. Contiene la definición de las animaciones utilizadas por la aplicación.
- /res/xml/. Contiene los ficheros XML utilizados por la aplicación.
- /res/raw/. Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos. Archivos multimedia de audio y video.

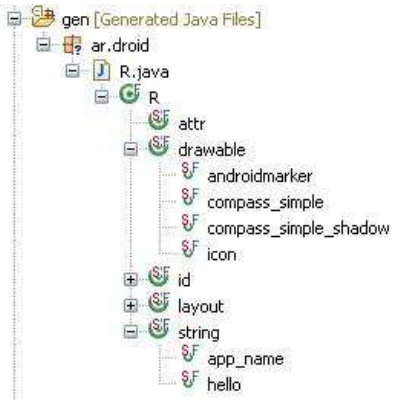
Paquete de recursos de un proyecto Android

Carpeta /gen/

Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto, la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente en java, dirigidos al control de los recursos de la aplicación.

El más importante es el que se puede observar en la imagen de abajo, el fichero R.java, y la clase R.

Esta clase R contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta /res/, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato. Así, por ejemplo, la constante R.drawable.icon contendrá el ID de la imagen "icon.png" contenida en la carpeta /res/drawable/.



Veamos como ejemplo la clase R creada por defecto para un proyecto nuevo

```
package com.example.android.maps;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int mainlayout=0x7f050000;
        public static final int mapview=0x7f050001;
        public static final int zoomview=0x7f050002;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Carpeta /assets/

Contiene archivos externos que necesite la aplicación (mp3, pdf, jar, etc).

La diferencia entre los recursos incluidos en la carpeta /res/raw/ y los incluidos en la carpeta /assets/ es que para los primeros se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Para los segundos sin embargo no se generarán ID y se podrá acceder a ellos por su ruta como a cualquier otro fichero del sistema. Usaremos uno u otro según las necesidades de nuestra aplicación.

Archivo AndroidManifest.xml

Este archivo es llamado AndroidManifest.xml para todas las aplicaciones, debe estar ubicado en la raíz de la aplicación, en el se declaran varias cosas como: componentes de la aplicación, permisos (cámara, Internet, etc.), librerías que se utilizan, el nivel mínimo de la API de Android que la aplicación requiere, recursos necesarios, pero la principal tarea de este archivo es informar de los componentes de la aplicación Android.

4.9 - HERRAMIENTAS DEL SDK DE ANDROID

El SDK de Android incluye una variedad de herramientas que ayudan al desarrollo de aplicaciones móviles. A continuación se describirán las herramientas más importantes que nos proporciona el SDK [37].

Emulador Android

El SDK de Android incluye un completo emulador que se puede utilizar para diseñar, depurar y testear las aplicaciones en un ambiente similar al que existe en un dispositivo real.

El emulador es una implementación de la máquina virtual Dalvik, lo que lo convierte en una plataforma válida para el funcionamiento de aplicaciones de Android como cualquier teléfono Android. Debido a que es disociado de cualquier hardware en particular, es una línea de base excelente para usar para probar las aplicaciones.

Permite simular varios eventos como llamadas entrantes o mensajes SMS, incluso conectar varios emuladores entre sí, ubicaciones del GPS.

Cuando el emulador se ejecuta, puede interactuar con el emulador del dispositivo móvil del mismo modo que un dispositivo móvil real, salvo que se debe utilizar el puntero del ratón para "tocar" la pantalla táctil y se puede utilizar algunas teclas del teclado para invocar ciertas teclas en el dispositivo.

Se pueden correr varias instancias del emulador, cada una debe tener su configuración AVD.

Figura 14. Emulador corriendo sobre Windows XP

Android Virtual Device (AVD)

Un AVD [41] es una configuración de las opciones del emulador que permite un mejor modelo del dispositivo real. En cada configuración se puede especificar la plataforma donde se ejecutara, las opciones de hardware como por ejemplo, si tiene cámara, cantidad de memoria, si tiene teclado QWERTY, GPS, etc. Se puede seleccionar el skin que se desea utilizar con el AVD, lo que permite controlar las dimensiones de la pantalla, apariencia y demás. Cada AVD funciona como un dispositivo independiente con su propio almacenamiento de datos, tarjeta SD.

Plugins de Desarrollo para en Eclipse (ADT- Android Development Tools)

El "ADT plugin" [40] agrega poderosas extensiones al ambiente integrado de Eclipse haciendo que la creación y depuración de las aplicaciones Android sea fácil y rápida. A continuación alguna de sus características:

- Provee acceso a otras herramientas de desarrollo de Android desde el entorno de Eclipse IDE.
- Provee un asistente para la creación de proyectos Android, el cual te ayuda a crear rápidamente todos los directorios y archivos necesarios para crear una nueva aplicación Android.
- Automatiza y simplifica el proceso de construcción de una aplicación.

- Provee un editor de código Android que te ayuda a escribir XML válido para el archivo "AndroidManifest.xml".

Dalvik Debug Monitor Service (DDMS)

La "Dalvik Debug Monitor Service" [38] es una herramienta integrada con la "Dalvik Virtual Machine", y nos permite administrar los procesos que corren en distintos emuladores o dispositivos y nos asiste en la depuración de ellos.

Esta aplicación permite hacer un seguimiento de los puertos utilizados, ver mensajes de Log, información acerca de los procesos en ejecución, gestión de memoria, eventos entrantes como llamadas y SMS, así como otras muchísimas opciones orientados a la depuración.

El DDMS monitoriza vía el ADB instancias de la maquina virtual. Con el plugin ADT provisto para Eclipse se pueden ver de forma integrada todas las funcionalidades que brinda del DDMS.

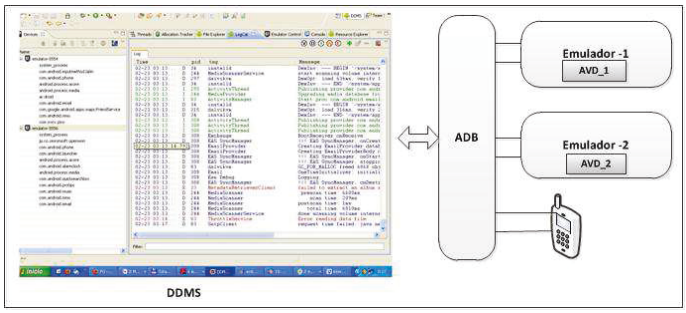


Figura 15. Interacción Android DDMS

Android Debug Bridge (ADB)

Es una versátil herramienta que permite administrar el estado de una instancia del emulador o dispositivo con Android. Es una aplicación cliente-servidor que se compone de:

1. Daemon: Proceso background en el emulador o dispositivo.
2. Cliente: Se ejecuta en la máquina de desarrollo. Se puede comunicar desde adb Shell, ADT plugin o DDMS.
3. Servidor: Corre en un proceso background en la máquina de desarrollo, gestión la comunicación entre el cliente y el adb daemon.

Es como un Conductor de comunicación entre la máquina de desarrollo y el dispositivo o emulador Android, el ADB [39] permite instalar aplicaciones, ejecutar comandos en la maquina desarrollo, consultar o modificar bases de datos SQLLite disponibles en el dispositivo, etc.

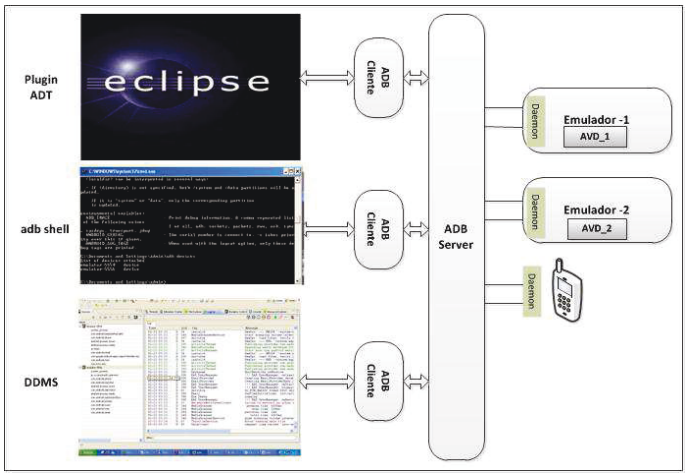


Figura 16. Estructura de Android ADB

Cualquiera que haya programado para otras plataformas, como Windows Mobile, sabe lo complicado que es todo el proceso. En especial, además de tener un modelo de aplicación bastante complejo, el propio Microsoft ponía obstáculos a los desarrolladores de software libre con su modelo de firmado y restricciones de capacidades.

En Android todo es muy diferente. El propio sistema es libre, y tanto el entorno de desarrollo como el framework son bastante más simples en cuanto se logra algo de práctica.

4.9.1 - Dalvik Virtual Machina

En Android, todas las aplicaciones se programan en el lenguaje Java y se ejecutan mediante una máquina virtual de nombre Dalvik, específicamente diseñada para Android. Esta máquina virtual ha sido optimizada y adaptada a las peculiaridades propias de los dispositivos móviles (menor capacidad de proceso, baja memoria, alimentación por batería, etc.) y trabaja con ficheros de extensión .dex (Dalvik Executables). Dalvik no trabaja directamente con el bytecode de Java, sino que lo transforma en un código más eficiente que el original, pensado para procesadores pequeños.

Diferencias respecto a la MV de Java

Las máquinas virtuales de Java, disponibles en la mayoría de los ordenadores hoy en día, están basada en el uso de pilas. Por el contrario, la Dalvik VM está basada en registros. Esto es debido a que los procesadores de los dispositivos móviles están optimizados para la ejecutar sentencias basadas en registros. Además se sabe, las máquinas virtuales basadas en registros permiten una ejecución más rápida, a costa de programas que ocupan más espacio tras la compilación.

Vale aclarar que a pesar de usar Java para la programación, el bytecode de Java no es ejecutable en un sistema Android.

Por otra parte, las librerías Java que utiliza Android son ligeramente distintas de las incluidas en Java Standard Edition (Java SE) o en Java Mobile Edition (Java ME). A pesar de las ligeras diferencias en su mayor parte el contenido de las librerías es idéntico.

Optimización

Dalvik VM es capaz de reducir en una parte el tamaño del programa buscando información duplicada en las distintas clases y reutilizándola.

El *garbarage collector* (liberar el espacio de objetos ya no accesibles) ha sido perfeccionada a fin de mantener siempre libre la máxima memoria posible.

Android hace un uso de archivos XML como forma de definir las interfaces gráficas y otros elementos gráficos. Los XML tienen que ser linkeados a la hora de compilar y son convertidos a bytecode para mejorar el rendimiento.

4.10 - CUADRO COMPARATIVO DE ANDROID, IPHONE Y WINDOWS MOBILE

Con Android, Google ha hecho una apuesta arriesgada entrando en un mundo en el que ya brillan con luz propia Nokia con Symbian, Apple con su IPhone, BlackBerry o Windows Mobile.

Hace poco se presentaba una nueva versión de iPhone, la 4.0, y hacia finales de año Microsoft prometió su renovado sistema para móviles, el Windows Phone 7. Mientras tanto, Android está poniéndose a la altura del iPhone de Apple y la decisión para los usuarios finales no es fácil.

Para evaluar y comparar los dispositivos Apple y Android, se tienen en cuenta unas 20 propiedades y en cada una de estas categorías daremos un ganador. Por ejemplo, entre muchas otras, se consideran:

- La facilidad de uso
- La duración de la batería
- Multitarea
- Teclado
- Juegos
- Reproductor de música

A continuación se presenta como una tabla comparativa, más adelante se ampliarán las comparaciones. Vale aclarar que se omiten las comparaciones con WP7 en detalle, pues a esta altura son estimadas, ya que ni siquiera hay una beta pública disponible para probar. Se podría haber hecho comparación con el viejo Windows Mobile 6.5, pues aún es grande la oferta de los dispositivos con este sistema, pero, al igual que BlackBerry OS, realmente ha quedado muy relegado en comparación a sus competidores.

	iPhone OS 4.0	Android 2.2	Windows Phone 7
Kernel	iPhone OS X	Linux Kernel	Windows CE 6
Multitarea	Si	Si	Si
Multitouch	Si	Si	Si
Internet tethering	Bluetooth y USB (mediante jailbreak)	USB + Wi-FI	Desconocido
Copiar y Pegar	Si	Si	Limitado
Teclado	Sólo Teclado virtual en pantalla	Soporta Teclado virtual en pantalla y teclado físico	Soporta Teclado virtual en pantalla y teclado físico
Tienda de música	Propia - iTunes	de terceros	Zune - propia
Mapas con navegacion GPS gratis	No	Google Maps	Bing Maps
Juegos	gran variedad de juegos en 3D disponibles App Store	Pocos pero con calidad 3D	Xbox Live
Redes sociales para jugeadores	Si	Si	Si
Soporte de Carpetas	Si	Si	Desconocido
Books	iBooks	No	No
Actualización de aplicaciones Over-the-Air	No	Si	No
Sincronización inalámbrica con la PC/Mac	No, pero es posible mediante el jailbreak	Desconocido	Si
Suite Office	iWork, QuickOffice	Google Docs	Office 2010 Mobile
Navegador Web	Safari	Dolphin	Internet Explorer
Soporte de Flash	No	Si	No inicialmente
Silverlight	No, Nunca	No, posiblemente	Si, de hecho
Buscador	Google	Google	Bing
Agrupamiento de correo	Mail App	Gmail App	Outlook Mobile
Pantalla de inicio personalizable	Si, limitado	Si	Si
Actualización del sistema operativo	Si	Si	Si
Actualización OTA	No	Si	Si
Temas, plantillas, skin	Limitado, pero completo via jailbreak	Si	No
Servicio en las nubes	Mobile Me	Google Apps	My Phone 2.0
Herramienta de desarrollo	Iphone OS SDK / Mac	Android SDK / Windows, Mac y Linux	Windows Phone 7 SDK / Windows
Disponibilidad de aplicaciones	170000 +	50000 +	Desconocido

Tabla 6. Cuadro comparativo de Android, iPhone 4 y WP7

Facilidad de Uso: *iPhone es el ganador*

Android se acerca mucho a la usabilidad que está ofreciendo Apple en su dispositivo, pero no lo suficiente, aun le falta madurez en esto. La interfaz del iPhone es lo más sencilla posible y el usuario es capaz de aprender a usar el móvil desde cero mucho más rápido que si se tratara de un Android.

Libertad del software: *Android es el ganador*

Es cierto modo, lo de ser software abierto gusta a todo tipo de usuarios, y también vende mucho. Lo más importante de que Android sea un sistema operativo "abierto", es que permite disponer un una mayor cantidad software. Porque las aplicaciones son lo más importante de un smartphone.

Duración de la batería: *iPhone es el ganador*

Apple sabe perfectamente lo que es importante para sus usuarios y se han tomado el tema de las baterías muy en serio. En el caso de Android, todo depende de qué dispositivo estamos hablando, ya que el hardware Android varía muchísimo. Por ejemplo, el modelo EVO lanzado recientemente ha recibido

duras críticas sobre la duración de su batería, y el modelo HTC Dimond también fue un fracaso en estos términos.

Software del Teclado: *IPhone es el ganador*

Se ha hablado mucho sobre la superioridad del sistema de teclado de iPhone, y eso son malas noticias para los usuarios de Android, sin embargo, en Android podemos instalar cualquier teclado personalizado. Pero en definitiva, se está hablando de las funcionalidades que vienen de fabrica, iPhone es poco mejor también en esta categoría. Igualmente basta con dar con el software de teclado que agrade al usuario para al menos igualar en este aspecto.

Reconocimiento de voz (Voice-to-text): *Android es el ganador*

Casi todos los campos de texto que aparecen en el sistema Android pueden ser completados usando la voz, y la verdad es que funciona realmente bien (mecho mejor cuando la voz es inglesa). Esto significa que se puede también escribirse mensajes o emails usando simplemente nuestra voz, cosa que no puede hacerse en iPhone.

Sincronización: *empate.*

Los smartphones de Android pueden sincronizarse con todos los servicios de Google, y claro, eso es una ventaja muy grande respecto a iPhone. Se puede actualizar el móvil automáticamente con todos los datos de tus cuentas de Google. Por otro lado, si olvidamos la sincronización con Google, Apple es más compatible con los sistemas de sincronización de programas populares como Outlook, iTunes o Agendas de Contactos conocidas.

Gaming: *iPhone es el ganador*

Si se es un apasionado de los videojuegos ya sabe, el móvil de Apple dispone de aplicaciones de juegos más potentes que las actuales de Android. Sin embargo, también es cierto que los títulos de juegos están aumentando en el Android Market y seguramente, pronto ambos dispositivos igualarán también en esta categoría.

Reproductor de música: *iPhone es el ganador*

Por el momento no es tema de discusión, iPhone integra una aplicación iPod muy sólida y los usuarios de Android no disponen de buenos reproductores por defecto.

Personalizable: *Android es el ganador*

Se puede cambiar el fondo de pantalla del iPhone, pero se podrá hacer poco más para personalizar la interfaz del iOS4. Sin embargo, con Android se tiene mucha más libertad, de hecho, yo se dice que es casi absoluta. Android cuenta con muchísimas aplicaciones de terceros para mejorar y aumentar la interfaz de usuario.

4.8.1 - Entornos de desarrollo Android e iPhone

La proyección en el desarrollo de aplicaciones para Android es mayor que para el iPhone, y en cierto modo es que hay mucho más por hacer, pero todo parece suponer que Android podría erigirse como la plataforma más usada en smartphones. Cualquiera persona puede desarrollar para Android, creando aplicaciones con la funcionalidad que se le ocurra. La apertura de Android llega a la plataforma en sí, con licencia Apache, por lo que cualquier fabricante u operador puede utilizarla, adaptándola a sus necesidades sin pagar por ello.

Por su parte Apple ejerce control sobre iPhone, qué aplicaciones se pueden distribuir y hasta se reserva cierto derecho para desinstalarlas. Tanto Android como iPhone son plataformas con una visión basada en el teléfono como nueva puerta a la red, con el que estar siempre conectados a Internet y sobre las montar otros negocios. Donde son radicalmente opuestas es en la estrategia principal para lograr constituirse en una referencia: Android es abierto, se puede llevar a dispositivos de varios fabricantes, cualquiera puede desarrollar sobre él... y Apple apuesta por controlar la experiencia de usuario, ofreciendo una solución en la que crea o supervisa todos los aspectos de la misma.

A la hora de programar se aprecia una mayor sencillez en Android, empezando por el hecho que puedes desarrollar desde cualquier sistema operativo y que es poco probable que te encuentres con una negativa de la compañía para publicar tu aplicación. En el caso de Apple la cosa cambia, necesitas forzosamente contar con MAC OS y la aplicación desarrollada tiene que ser aprobada por los estrictos regímenes de Apple.

iPhone es programado con *Objective C*, un lenguaje tipo C, orientado a objetos, cuya elegancia es comparable a la de Java. El nuevo SDK permite acceso a los APIs del sistema operativo, pero no a modificar el sistema operativo mismo, por supuesto.

Android es programado en Java, pero dependiendo del proveedor, es posible programar directamente en C o hasta en Assembly (aunque esto no es recomendado). El proveedor (es decir, el que fabrique un celular Android) tiene acceso completo al sistema operativo y puede no solo crear *drivers* compatibles con el API de Android, sino que hasta nueva funcionalidad si así lo desea.

Ambas plataformas ofrecen un juego de librerías de APIs bastante completo, sin embargo en Android es posible tener acceso a más bajo nivel del sistema operativo, así como a dispositivos externos conectados por puertos de conexión de todo tipo.

El iPhone ofrece un SDK que es completo, que incluye un simulador, un debbuger y diagnosticador remoto, y un creador de interfaces gráficos. Android por otro lado, ofrece un SDK que es mayoritariamente dependiente de un IDE externo como Eclipse (aunque puede utilizarse en otros IDEs, como Netbeans). Sin embargo, ofrece un excelente simulador y tiene interfaces para debugear remotamente.

Además, como se dijo, funciona en varias plataformas. Por ahora su limitante es que no ofrece una manera gráfica de crear interfaces de usuario, aunque se espera que terceros ofrezcan tales herramientas.

5 - ESPECIFICACIÓN DEL PROTOTIPO

El objetivo de este proyecto es construir un prototipo de aplicación para plataformas móviles, más precisamente Android, que permita la consulta e interacción con lo que se refiere a toda la actividad sucedida en el ámbito académico.

Llamaremos a esta información proporcionada *servicios*. Estos servicios, proveerán información sobre distintos eventos que ofrecen los distintos departamentos de la universidad, como por ejemplo talleres, cursos, conferencias, etc.

El prototipo tendrá una arquitectura distribuida: por un lado el cliente móvil y por el otro el servidor que proveerá los servicios requeridos por el cliente. Básicamente podemos identificar la arquitectura del sistema en el siguiente grafico, aunque describiremos en detalle en los componentes más adelante.

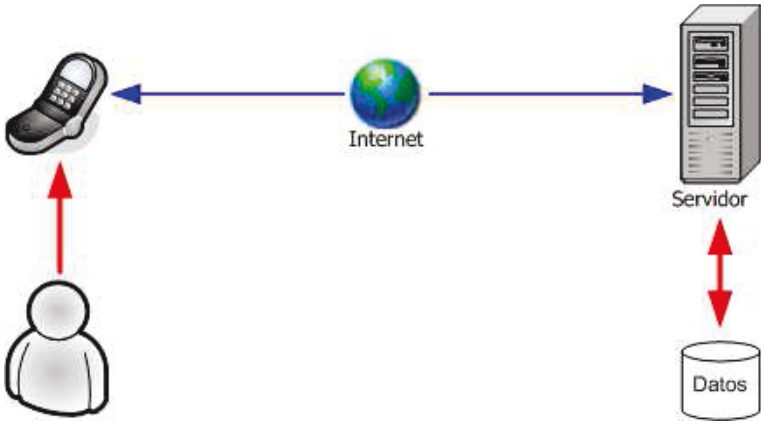


Figura 17. Arquitectura básica del sistema AR-DROID

En este tipo de sistemas distribuidos para dispositivos móviles, el cliente realiza algunas tareas, como la determinación de la ubicación y de presentación de resultados, mientras que el servidor (proveedor) realiza el resto del procesamiento. Vemos que los requisitos se hacen, por tanto, evidentes. El proveedor deberá disponer, en primer lugar, de una red de comunicación entre sus sistemas y los usuarios. En segundo lugar, necesita una arquitectura de servidores que se encarguen de recibir peticiones, procesarlas y brindar uno o un conjunto de resultados.

Descubrimos un modelo de datos inicial de la aplicación, donde identificamos los siguientes elementos:

- *Entidades* para modelar facultades, museos, bibliotecas, etc.
- *Eventos* para modelar los distintos acontecimientos que tienen lugar en las entidades.
- *Comentarios* y calificación sobre eventos a modo de retroalimentación para la aplicación.

Los servicios serán consumidos por la aplicación cliente en un dispositivo móvil que visualizara la información sobre un mapa de acuerdo a la ubicación del usuario. El usuario podrá consultar individualmente la información completa de las entidades y eventos con la posibilidad de interactuar con la aplicación a modo de red social brindando su opinión sobre los eventos.

La aplicación consumirá esta información en tiempo real haciendo uso de los distintos sensores con los que cuentan los dispositivos (como el GPS, el acelerómetro y el compás).

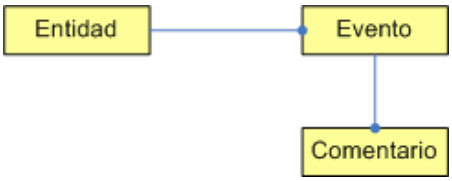
El prototipo incluirá una funcionalidad de realidad aumentada que a través de la cámara del dispositivo móvil será capaz de identificar y señalar en pantalla las unidades académicas que se encuentren registradas en el catalogo de entidades del proveedor de servicios. Este último se administrara desde una interfaz Web. Desde aquí se podrán gestionar las entidades físicas y la información de los distintos servicios brindados por cada una.

5.1 - CONCEPTOS DE LA APLICACIÓN

Podemos definir el contexto de la aplicación a través de diagramas de conceptos evolutivos, determinando las distintas iteraciones sucedidas para llegar a un modelo final para utilizar de base en la construcción del modelo de datos del prototipo.

Luego definiremos aspectos generales de lo que se quiere lograr para establecer los límites del sistema y sus funcionalidades.

A continuación definiremos el modelo de contexto; dijimos anteriormente que inicialmente identificamos tres conceptos principales:



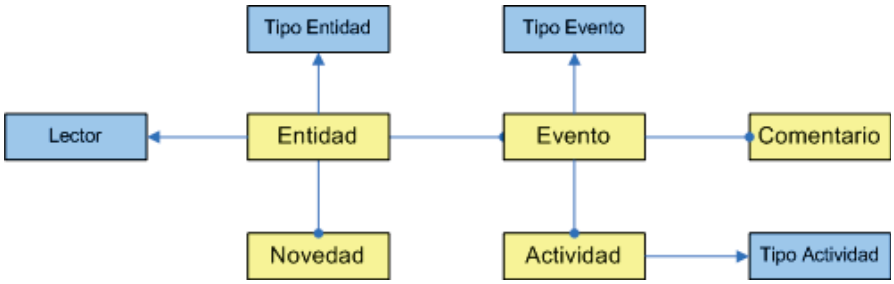
Existen eventos que ocurren en entidades en particular, que pueden involucrar distintos tipos de actividades en distintos lugares. Podemos tomar como ejemplo un algún tipo de seminario sobre un tema en particular: El evento sería "Seminario Realidad Aumentada" y este englobaría distintas actividades: una introducción teórica en el auditorio X, al otro día un taller en la sala de prácticos, y por ultimo una practica al aire libre en el parque de juegos.

Entonces definimos que un evento puede involucrar muchas actividades:

Luego necesitamos una forma de distinguir los distintos tipos de entidades: como sabemos que las entidades puede ser facultades, museos, teatros, etc. Le asociamos a cada una un *Tipo Entidad*. Lo mismo ocurrió con los eventos y las actividades: Necesitaremos una forma de filtrar los eventos y actividades para una mejor y más precisa presentación de datos al usuario, con la finalidad de entregarle solo lo que esta buscando y le interesa. Entonces definimos un evento tendrá un *Tipo Evento* y también se conocerá el *Tipo Actividad* para la/las actividad/es de ese evento.

De las entidades, además de los eventos nos interesaría conocer que esta sucediendo en esta, por lo que acceder a las novedades de las distintas entidades para mantenernos informados. Entonces conoceremos una lista de novedades para cada entidad.

Si vemos al prototipo como un sistema de información centralizada, estaremos limitando su funcionalidad y el éxito de la aplicación. Porque de esta manera quizás estemos obligando al administrador a duplicar la información de eventos y novedades para, por ejemplo, una facultad, al saber que muchas facultades distribuyen sus novedades a través de RSS o alguna red social. Por esto es que asociaremos a las entidades un par de lectores. Estos lectores, opcionales, serán los encargados de alimentar la información de eventos y novedades para las entidades. Las fuentes de datos pueden ser varias, como ser RSS(*), redes sociales (como Facebook o Twitter), entradas blogs, entre otras.



Para los eventos de una entidad vamos a necesitar saber su repetición. Suponiendo en un teatro todos los viernes a la noche se realiza la conocida "Noche de museo abierto", sería tedioso para el administrador cargar el mismo evento todas las semanas. Podemos definir para un evento su *Periodicidad*, pudiendo ser única (el evento es sucede únicamente una vez), diaria, semanal, mensual.

Por otro lado exploremos algo más con lo que se busca que el usuario devuelva a la entidad a manera de comentario. Talvez algunos eventos puedan requerir un feedback algo más complejo que un simple par de líneas por parte del usuario. Por ejemplo una presentación de un nuevo producto por parte de la empresa Oracle, puede requerir que el usuario de una opinión más especifica sobre el producto y/o sobre el presentador. Entonces el administrador debe tener la capacidad de definir una *Encuesta* para un evento, que el usuario completará eventualmente.

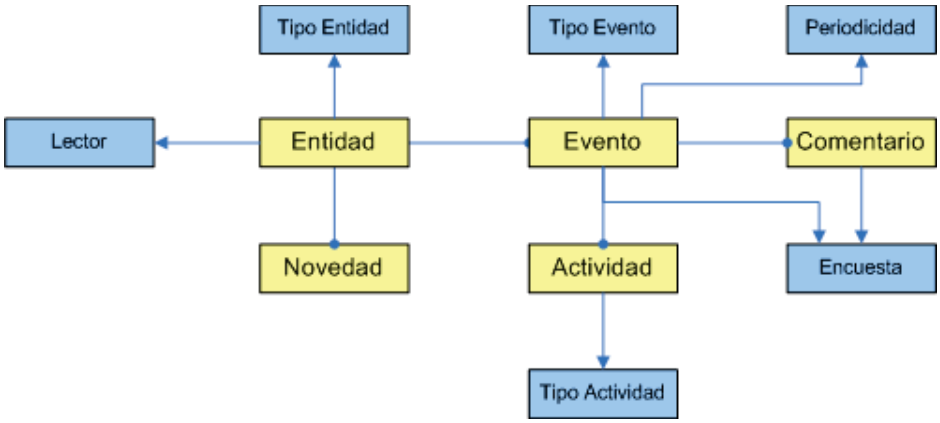


Figura 18. Diagrama de conceptos de AR-DROID

5.1.1 - Descripción de Conceptos

- ENTIDAD: Una entidad representa un lugar físico donde tienen lugares los eventos y actividades. Representan facultades, museos, salones, teatros, etc. Conoceremos de estos su nombre, descripción, URL de la entidad, ubicación y una imagen representativa.
- EVENTO: Un evento es un acontecimiento que tiene lugar en las entidades, que puede ser un curso, una exposición, conferencia, etc. Un evento se referencia a una entidad, y se conocen titulo, descripción, ubicación, una imagen representativa, y la fecha de ocurrencia.
- ACTIVIDAD: Las actividades forman parte de los eventos, donde un evento puede incluir muchas actividades. Una actividad esta formada por un nombre, descripción y ubicación.
- TIPO ENTIDAD: Un tipo de entidad determina y clasifica una entidad. A este simple concepto le asociaremos un nombre y una imagen.
- TIPO EVENTO: El tipo de evento clasifica un evento. Solo conoceremos el nombre para este concepto.
- TIPO ACTIVIDAD: Al igual que los anteriores, los tipos de actividades clasifican a las actividades. Le asignaremos un nombre y un color para identificarlas.
- PERIODICIDAD: La periodicidad determina la repetición de un evento. Esta puede ser única, diaria, semanal o mensual. Entonces conoceremos una fecha, un clasificador del tipo de periodicidad, y el parámetro propio al tipo (día de la semana, o día del mes).
- NOVEDAD: Las novedades son informaciones que se conocen de las distintas entidades. Estas no formaran parte del dominio de datos, sino que se obtendrán de fuentes de alimentación de las entidades.

- ENCUESTA: Una "encuesta" es una estructura configurable que se puede definir para cada evento. Esta estructura se le presentará al usuario en el momento de dar un feedback a un evento al que concurrió. La encuesta estará compuesta básicamente de un titulo y las preguntas.
- COMENTARIO: Un comentario es la respuesta por parte del usuario a un evento al que concurrió. Es la contestación a una encuesta sobre un evento. Tendremos del comentario la lista de respuestas a cada pregunta y un texto a modo de nota general.
- LECTOR: Un lector alimentará automáticamente los eventos y novedades para las entidades del sistema. Como dijimos tendremos al menos tres fuentes de datos: Facebook, el API del sistema(*), y Twitter. Del lector se conocerá la parametrización de conexión para obtener los datos desde las fuentes.

5.2 - LISTA DE REQUERIMIENTOS

Describiremos el prototipo a implementar a través de sus requerimientos. A estos los distinguiremos entre *Funcionales* y *No funcionales*. También necesitaremos una distinción entre las funcionalidades del cliente móvil, y las funcionalidades del administrador Web.

Los Funcionales son los que definen el comportamiento interno del sistema: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran como los casos de uso serán llevados a la práctica. Los No funcionales especifican criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos.

Esta lista de requerimientos definirá luego los Casos de uso.

5.2.1 - Requerimientos del cliente móvil

Funcionales:

ID	Nombre	Descripción
RFC1	Mostrar mapa	El cliente debe mostrar el mapa en pantalla provisto por Google Maps.
RFC2	Obtener ubicación	El cliente deberá automáticamente determinar la posición del usuario si tiene los mecanismos para hacerlo.
RFC3	Mostrar vista RA	El cliente debe mostrar la vista de realidad aumentada sobre la captura de video desde la cámara del dispositivo móvil.
RFC4	Mostrar detalle de un objeto	El sistema debe mostrar el detalle de una entidad o evento a petición del usuario. Esta funcionalidad debe permitirse tanto accediendo desde el mapa, como también desde la vista RA.
RFC5	Obtener entidades	El cliente debe recuperar las entidades registradas para visualizarlas en el mapa y en la vista de RA.
RFC6	Obtener eventos	El cliente debe recuperar las actividades para una entidad determinada.
RFC7	Configurar aplicación	El sistema debe permitir la edición de parámetros de configuración propios de la misma.
RFC8	Comentar evento	El sistema debe permitir la publicación de comentarios y respuestas a encuestas de los eventos que los implementen.
RFC9	Visualizar novedades	El sistema debe visualizar las novedades de las entidades que definan una fuente de alimentación para estas.
RFC10	Buscar	El sistema de implementar algún mecanismo para realizar búsquedas generales sobre las entidades, sus eventos y actividades.
RFC11	Indicaciones a una dirección	El sistema debe determinar el recorrido a seguir para llegar a una determinad entidad o lugar de un evento.

No funcionales:

ID	Nombre	Descripción
RNFC1	Tiempo de respuesta	El sistema debe optimizar el tiempo de respuesta de conexión para el intercambio de datos con el servidor.
RNFC2	Aspectos de interfaz gráfica	El sistema debe contar con una interfaz gráfica amigable, previsible y fácil de utilizar.
RNFC3	Tolerancia a fallos	El sistema debe poder recuperarse ante un error para evitar el cierre forzoso del mismo.
RNFC4	Abstracción de hardware	El sistema debería ser soportado por la mayor gama de dispositivos con Android como sea posible.

5.2.2 - Requerimientos del administrador Web

Funcionales:

ID	Nombre	Descripción

RFS1	Administrar Entidades	El sistema debe permitir crear, modificar, eliminar y consultar entidades.
RFS2	Administrar Eventos	El sistema debe permitir crear, modificar, eliminar y consultar eventos para una entidad.
RFS3	Administrar Actividades	El sistema debe permitir crear, modificar, eliminar y consultar actividades para un evento.
RFS4	Administrar Tipo de Entidades	El sistema debe permitir crear, modificar, eliminar y consultar tipos de entidades.
RFS5	Administrar Tipo de Eventos	El sistema debe permitir crear, modificar, eliminar y consultar tipos de eventos.
RFS6	Administrar tipo de Actividades	El sistema debe permitir crear, modificar, eliminar y consultar tipos de actividades.
RFS7	Administrar Encuestas	El sistema debe permitir crear, modificar, eliminar y consultar Encuestas.
RFS8	Sincronizar Eventos	El sistema debe permitir sincronizar los eventos de una entidad según parametrización.

No Funcionales:

ID	Nombre	Descripción
RNFS1	Basado en Web	Debe ser 100% basado en Web para permitir su utilización a través de navegadores de Internet.
RNFS2	Independiente Navegador	El sistema debe operar de manera independiente del navegador que se utilice.
RNFS3	Interfaz simple	Debe contar con una interfaz simple que no requiere alto grado de capacitación para su utilización
RNFS4	Simplicidad carga de datos	Debe permitir una carga simple de datos, por ejemplo fechas.
RNFS6	Disponibilidad	El sistema debe estar 100% disponible para poder responder a requerimientos del cliente móvil.
RNFS7	Capacidad de respuesta	El sistema debe responder de manera rápida a requerimientos tanto desde la administración Web como a requerimientos del cliente móvil.
RNFS8	Validación de Información	El sistema debe validar automáticamente el ingreso de datos. El ingreso de validación debe tener en cuenta obligatoriedad de campos, longitud de caracteres, tipos de datos, etc.

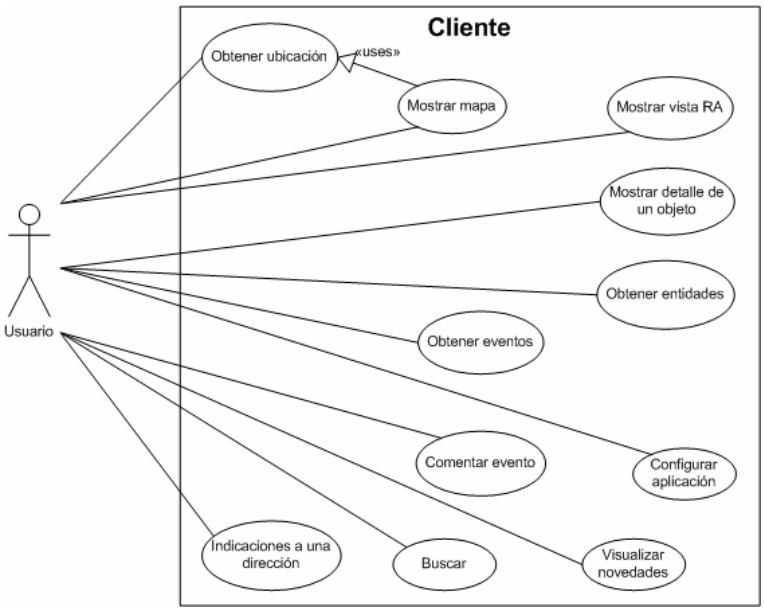
5.3 - CASOS DE USO

El propósito de los casos de uso es describir en lenguaje natural la funcionalidad completa de un sistema a desarrollar y su empleo se realiza en el proceso de especificación de requisitos del sistema. Permiten capturar y definir los requisitos que debe cumplir una aplicación.

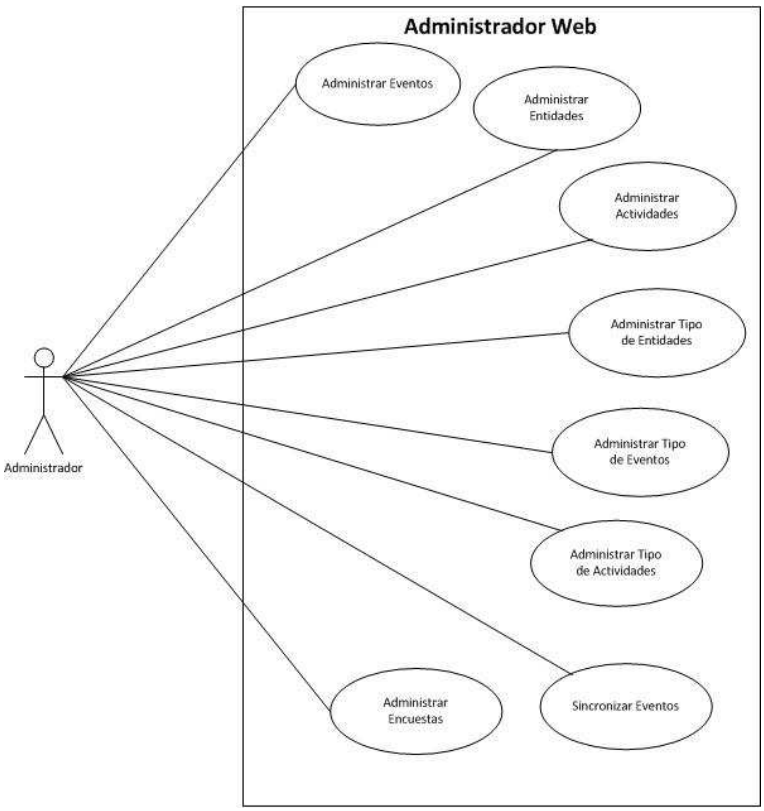
En este apartado utilizaremos los casos de uso como forma de acercar al lector las funcionalidades que la aplicación va a desempeñar frente al usuario. Sin embargo, no vamos a profundizar demasiado en las posibilidades que ofrece esta técnica, sino que vamos a limitarnos en aquellos aspectos que son más ilustrativos que le ayuden a comprender mejor qué es lo que realiza la aplicación.

Hemos identificado los siguientes casos de uso del sistema:

5.3.1 - Casos de Uso del cliente móvil



5.3.2 - Casos de Uso del Administrador Web



5.3.3 - Descripción de los casos de uso Cliente móvil

Nombre	Mostrar mapa
Req. que satisface	RFC1 - Mostrar mapa
Actor	Usuario
Descripción	Cuando se inicia la aplicación, la primera vista que se le presente al usuario debe ser la del mapa. Este mapa debe ser interactivo y navegable, permitiendo al usuario hacer zoom y moverse por el mismo. Inicialmente los únicos elementos que se muestran en el mapa son las entidades registradas en el sistema y los eventos para hoy. De estar disponible, se debe mostrar al usuario un indicador su ubicación actual.
Precondiciones	• Debe existir conexión a Internet

Nombre	Obtener ubicación
Req. que satisface	RFC2 - Obtener ubicación
Actor	Usuario
Descripción	El usuario puede indicar la utilización de un dispositivo receptor GPS para determinar su ubicación actual. Esta funcionalidad no es requerida para ejecutar la aplicación, pero sin su presencia el resto de las funcionalidades sin limitadas. Esta ubicación se utiliza para indicar al usuario su posición actual en el mapa, y también para construir los elementos virtuales en la vista RA.
Precondiciones	• Debe existir un mapa inicializado • Debe existir una conexión a Internet • Debe existir un dispositivo receptor GPS y esta habilitado

Nombre	Mostrar vista RA
Req. que satisface	RFC3 - Mostrar vista RA
Actor	Usuario
Descripción	Esta vista es capaz de mostrar las entidades registradas e información de estas sobre la captura de video de la cámara del dispositivo. Estos objetos virtuales deben estar alineados con la visión de la cámara en todo momento. La aplicación debe ser capaz de intercambiar entre vista de mapa y vista de realidad aumentada.
Precondiciones	• Debe existir una conexión a Internet • Debe existir un dispositivo receptor GPS y esta habilitado

Nombre	Mostrar detalle de un objeto
Req. que satisface	RFC4 - Mostrar detalle de un objeto
Actor	Usuario
Descripción	Se debe permitir al usuario obtener la información en detalle de cualquier objeto registrado en el sistema. Estos objetos son las entidades, los eventos y las actividades. De estos objetos se deben identificar sus atributos, y mostrarse de forma clara en pantalla.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Obtener entidades
Req. que satisface	RFC5 - Obtener entidades
Actor	Usuario
Descripción	Tanto la vista de realidad aumentara como la vista del mapa deben mostrar las entidades registradas en el sistema de forma que puedan identificarse claramente. Las entidades deben mostrarse en su ubicación original con su nombre y un icono que permita identificar rápidamente que tipo de entidad es.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Obtener eventos

Req. que satisface	RFC6 - Obtener eventos
Actor	Usuario
Descripción	El usuario debe ser capaz de visualizar los eventos para una determinada entidad. Estos eventos pueden ser mostrados en forma de lista y también consultados sobre el mapa. Los eventos se mostrarán en el mapa conforme a filtros que indique el usuario, pudiendo clasificarse por tipo de evento. También se permitirá al usuario la consulta particular de eventos a realizarse próximamente.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Configurar aplicación
Req. que satisface	RFC7 - Configurar aplicación
Actor	Usuario
Descripción	Existirá una manera sencilla de que el usuario configure los aspectos básicos de la aplicación. Estos aspectos serán de carácter visual, no involucrando prácticamente aspectos funcionales.
Precondiciones	

Nombre	Comentar evento
Req. que satisface	RFC8 - Comentar evento
Actor	Usuario
Descripción	Se permitirá al usuario brindar algún tipo de feedback para un evento. Este feedback puede ser implementado a través de un simple comentario sobre el evento, o respondiendo una encuesta parametrizable asociada al evento.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Visualizar novedades
Req. que satisface	RFC9 - Visualizar novedades
Actor	Usuario
Descripción	Podrán ser consultadas las novedades de las entidades que definieron algún tipo de lector de novedades. Estas se mostraran al usuario en pantalla en forma de listado o pagina Web.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Buscar
Req. que satisface	RFC10 - Buscar
Actor	Usuario
Descripción	Se debe brindar al usuario un mecanismo para buscar dentro de la base de datos del sistema. La búsqueda será por texto simple y los resultados contendrán la lista de entidades, eventos y/o actividades que cumplan con los criterios de búsqueda. Desde cada resultado se podrá mostrar el detalle del objeto encontrado y ubicarlo en el mapa.
Precondiciones	• Debe existir una conexión a Internet

Nombre	Indicaciones a una dirección
Req. que satisface	RFC11 - Indicaciones a una dirección
Actor	Usuario
Descripción	Conocida la ubicación actual de usuario, podrá solicitar las indicaciones de como llegar a cualquier evento o actividad definida en el sistema. Estas indicaciones se mostrarán en el mapa e identificará el recorrido a realizar para llegar al evento o actividad.
Precondiciones	• Debe existir un mapa inicializado • Debe existir una conexión a Internet

5.3.4 - Descripción de los casos de uso Administrador Web

Nombre	Administrar Entidades
Req. que satisface	RFS1 - Administrar Entidades
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear entidades. El sistema validara la información obligatoria cuando se haya seleccionado la operación crear o modificar. Para facilitar el ingreso de la ubicación geográfica de la entidad ya sea en la modificación y creación y así también como para mejorar la visualización de la ubicación se utilizará un mapa-geográfico. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Eventos
Req. que satisface	RFS2 - Administrar Eventos
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear Eventos. Al crear o modificar un Eventos el sistema validara la información obligatoria. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Actividades
Req. que satisface	RFS3 - Administrar Actividades
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear Actividades. Al agregar o modificar una Actividad el sistema validara la información obligatoria. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Tipo de Entidad
Req. que satisface	RFS4 - Administrar Tipo de Entidad
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear tipo de entidades. En la modificación y creación el sistema validara la información obligatoria. Se podrán eliminar los tipos de entidades que no sean utilizados por alguna entidad. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Tipo de Eventos
Req. que satisface	RFS5 - Administrar Tipo de Eventos
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear tipos de eventos. En la modificación y creación el sistema validara la información obligatoria. Se podrán eliminar los tipos de eventos que no sean utilizados por algún evento existente en el sistema. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Tipo de Actividades
Req. que satisface	RFS6 – Administrar Tipo de Actividades
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear tipo de actividades. En la modificación y creación el sistema validara la información obligatoria. Se podrán eliminar los tipos de actividades que no sean utilizados por alguna actividad existente en el sistema. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Administrar Encuestas
Req. que satisface	RFS7 - Administrar Encuestas
Actor	Administrador
Descripción	El sistema permitirá consultar, modificar, eliminar y crear Encuestas. Las encuestas son utilizadas por los eventos para obtener una valoración sobre el mismo. La valoración se hará a través de la aplicación cliente-móvil. En la modificación y creación el sistema validara la información obligatoria. Se podrán eliminar las encuestas que no sean utilizados por algún evento existente en el sistema. Al finalizar cada operación (ABM) se informara un mensaje indicando el resultado de la operación.
Precondiciones	El administrador debe estar autenticado en el sistema.

Nombre	Sincronizar Eventos
Req. que satisface	RFS8 – Sincronizar Eventos
Actor	Administrador
Descripción	Se podrá sincronizar los eventos de una entidad. A partir de la consulta de una entidad se puede acceder a la opción de sincronizar eventos. Esta opción creara en el sistema todos los eventos que se encuentren según la parametrización del lector de actividades de la entidad seleccionada. Los eventos creados tendrán por defecto un tipo de evento y encuesta provisto por el sistema. Luego de la sincronización se mostrara la consulta de la entidad para poder ver en detalle los datos de entidad.
Precondiciones	El administrador debe estar autenticado en el sistema.

5.4 - ARQUITECTURA DEL SISTEMA

Como dijimos anteriormente, el prototipo tendrá una arquitectura distribuida. Pero además de los componentes cliente y servidor, necesitaremos de otros componentes de software para llevar a cabo la tarea de servicio y visualización de datos. Entonces, podemos extender la definición de la arquitectura del sistema en la siguiente figura:

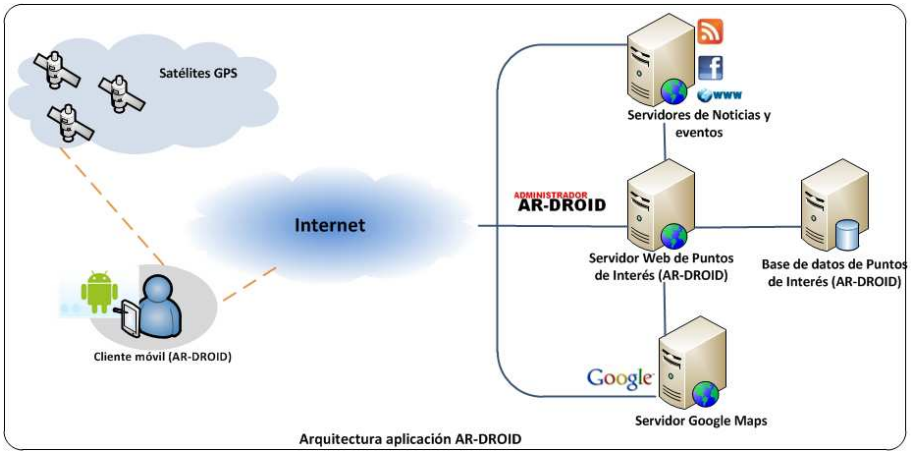


Figura 19. Arquitectura extendida de AR-DROID

Se mostrará sobre *Google Maps* los puntos de interés, así como la ubicación actual del usuario móvil y la ruta entre dos sitios tomando como punto de partida la ubicación actual y como destino el lugar donde desea llegar.

La aplicación utilizará el componente *GPS* presente en el mismo dispositivo móvil para conocer la propia ubicación, pudiendo de este modo mostrársela al usuario en el mapa

Se deberá disponer de una *Conexión a Internet* para poder intercambiar información con el servidor de la aplicación (feedback), obtener los puntos de interés, las rutas a partir de los servicios de Google Maps y conectarse a los servidores de noticias (RSS, Facebook, portal Web). Este intercambio se realiza utilizando la tecnología JSON(*) y sobre el protocolo HTTP, el funcionamiento más detallado de esta comunicación se detallara más adelante.

El *Cliente* funcionará en dispositivos móviles con plataforma Android (a partir de la versión 2.0). Es imprescindible la presencia de GPS y conexión a Internet para el funcionamiento de la aplicación.

El cliente móvil realizará tareas como la determinación de la ubicación del usuario, presentación de resultados obtenidos a través de lo comunicación a distintos servicios como, el servidor de noticias, servidor de Google Maps para obtener la trayectoria al punto deseado y el servidor Web de puntos de interés de la aplicación.

El dispositivo móvil visualizará la información sobre un mapa de acuerdo a la ubicación del usuario. El usuario podrá consultar individualmente la información completa de las entidades y eventos con la posibilidad de interactuar con la aplicación a modo de red social brindando su opinión sobre el evento/actividad. La aplicación consumirá esta información en tiempo real haciendo uso de los distintos sensores con los que cuentan los dispositivos (como el GPS, el acelerómetro y el compás).

El prototipo incluirá una funcionalidad de realidad aumentada que a través de la cámara del dispositivo móvil será capaz de identificar y señalar en pantalla las unidades académicas que se encuentren registradas en el catálogo de entidades del proveedor de servicios. Este último se administrara desde una aplicación Web donde se podrán gestionar las entidades físicas y la información de los distintos servicios brindados por cada una.

El *Servidor* permanece siempre a la espera de conexiones por parte de los diferentes usuarios. Deberá disponer, en primer lugar, de una red de comunicación entre sus sistemas y los usuarios. En segundo lugar, necesita una arquitectura de servidores que se encarguen de recibir peticiones, procesarlas y brindar uno o un conjunto de resultados.

El servidor cumplirá las siguientes funcionalidades: recibir peticiones por parte del cliente móvil para brindar información sobre los distintos puntos de interés y actualizar información enviada por el cliente.

El servidor contará con:

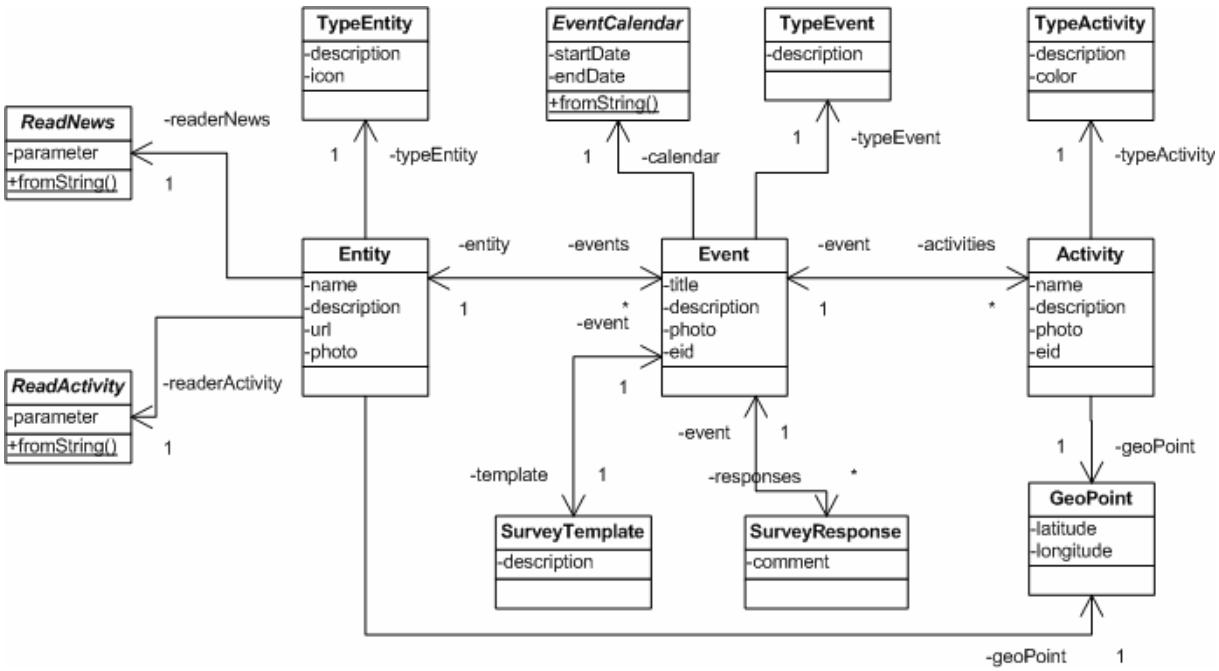
- Un servidor de bases de datos MySQL donde se almacenara toda la información necesaria para el funcionamiento del sistema
- Aplicación Web que permite administrar el catálogo de la de entidades, eventos, actividades, etc. y se comunicara con servidores de eventos para sincronizarlos e incorporarlos al sistema.

5.5 - DIAGRAMAS DE CLASES

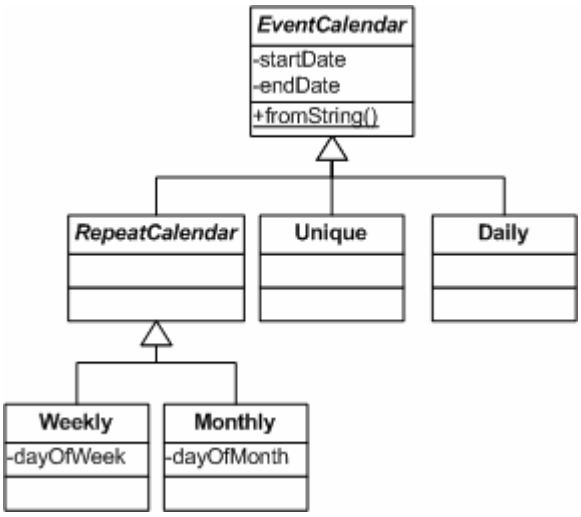
Las clases utilizadas para resolver el dominio del problema, sus atributos y operaciones, la visibilidad que de estos atributos tienen las demás clases, así como las relaciones que existen entre ellas y como colaboran, constituyen el modelo de clases. Mediante este tipo de modelos se expresa de manera estática, con mayor o menor nivel de detalle, la futura implementación del sistema, así como permite dar una idea bastante cercana a la forma en la que se ha abordado el problema.

Mostramos más abajo el diagrama de clases del prototipo. Este es el modelo de datos en común, al estar estas clases definidas tanto en el servidor como en el cliente. Decimos que están definidas en ambos extremos del prototipo porque van a diferir en implementación: en algunos casos las clases del servidor servirán de parametrización para las clases del cliente que implementaran las acciones en base a esa parametrización.

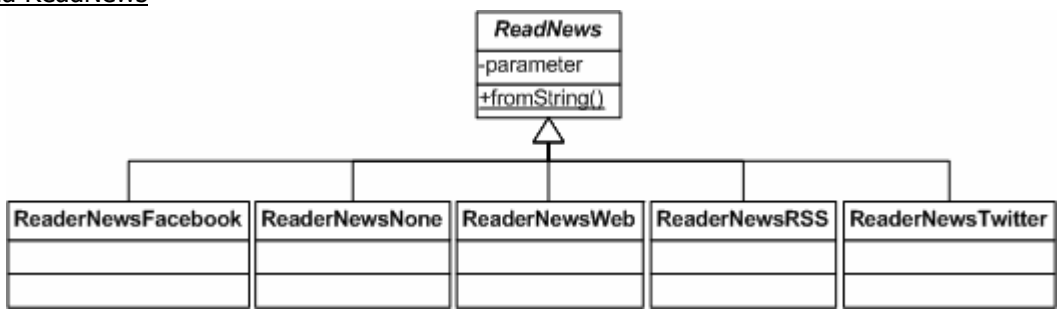
Daremos este diagrama con el objetivo principal de ayudar a comprender cómo buscamos una solución al dominio y pretende servir de anticipo a la explicación más detallada de la implementación en los capítulos posteriores.



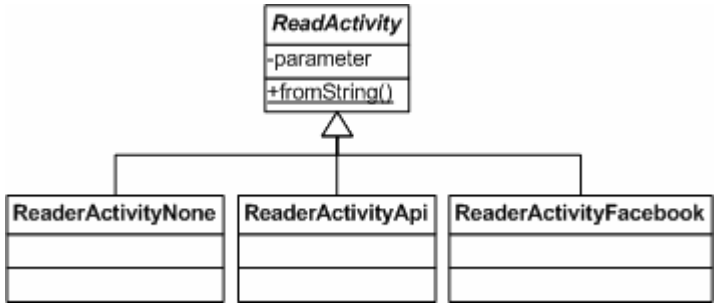
Jerarquía EventCalendar



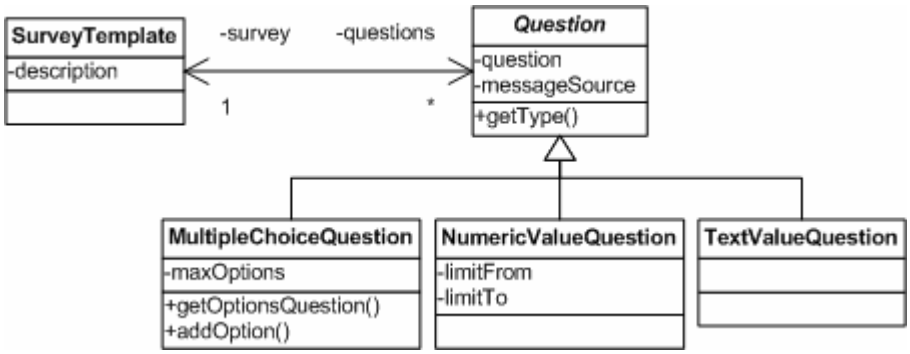
Jerarquía ReadNews



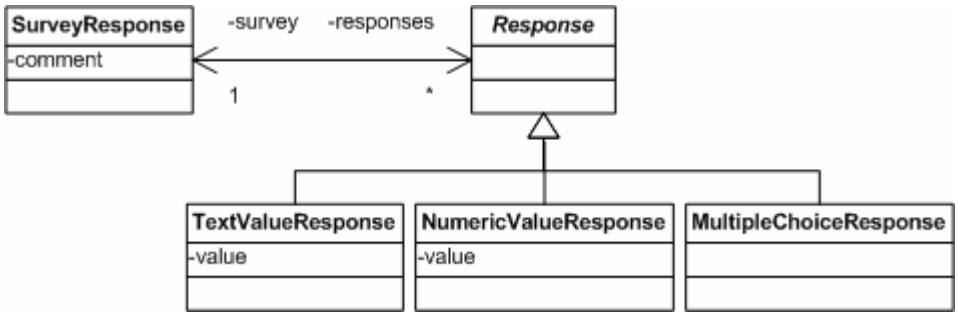
Jerarquía ReadActivity



Composición SurveyTemplate



Composición SurveyResponse



5.5.1 - Descripción de las clases

Entity: La clase *Entity* es una de las más significativas del modelo. Representa lo que es una entidad en el sistema. Esta compuesta por un *GeoPoint* (punto localizado), una lista de eventos implementados con *Event*, y un par de lectores para alimentarse de eventos y mostrar novedades a los clientes móviles (alguna implementación de *ReadNews* y *ReadActivity*).

Event: Es la clase creada para representar los distintos tipos de acontecimientos que suceden en las entidades. Esta se compone de una lista de actividades (*Activity*) un calendario de periodicidad (*EventCalendar*) y un tipo de entidad (*TypeEntity*). También incluye composiciones para implementar el

feedback de la aplicación, *SurveyTemplate* y una lista de *SurveyResponse*.

Activity: *Activity* es la clase que implementa cada actividad que componen un evento; independientemente si el evento tiene un o varias actividades, un *Event* cuenta con una lista de *Activiy*. La clase esta compuesta de un tipo de actividad (*TypeActivity*) y su ubicación (*GeoPoint*).

GeoPoint: Esta clase representa una posición localizable, compuesta por latitud, longitud y altitud. La utilizaremos para elementos *Entity* y *Event*.

ReadNews: Es una jerarquía para definir las distintas implementaciones de los lectores de novedades para las *Entity*. Esta jerarquía tiene la particularidad de implementar la parametrización del lado del servidor, y la funcionalidad del lado del cliente.
Las distintas implementaciones son: *Web* (las novedades se obtienen de una pagina Web), *RSS* (se obtienen desde una fuente RSS o Atom), *Facebook* (son las publicaciones desde un grupo abierto de Facebook), *Twitter* (las novedades son los Twitts de un usuario Twitter). También hay una implementación *None*, para no dotar a la entidad de novedades.

ReadActivity: Es una jerarquía para definir las distintas implementaciones de los lectores de eventos para las *Entity*. Estos lectores alimentan automáticamente a las *Entity* de elementos *Event* . Esta jerarquía, como sucede con *ReadNews*, también implementa la parametrización del lado del servidor, y la funcionalidad del lado del cliente.
Las distintas implementaciones son: *API* (los eventos se obtienen de una URL donde se devuelve un XML definido - ver "Definición API"), *Facebook* (son las eventos publicados en un grupo abierto de Facebook) y *None* (No se alimenta a la entidad automáticamente).

EventCalendar: *EventCalendar* es una jerarquía para definir las distintas implementaciones de los lectores de eventos para las *Entity*. Estos lectores alimentan automáticamente a las *Entity* de elementos *Event* . Esta jerarquía, como sucede con *ReadNews*, también implementa la parametrización del lado del servidor, y la funcionalidad del lado del cliente.
Las implementaciones son: Unique (el evento no se repite), Daily (se repite diariamente), WorkingDaily (se repite de lunes a viernes), Weekly (una vez por semana) y Monthly (una vez por mes).

SurveyTemplate: Esta clase implementa la funcionalidad de feedback que darán los usuarios a los eventos. Es la parametrización a las preguntas que proveerán la estructura de la opinión de los usuarios. Está compuesta por una lista de *Question*.

Question: Es la jerarquía que implementa las distintas formas de "preguntas" que conformaran una *SurveyTemplate*.
Las implementaciones posibles son: *MultipleChoiceQuestion* (define la selección de una o varias opciones), *NumericValueQuestion* (pregunta con una respuesta numérica) y *TextValueQuestion* (implementa una pregunta que espera una respuesta textual).

SurveyResponse: Esta clase implementa también la funcionalidad de feedback que darán los usuarios a los eventos, definiendo la parte de respuesta del usuario. Constituye de las respuestas a las preguntas que define una *SurveyTemplate*. Está compuesta por una lista de *Response*.

Response: Es la jerarquía que implementa las distintas respuestas que manifiesta el usuario a través del cliente a una *SurveyTemplate*. Estas conforman la lista de respuestas de *SurveyResponse*.
Las implementaciones posibles son: *MultipleChoiceResponse* (la selección de una o varias opciones), *NumericValueResponse* (respuesta numérica) y *TextValueResponse* (respuesta textual).

6 - IMPLEMENTACIÓN DEL ADMINISTRADOR WEB

El administrador Web, como dijimos, constituirá la parte servidor de la aplicación. Este será un proyecto Java implementado sobre el Framework Groovy on Grails.

Esta sección describirá el proceso de construcción del administrador. Abordaremos detalles técnicos de implementación y cuestiones de diseño.

Hemos creado un documento explicativo de cómo montar el ambiente de desarrollo e instalar las herramientas necesarias para poder levantar el administrador Web. Ver apéndice *A - PREPARACIÓN DE AMBIENTE*.

6.1 - EL FRAMEWORK GROOVY ON GRAILS

Grails es un marco de trabajo para el desarrollo de aplicaciones Web que se basa en las ideas de *Codificación a través de Convenciones* (coding by convention) y *DRY* (don't repeat yourself) utilizando el lenguaje de programación Groovy.

Grails se ejecuta sobre una máquina virtual de Java, teniendo acceso completo a la plataforma y librerías Java.

Con Grails pueden crearse fácilmente aplicaciones Web gracias a:

- Ofrece un Entorno completo para el desarrollo y publicación de aplicaciones: Todas las dependencias y configuración requeridas para ejecutar las aplicaciones Web son proporcionadas por Grails. Usted debe solamente preocuparse por el código particular de su aplicación.
- Incluye un servidor Jetty embebido.
- El modo de desarrollo automáticamente recarga los cambios realizados al código de la aplicación sin tener que reiniciar el servidor Web.
- Ofrece mapeo de persistencia automático para las clases del Dominio del problema, así como también manejo automático de las relaciones entre las entidades.
- Ofrece Scaffolding(*) para el acceso a datos, ofreciendo posibilidades para el desarrollo rápido de operaciones CRUD (Create, Read, Update, Delete).
- Soporte de tecnologías para la implementación de vistas de datos utilizando "dynamic tag libraries" y "Groovy Server Pages (GSP)".
- Extensibilidad a través de la utilización de plugins.

Grails puede usarse como un entorno de desarrollo autónomo que esconde todos los detalles de configuración así como también permite integrar lógica de negocio escrita en Java. Entre los objetivos de Grails está el hacer el desarrollo Web lo más simple y atractivo posible de tal forma que un amplio segmento de desarrolladores puedan utilizarlo.

Grails esta construido sobre cinco fuertes pilares:

- *Groovy* para la creación de propiedades y métodos dinámicos en los objetos de la aplicación
- *Spring* para los flujos de trabajo e inyección de dependencias
- *Hibernate* para la persistencia
- *SiteMesh* para la composición de las vistas
- *Ant* para la gestión de procesos de desarrollo

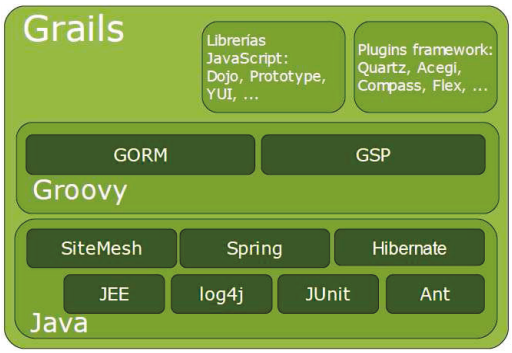


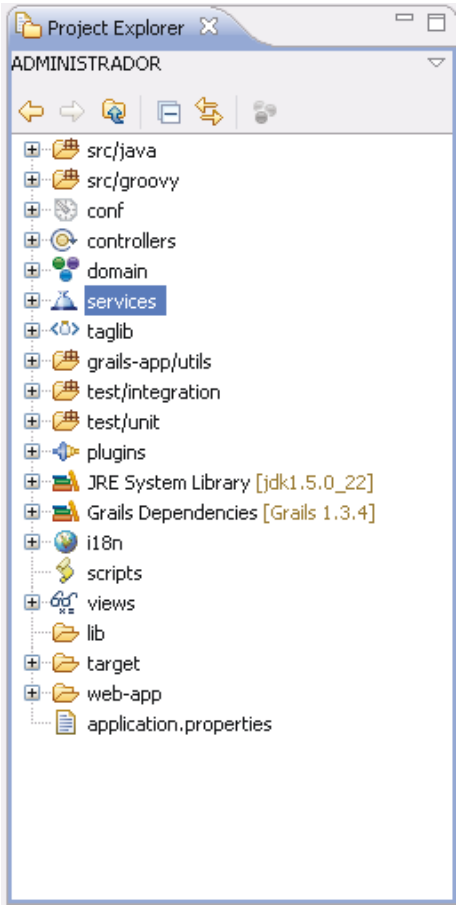
Figura 20. Estructura de diseño de Gralis

Groovy es el lenguaje dinámico, de código libre, para la Máquina Virtual de Java. Ofrece una sintaxis flexible en Java, tal que la mayoría de los desarrolladores de Java pueden aprenderlo fácilmente. Groovy proporciona características que están presentes en otros lenguajes dinámicos como Ruby, Python o Smalltalk. Groovy realmente sobresale en su capacidad para definir fácilmente lenguajes específicos de dominio (DSL), que puede ser utilizado como una capa de abstracción que permite a los no técnicos codificar, por ejemplo, reglas de negocio.

La combinación de Groovy y Grails ofrece beneficios de productividad que desafía a Ruby on Rails, pero para la plataforma Java probada, escalable e integrable.

6.2 - CREANDO EL PROYECTO

Desde Eclipse, nuestro entorno de desarrollo, crearemos un nuevo proyecto Grails:



Proyecto Grails

A continuación daremos una breve descripción de cada componente:

- src/java: Contenedor para fuentes .java
- src/groovy: Contenedor para fuentes dinámicos .groovy
- conf: Almacena archivos de configuración
- controllers: Meta-contenedor para organizar los controladores de la aplicación
- domain: Entidades. Meta -contenedor para organizar los objetos de negocio de la de la aplicación
- services: Meta -contenedor para organizar los servicios que implementan la lógica de negocio de la aplicación
- taglib: Contenedor de librerías de etiquetas para GSP, más simplificados que JSP
- grails-app/utils: Contenedor para fuentes .java y .groovy
- test/integration y test/unit: Contenedor de fuentes para implementar test JUnit
- plugins: Contiene los plugins utilizados por el proyecto, que permiten ampliar la funcionalidad de nuestra aplicación
- i18n: Almacén para archivos de recursos de internacionalización de la aplicación
- scripts: Contenedor de script en Groovy. Permiten ejecutar código dinámico sobre la consulta de Grails
- views: Contenedor de vistas de la aplicación, Groovy utiliza GSP, una versión simplificada de JSP.

6.3 - CREANDO LAS ENTIDADES

Desde el proyecto, creamos una nueva *Domain class*, una clase Groovy para implementar objetos de negocio. Empezaremos por implementar la clase para definir un tipo de entidad:

TypeEntity.goovy

```
 1 package ar.droid.admin
 2
 3 class TypeEntity {
 4
 5       String description
 6       byte[] icon
 7
 8       static constraints = {
 9               description(blank: false)
10               icon(nullable: false)
11       }
12
13       static mapping = { icon sqlType: 'blob' }
14
15       @Override
16       public String toString() {
17               return description
18       }
19 }
```

Nombremos algunos detalles que observamos en el código:

- La estructura `constraints` define validaciones sobre la entidad. Nos esta diciendo que `description` no puede ser blanco, e `icon` no puede ser nulo.
- La estructura `mapping` define directivas de mapeo. Esto esta obligando a Hibernate definir el campo `icon` de tipo `'blob'`.

`TypeEntity` es ahora una clase Groovy que implementa persistencia, permite operaciones de salvado y recuperación de datos, y esta lista para definir Scaffolding.

6.4 - CREANDO EL CONTROLADOR Y LAS VISTAS

Groovy cuenta con una librería de scripts útiles, entre los que podemos encontrar:

- generate-controller.groovy: Dada una *domain class* genera el controlador para operaciones CRUD sobre el modelo.
- generate-views.groovy: Dada una *domain class* genera la vista para operaciones el modelo, generando cuatro archivos GSP; list, create, edit y show.
- generate-all.groovy: Dada una *domain class* genera tanto el controlador como las vistas.

Utilizando el script generate-all, crearemos el controlador y las vistas para el modelo `TypeEntity`, los que nos crea los siguientes archivos:

TypeEntityController.goovy

```
 1 package ar.droid.admin
 2
 3 class TypeEntityController {
 4
 5     static allowedMethods = [save: "POST", update: "POST", delete: "POST"]
 6
 7     def index = {
 8         redirect(action: "list", params: params)
 9     }
10
11     def list = {
12         params.max = Math.min(params.max ? params.int('max') : 10, 100)
13         [typeEntityInstanceList: TypeEntity.list(params), typeEntityInstanceTotal:
TypeEntity.count()]
14     }
15
16     def create = {
17         def typeEntityInstance = new TypeEntity()
18         typeEntityInstance.properties = params
19         return [typeEntityInstance: typeEntityInstance]
20     }
21
22     def save = {
23         def typeEntityInstance = new TypeEntity(params)
24         if (typeEntityInstance.save(flush: true)) {
25             flash.message = "${message(code: 'default.created.message', args:
[message(code: 'typeEntity.label', default: 'TypeEntity'), typeEntityInstance.id])}"
26             redirect(action: "show", id: typeEntityInstance.id)
27         }
28         else {
29             render(view: "create", model: [typeEntityInstance:
typeEntityInstance])
30         }
31     }
32
33     def show = {
34         def typeEntityInstance = TypeEntity.get(params.id)
35         if (!typeEntityInstance) {
36             flash.message = "${message(code: 'default.not.found.message', args:
[message(code: 'typeEntity.label', default: 'TypeEntity'), params.id])}"
37             redirect(action: "list")
38         }
39         else {
40             [typeEntityInstance: typeEntityInstance]
41         }
42     }
43
44     def edit = {
45         ...
53     }
54
55     def update = {
56         ...
80     }
81
```

```
 82     def delete = {
 83         ...
 99     }
100 }
```

En el caso del controlador, nos generó una clase que responde a las acciones:

- index: atiente la petición inicial del controlador. Por defecto redirecciona a la lista de registros
- list: lista los registros del modelo
- create: muestra la pantalla de creación de un nuevo modelo
- save: persiste un nuevo modelo en la base de datos
- show: visualiza un modelo
- edit: muestra la pantalla de edición de un modelo
- update: actualiza un modelo en la base de datos
- delete: elimina un modelo de la base de datos

El controlador nos generó un archivo de código con las operaciones básicas para CRUD. Aunque tendremos que retocar algo en las acciones para mejorar estas operaciones. Los cambios resultan en el siguiente impacto en el código del controlador:

EntityController.goovy

```
  1 package ar.droid.admin
  2
  3 class TypeEntityController {
  4
  5     ...
 54
 55     def update = {
 56         def typeEntityInstance = TypeEntity.get(params.id)
 57         if (typeEntityInstance) {
 58             if (params.version) {
 59                 def version = params.version.toLong()
 60                 if (typeEntityInstance.version > version) {
 61
 62                     typeEntityInstance.errors.rejectValue("version",
"default.optimistic.locking.failure", [message(code: 'typeEntity.label', default:
'Tipo de entidad')] as Object[], "Otro usuario ha actualizado el objeto")
 63                     render(view: "edit", model: [typeEntityInstance:
typeEntityInstance])
 64                     return
 65                 }
 66             }
 67
 68                      def icon = typeEntityInstance.icon;
 69                      typeEntityInstance.properties = params;
 70                      if (params.get("icon").size == 0){
 71                              typeEntityInstance.icon = icon;
 72                      }
 73             if (!typeEntityInstance.hasErrors() && typeEntityInstance.save(flush:
true)) {
 74                 flash.message = "${message(code: 'default.updated.message', args:
[message(code: 'typeEntity.label', default: 'Tipo de entidad'),
typeEntityInstance.id])}"
 75                 redirect(action: "show", id: typeEntityInstance.id)
 76             }
 77             else {
 78                 render(view: "edit", model: [typeEntityInstance:
typeEntityInstance])
 79             }
 80         }
 81         else {
 82             flash.message = "${message(code: 'default.not.found.message', args:
[message(code: 'typeEntity.label', default: 'Tipo de entidad'), params.id])}"
 83             redirect(action: "list")
 84         }
 85     }
 86
```

```
 87      ...
105
106      def showIcon = {
107              def typeEntityInstance = TypeEntity.get(params.id)
108              response.contentType = "image/jpeg"
109              response.contentLength = typeEntityInstance?.icon.length
110              response.outputStream.write(typeEntityInstance?.icon)
111      }
112 }
```

A la definición del método update le agregamos una condición para no sobrescribir el contenido de la propiedad `icon` cuando viene vacía desde el requerimiento HTTP, sino se perdía la imagen al actualizar un tipo de entidad. Vemos en el código otras condiciones, que no son más que chequeos de existencia y control de concurrencia que genera automáticamente Groovy.

También agregamos una acción para devolver la imagen representativa del tipo de entidad. Esta se obtiene desde una etiqueta <img> por ejemplo, desde la URL *http://server/ADMIN/typeEntitty/showIcon/{typeEntity_ID}*.

En cuanto a las vistas, y como dijimos anteriormente, Groovy nos genero cuatro archivos bajo la carpeta views/typeEntity. Veremos a continuación fragmentos de los archivos list.gsp y create.gsp

views/typeEntity/list.gsp

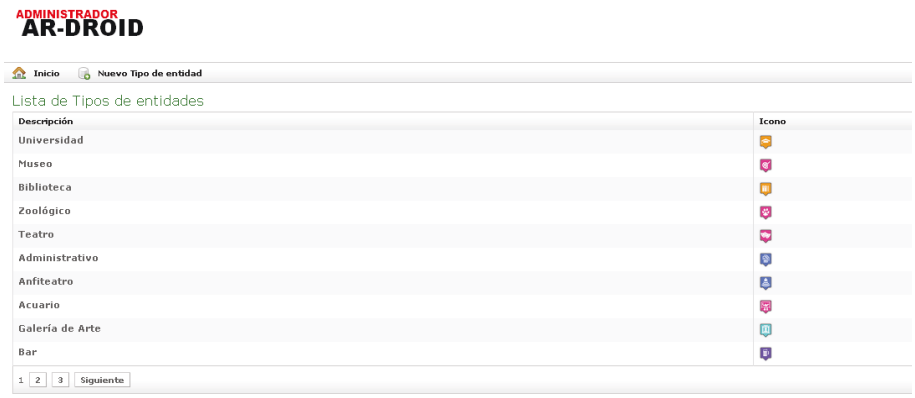
```
 1 <%@ page import="ar.droid.admin.TypeEntity" %>
 2 <html>
 3     <head>
 4         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 5         <meta name="layout" content="main" />
 6         <g:set var="entityName" value="${message(code: 'typeEntity.label', default:
'TypeEntity')}" />
 7         <title><g:message code="default.list.label" args="[entityName]" /></title>
 8     </head>
 9     <body>
10         <div class="nav">
11             <span class="menuButton"><a class="home" href="${createLink(uri:
'/')}"><g:message code="default.home.label"/></a></span>
12             <span class="menuButton"><g:link class="create"
action="create"><g:message code="default.new.label" args="[entityName]"
/></g:link></span>
13         </div>
14         <div class="body">
15             <h1><g:message code="default.list.label" args="[entityName]" /></h1>
16             <g:if test="${flash.message}">
17             <div class="message">${flash.message}</div>
18             </g:if>
19             <div class="list">
20                 <table>
21                     <thead>
22                         <tr>
23
24                             <g:sortableColumn property="id" title="${message(code:
'typeEntity.id.label', default: 'Id')}" />
25
26                             <g:sortableColumn property="description"
title="${message(code: 'typeEntity.description.label', default: 'Description')}" />
27
28                             <g:sortableColumn property="icon"
title="${message(code: 'typeEntity.icon.label', default: 'Icon')}" />
29
30                         </tr>
31                     </thead>
32                     <tbody>
33                     <g:each in="${typeEntityInstanceList}" status="i"
var="typeEntityInstance">
34                         <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
35
36                             <td><g:link action="show"
id="${typeEntityInstance.id}">${fieldValue(bean: typeEntityInstance, field:
"id")}</g:link></td>
37
```

```
38                              <td>${fieldValue(bean: typeEntityInstance, field:
"description")}</td>
39
40                              <td>${fieldValue(bean: typeEntityInstance, field:
"icon")}</td>
41
42                          </tr>
43                      </g:each>
44                      </tbody>
45                  </table>
46              </div>
47              <div class="paginateButtons">
48                  <g:paginate total="${typeEntityInstanceTotal}" />
49              </div>
50          </div>
51      </body>
52  </html>
```

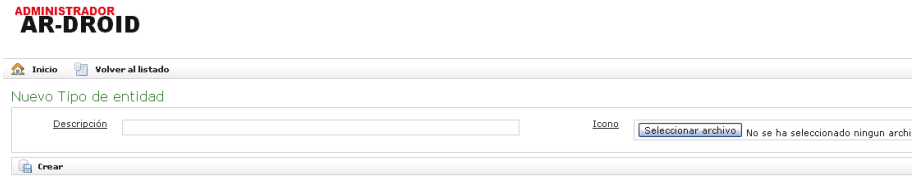
Veamos un poco en detalle este código:

- La línea 1 nos dice que vamos a trabajar con el modelo **`ar.droid.admin.TypeEntity`**
- La línea 6 esta utilizando un **`g:set`** para setear una variable en el contexto actual, en este caso establece la variable **`entityName`** con el valor desde un archivo de recursos, o por defecto **`'TypeEntity'`**
- La línea 12 crea un link a través de la etiqueta **`g:link`** para la acción **`create`**. Al no especificar un controlador, se considera el controlador actual.
- La línea 33, con la etiqueta **`g:each`** inicia una iteración sobre todos los elementos de la colección **`typeEntityInstanceList`**.
- La línea 36 imprime el valor de la propiedad "id" de cada objeto a través de la utilización de la función `fieldValue`.

Como resultado de esta definición del objeto de modelo, la posterior creación automática de controlador y vistas, y el retoque de los archivos de vistas, obtenemos el siguiente resultado en la aplicación Web:



Listado de tipos de entidades



Pantalla de creación de un nuevo tipo de entidad

Este es el procedimiento para crear entidades y generar automáticamente el Scaffolding de estas. Todas las clases del modelo, definidas anteriormente en el diagrama de clases serán implementadas en Groovy como *domain class*. Transcribimos ahora una de las entidades más importantes del modelo, la *domain class* Entity:

Entity.groovy

```groovy
package ar.droid.admin

import ar.droid.admin.reader.*

class Entity {

	static hasMany = [events: Event]
	static embedded = ['geoPoint']

	String name
	String description
	String url
	byte[] photo
	TypeEntity typeEntity
	GeoPoint geoPoint

	ReaderNews readerNews
	ReaderActivity readerActivity

	static constraints = {
		url(url:true)
		name(blank: false)
		readerActivity(nullable: false)
		readerNews(nullable: false)
	}
	static mapping = {
		description type: 'text'
		photo sqlType: 'blob'
	}

	@Override
	public String toString() {
		return name
	}
}

class GeoPoint {
	Double latitude;
	Double longitude;

	@Override
	public String toString() {
		return this.latitude + '@' + this.longitude;
	}
}
```

En el caso de la entidad `Entity`, implementaremos algunos cambios en el controlador y como interactúa con el modelo. Ya que la labor de alta o edición de este tipo de objeto requiere de varias tareas complementarias, debemos separar la lógica de negocio con la capa de control de la aplicación.

Como sabemos, Grails es un entorno para desarrollar siguiendo el patrón MVC(*), pero lo que hay que tener claro es, es que en éste patrón no son tres, sino cuatro, las capas en las que deberíamos separar nuestros componentes: a las tradicionales Modelo, Vista y Controlador, debemos sumarle la capa de servicio, que debiera implementar la lógica de negocio.

Para implementar y utilizar servicios en Groovy debemos especificar una propiedad con el nombre del mismo en el controlador, siguiendo la convención *entityName*Service:

EntityController.groovy

```groovy
package ar.droid.admin

import java.util.ArrayList
import grails.converters.deep.*
import ar.droid.admin.reader.*

class EntityController {
    def entityService

    static allowedMethods = [save: "POST", update: "POST", delete: "POST"]
```

```
 11
 12      ...
 20
 21      def create = {
 22          def entityInstance = new Entity()
 23              entityInstance.geoPoint = new GeoPoint()
 24          entityInstance.properties = params
 25
 26          return [entityInstance: entityInstance]
 27      }
 28
 29      def save = {
 30              def entityInstance = entityService.saveEntity(params)
 31              if(entityInstance.hasErrors()){
 32                      request.readerActivity_select =
entityInstance.readerActivity.class
 33                      request.readerNews_select = entityInstance.readerNews.class
 34                      render(view: "create", model: [entityInstance:
entityInstance])
 35              }
 36              else{
 37                      flash.message = "${message(code: 'default.created.message',
args: [message(code: 'entity.label', default: 'Entidad'), entityInstance.id])}"
 38                      redirect(action: "show", id: entityInstance.id)
 39              }
 40      }
 41
 42      ...
104
105      def newevent = {
106          def entityInstance = Entity.get(params.id)
107              redirect(controller: "event", action: "create", entity: params.id)
108      }
109
110      ...
123 }
```

Vemos que en la línea 8 definimos el servicio a usar, `def entityService`. Esta convención en el nombre es importante pues el servido se instanciará automáticamente a través de inyección de dependencias. En la línea 30 vemos como utilizamos el servicio para servir el alta de una nueva entidad a través de `entityService.saveEntity(params)`.

A continuación transcribimos la implementación del servicio:

EntityService.groovy

```
 1 package ar.droid.admin;
 2
 3 import ar.droid.admin.reader.ReaderNews
 4 import ar.droid.admin.reader.ReaderActivity
 5
 6 class EntityService {
 7       static transactional = true
 8
 9       def Entity saveEntity(params) {
10               def entityInstance = new Entity()
11               entityInstance.properties = params
12               entityInstance.readerActivity =
ReaderActivity.fromString(params.readerActivity_select)
13               entityInstance.readerNews =
ReaderNews.fromString(params.readerNews_select)
14
15               // crear composiciones
16               entityInstance.readerActivity.properties = params.readerActivity
17               entityInstance.readerNews.properties = params.readerNews
18
19               entityInstance.geoPoint = new GeoPoint()
20
21               // validar posición
22               if(params.latitude == null || ''.equals(params.latitude) ||
params.longitude == null || ''.equals(params.longitude)){
```

```
23                          entityInstance.errors.rejectValue('geoPoint', 'Debe
seleccionar la ubicación')
24                  }
25                  else {
26                          // armar geopoint
27                          entityInstance.geoPoint.latitude = new
BigDecimal(params.latitude)
28                          entityInstance.geoPoint.longitude = new
BigDecimal(params.longitude)
29
30                          entityInstance.save(flush: true)
31                  }
32                  return entityInstance;
33          }
34
35          def Entity updateEntity(params) {
36                  ...
62          }
63
64          ...
67  }
```

Mostramos a modo de pantallazo general el diagrama de base de datos generado automáticamente por Hibernate a través de Grails.

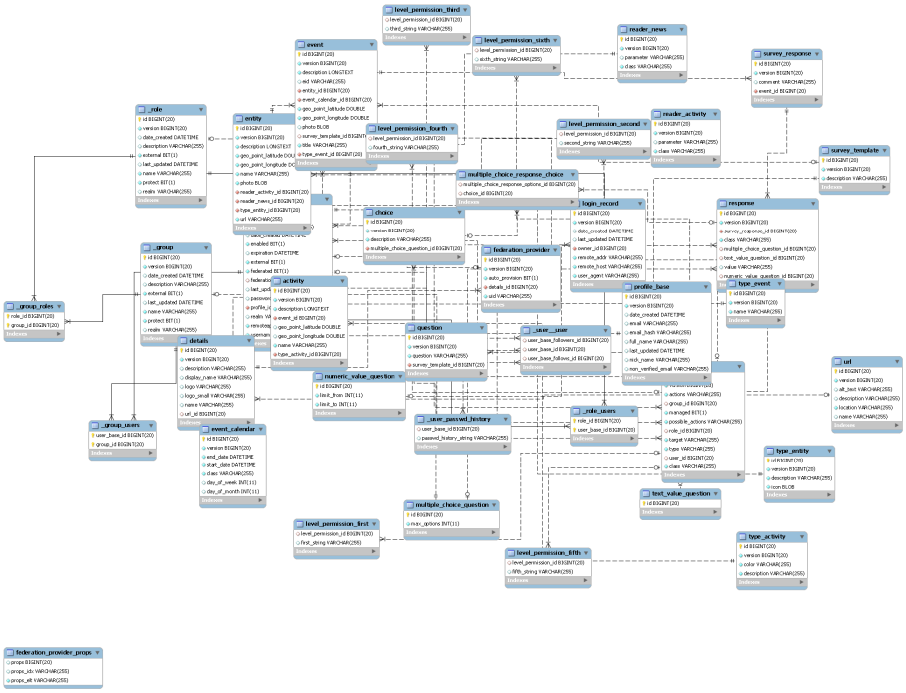


Diagrama ER resultante de del mapeo de Hibernate

6.5 - CONFIGURACIÓN DE LA APLICACIÓN

Grails define una metodología conocida como "convención antes que la configuración", esto es: con la configuración por defecto Grails se puede desarrollar una la aplicación sin hacer configuración alguna. Grails con un contenedor embebido y HSQLDB(*) en memoria lo que significa que ni siquiera se debe configurar una base de datos. Sin embargo, en aplicaciones empresariales evidentemente queremos manipular estos parámetros.

Como todo Framework de desarrollo, Groovy on Grails define una serie de parámetros de configuración para su funcionamiento. Si bien la creación de un proyecto desde el entorno de desarrollo ya permite tener un contexto funcional, las aplicaciones generalmente necesitan configuraciones tales como acceso a datos, redirecciones, cuentas y privilegios, etc.

Groovy cuenta con varios archivos de configuración y afines ubicados bajo el meta paquete */conf*, a saber:

BootStrap

Esta clase define dos métodos que se ejecutan al cargar la aplicación Web y al terminarla. Es útil para crear datos de prueba en ambientes de *testing* o evolutivos. También es utilizada en ambientes productivos para hacer chequeos de tipos.

BootStrap.groovy

```
1 class BootStrap {
2
3     def init = { servletContext ->
4     }
5     def destroy = {
6     }
7 }
```

BuildConfig

Desde Grails 1.1, este archivo es utilizado para controlar distintos aspectos del proceso de compilación como las rutas de trabajo y la resolución de dependencias.

BuildConfig.groovy

```
 1 grails.project.class.dir = "target/classes"
 2 grails.project.test.class.dir = "target/test-classes"
 3 grails.project.test.reports.dir = "target/test-reports"
 4 grails.project.dependency.resolution = {
 5       inherits("global") {
 6       }
 7       log "warn" // log level of Ivy resolver, either 'error', 'warn', 'info',
'debug' or 'verbose'
 8       repositories {
 9               grailsPlugins()
10               grailsHome()
11               grailsCentral()
12       }
13       dependencies { runtime 'mysql:mysql-connector-java:5.1.5' }
14 }
```

Config

El archivo Config.groovy contiene los parámetros de configuración general de nuestra aplicación. Se trata de un archivo de ConfigSlurper, lo cual permite la declaración de variables y el uso de tipos de datos en la configuración. Cualquier parámetro que definamos en este archivo estará disponible desde cualquier artefacto de la aplicación a través del objeto global grailsApplication.config.

Config.groovy

```
 1 grails.project.groupId = appName
 2 grails.mime.file.extensions = true // enables the parsing of file extensions from
URLs into the request format
 3 grails.mime.use.accept.header = false
 4 grails.mime.types = [ html: ['text/html','application/xhtml+xml'],
```

```
 5                        xml: ['text/xml', 'application/xml'],
 6                        text: 'text/plain',
 7                        js: 'text/javascript',
 8                        rss: 'application/rss+xml',
 9                        atom: 'application/atom+xml',
10                        css: 'text/css',
11                        csv: 'text/csv',
12                        all: '*/*',
13                        json: ['application/json','text/json'],
14                        form: 'application/x-www-form-urlencoded',
15                        multipartForm: 'multipart/form-data'
16                    ]
17 // The default codec used to encode data with ${}
18 grails.views.default.codec = "none" // none, html, base64
19 grails.views.gsp.encoding = "UTF-8"
20 grails.converters.encoding = "UTF-8"
21 ...
```

DataSource

Grails es Java, y por tanto la configuración de acceso a datos recae en última instancia en JDBC. El archivo DataSource.groovy contiene los parámetros de conexión con la base de datos que vayamos a utilizar en cada entorno.

En nuestro caso manipularemos una base de datos MySQL, y solo utilizaremos el ambiente de *development*.

DataSource.groovy

```
 1 dataSource {
 2       pooled = true
 3       loggingSql = true
 4       driverClassName ="com.mysql.jdbc.Driver"
 5       username = "root"
 6       password = "password"
 7       dbCreate = "update"
 8       url = "jdbc:mysql://localhost:3306/ardroid"
 9 }
10 hibernate {
11       cache.use_second_level_cache=true
12       cache.use_query_cache=true
13       cache.provider_class='org.hibernate.cache.EhCacheProvider'
14 }
15 environments {
16       development {
17                dataSource {
18                         dbCreate = "update"
19                         url = "jdbc:mysql://localhost:3306/ardroid"
20                }
21       }
22       test {
23                ...
27       }
28       production {
29                ...
33       }
34 }
```

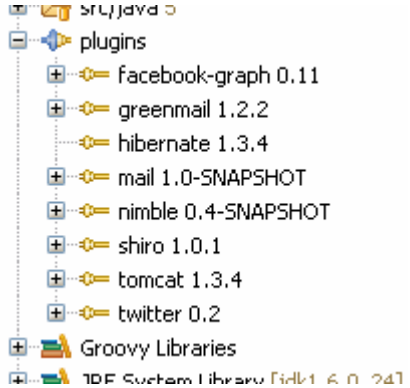
UrlMappings

Para cada petición HTTP, Grails determinará el controlador que debe invocar en función de las reglas fijadas en UrlMappings.groovy, creará una instancia de la clase elegida e invocará la acción correspondiente.

6.6 - FUNCIONALIDADES ADICIONALES

Grails cuenta con un buen soporte para la implementación de plugins. Los plugins son pequeñas aplicaciones que se relacionan con otras para aportarle una función nueva y generalmente muy especifica. Estas aplicaciones adicionales son ejecutadas por la aplicación principal e interactúan entre si.

Pero aparte de incorporar funcionalidad definida por terceros, un aspecto muy importante del sistema de plugins de Grails es que permite diseñar aplicaciones de forma modular, aumentando la calidad de nuestro software en general.



Estructura de plugins de Groovy on Grails

Necesitaremos dos plugins adicionales para nuestro administrador Web.

1. *nimble*: Es un plugin para implementar seguridad a la aplicación de forma automática y transparente. Esto es, no debemos incluir código a nuestra aplicación más que un par de archivos de configuración. Entre otras cosas provee autenticación flexible, control de acceso y perfiles de usuario. Este plugin depende de otros como *mail*, *shiro* y *greenmail* para implementar su funcionalidad.
2. *twitter-plugin*: El plugin de Twitter permite ver y actualizar la información de Twitter para un usuario.
3. *facebook-graph*: Este plugin permite el acceso a la API de Facebook (https://graph.facebook.com) y hace más fácil el desarrollo de inicio de sesión único mediante autenticación de Facebook. Lo utilizaremos para obtener actualizaciones y eventos para alimentar a las entidades automáticamente.

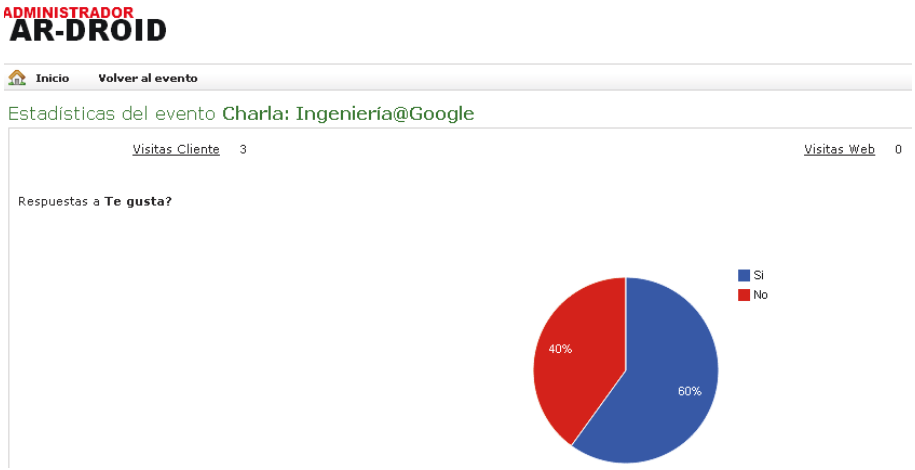
6.7 - ESTADÍSTICAS EN LOS EVENTOS

La implementación del administrador Web incluye un modulo de estadísticas sobre los eventos registrados. La finalidad del mismo es tener un conocimiento de las visitas y valoraciones que los usuarios realizan sobre los eventos visualizados.

La información estadística AR-DROID se divide en dos:

- La cantidad de visitas al evento: Discriminando si son visitas a través del cliente móvil, o es una visita Web siguiendo un enlace de recomendación de un usuario.
- La valoración de los usuarios: Dependiendo del tipo de encuesta parametrizada para el evento, se muestran distintos tipos de gráficos estadísticos.

El modulo de estadísticas fue implementado utilizando la API de Google Chart para construir los gráficos.



Estadísticas para un evento con template de encuesta Pregunta Simple

Para armar los gráficos se utilizan todas las valoraciones realizadas por los usuarios para cada uno de los eventos.

StatsController.groovy

```
 1 class StatsController {
 2       static allowedMethods = [show: "GET"]
 3
 4       def show = {
 5               def eventInstance = Event.get(params.id)
 6               if(eventInstance != null){
 7                       [eventInstance: eventInstance, question:
eventInstance.surveyTemplate.first(), questions:
eventInstance.surveyTemplate.questions, responses: eventInstance.responses]
 8               }
 9               else
10                       redirect(controller: "event", action: "list")
11       }
12
13       ...
20 }
```

Desde la lista de eventos registrados, pueden consultarse las estadísticas para cada uno. Estas se mostrarán de acuerdo al *template* de encuesta definido como:

- MultipleChoiseQuestion: Se muestra un gráfico de torta con el total de las elecciones de los usuarios para cada una de las opciones definidas en el *template* de encuesta.
- NumericValueQuestion: Se muestra en modo grafico de barras, el total de valores de respuesta posibles (1...N) y la suma de las calificaciones de los usuarios.
- TextValuQuestion: Al ser este tipo de valoración de texto libre (como por ejemplo dejar un comentario), se muestra la lista de todos los comentarios registrados para el evento.

6.8 - COMUNICACIÓN ENTRE CLIENTE Y SERVIDOR

Como dijimos, la aplicación cliente AR-DROID se comunica con el servidor de la aplicación para:

1. Obtener información de los distintos puntos de interés a mostrar (Entidades, Eventos, Actividades)
2. Actualizar información de feedback sobre algún evento.

Para lograr la comunicaron entre el cliente y el servidor se implementaron servicios Web bajo REST (Representational State Transfer).

El consorcio W3C define los Servicios Web [44] como sistemas software diseñados para soportar una interacción interoperable maquina a máquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

REST (Representational State Transfer) [42] [43] es un estilo de arquitectura de software para sistemas de Hipermedia distribuidos tales como la Web. El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP. En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP(*) sin una capa adicional, como hace SOAP(*).

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST no es un estándar, está basado en estándares:

- HTTP
- URL
- Representación de los recursos: XML/JSON/HTML/GIF/JPEG/…
- Tipos MIME: text/xml, text/json, text/html, …

La motivación de REST es la de capturar las características de la Web que la han hecho tan exitosa. La Web evidentemente es un ejemplo clave de diseño basado en REST, ya que muchos principios son la base de REST. Posteriormente mostraremos un posible ejemplo real aplicado a Servicios Web.

HTTP posee un interfaz uniforme para acceso a los recursos, el cual consiste de URIs, métodos, códigos de estado, cabeceras y un contenido guiado por tipos MIME. Los métodos HTTP más importantes son PUT, GET, POST y DELETE. Ellos suelen ser comparados con las operaciones asociadas a la tecnología de base de datos, operaciones CRUD: CREATE, READ, UPDATE, DELETE.

Acción	HTTP	SQL
Create	PUT	Insert
Read	GET	Select
Update	POST	Update
Delete	DELETE	Delete

Como crear una interfaz basada en REST? Para esto debemos:

1. Identificar todas las entidades conceptuales que se desean exponer como recursos.
2. Identificar los recursos apropiadamente mediante URIs. Los recursos deberían ser nombres no verbos. Por ejemplo en nuestro sistema para tendremos una URI para recuperar todos las entidades y otra para recuperar una entidad en particular.
    a. entities (URI para recuperar una entidad). URL: http://www.ardroid.com/entities/1
    b. allEntities (URI para recuperar todas las entidades). URL: http://www.ardroid.com/allEntities
3. Elegir el formato de representación: Esta representación puede ser un documento HTML, XML, JSON, una imagen, etc. Por cada recurso se tiene que decir cuál va a ser su representación.
4. Que métodos son soportados en cada URI. Categorizar los recursos de acuerdo con si los clientes pueden obtener una representación del recurso o si pueden modificarlo.

El acceso de puede hacer de muchas formas, recibiendo una representación del recurso (GET o HEAD), añadiendo o modificando una representación (POST o PUT) y eliminando algunas o todas las representaciones (DELETE).

HTTP	CRUD	Descripción
POST	CREATE	Crear un nuevo recurso
GET	RETRIEVE	Obtener la representación de un recurso
PUT	UPDATE	Actualizar un recurso
DELETE	DELETE	Eliminar un recurso

Para nuestro ejemplo podríamos tener los siguientes métodos soportados para los recursos identificados.

Recurso	Método	Representación
entities	GET	Formato de la entidad (JSON)
entities	PUT	Formato de la entidad (JSON)
---	DELETE	No permitido
AllEntities	GET	Formato de la lista de entidades (JSON)
AllEntities	POST	Formato de la lista de empleados (JSON)

Códigos de error que pueden ser devueltos: es necesario enumerar los códigos de estado HTTP típicos que podrían ser devueltos. Ejemplo: HTTP 200, HTTP 404, HTTP 310, etc.

6.8.1. Implementación de servicios Web con REST en Grails

Para implementar REST con Grails [45] se debe proporcionar el mapeo de URLs a través de la clase UrlMappings.groovy.

Es esta clase se podrá definir la URL de cada recurso, métodos permitidos para cada uno y la acción del controlador lo implementa. El mapeo de URLs puede ser configurado para mapear con los métodos HTTP como GET, PUT, POST y DELETE y así obtener una API REST restringida a los métodos HTTP. Cada método HTTP se mapea a una única acción dentro del controlador.

UrlMappings.groovy

```
1  class UrlMappings {
2
3        static mappings = {
4                "/events/$id?"(controller:"request"){
5                        action = [GET:"events"]
6                }
7
8                "/allevents/$token?"(controller:"request"){
9                        action = [GET:"allevents "]
10               }
11
12               "/feedback?"(controller:"request"){
13                       action = [GET: "allFeedback",POST:"save"]
14               }
15       }
16 }
```

Podemos ver en el mapeo del servicio `events` lo siguiente:

1. Que el método HTTP soportado para la solicitud de eventos será un GET, es decir, ante un GET de la forma http: //www.ardroid.com/events/xx se estará solicitando los eventos para

la entidad N indicada como parámetro.

2. El servicio será atendido por el controlador `RequestController` y la acción del controlador que lo implementará será "events".

Las peticiones al servidor las centralizamos en un controlador (`RequestController`), donde el cliente móvil realiza las solicitudes.

RequestController.groovy

```
1  class RequestController {
2        static allowedMethods = [save: "POST", update: "POST", delete: "POST"]
3        def eventService
4
5        def typeEntities = {
6                render TypeEntity.list() as JSON
7        }
8
9        def typeEvents = {
10               render TypeEvent.list() as JSON
11       }
12
13       def entities = {
14               render Entity.list() as JSON
15       }
16
17       def events = {
18               def entity = Entity.get(params.id);
19               render entity.events as JSON
20       }
21
22       def imageEntity ={
23               def entityInstance = Entity.get(params.id)
24               response.contentType = "image/jpeg"
25               response.contentLength = entityInstance?.photo.length
26               response.outputStream.write(entityInstance?.photo)
27       }
28       ...
29       def feedback ={
30               def json = request.JSON
31               def event = Event.get(json.event.id)
32
33               println json.toString()
34               if (!event.hasErrors()){
35                       def survey = new SurveyResponse()
36                       survey.fecha = new Date()
37
38                       survey.createResponsesFromJSON(event,json)
39
40                       if (!survey.hasErrors())
41                               eventService.saveResponse(survey)
42               }
43       }
44 }
```

Vemos como a la clase de dominio `Entity` le pedimos que nos retorne la lista de entidades guardadas. Groovy nos provee un mecanismo sencillo para convertir cualquier tipo de objetos a su cadena alfanumérica JSON, a través de la sentencia `as JSON`.

Vemos a continuación como responde el servidor a la solicitud de entidades por parte del cliente:

```
[ { "description" : "La Facultad de Informática, creada en 1999, es el producto de un
proceso del crecimiento y desarrollo del departamento de informática que existía en la
Facultad de Ciencias Exactas.",
    "geoPoint" : { "altitude" : 19.68,
        "class" : "ar.droid.admin.GeoPoint",
        "latitude" : -34.9034586930727,
        "longitude" : -57.937646554706703
      },
    "id" : 1,
```

```
    "name" : "Facultad de Informática",
    "photoUrl" : "/entity/showImage/1",
    "readerNews" : { "class" : "ar.droid.admin.reader.ReaderNewsRSS",
        "entity" : { "class" : "Entity",
            "id" : 1
          },
        "id" : 3,
        "parameter" : "http://info.unlp.edu.ar/rss/NONE"
      },
    "typeEntity" : { "description" : "Facultad",
        "iconUrl" : "/typeEntity/showIcon/2",
        "id" : 2
      },
    "url" : "http://www.info.unlp.edu.ar"
  },
  ...
]
```

Este intercambio de información re realiza mediante la tecnología JSON (*) (JavaScript Simple Object Notation) que es un formato ligero para el intercambio de datos. Está basado en un subconjunto del lenguaje de programación JavaScript y se utiliza sustituyendo a XML(*), esta basado en el estándar ECMA-262 [46] [47]

Hemos elegido JSON por sobre XML por distintas razones:

- La simplicidad de JSON, leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo, manipularlo y generarlo. Es mucho más simple que XML ya que posee una gramática más pequeña y mapea más directamente con las estructuras de datos usadas en los lenguajes de programación.
- Es mucho más ligero que XML para el intercambio de datos sobre la red.
- Por su estructura resulta mucho más rápido y sencillo procesarlo.

Diferencias entre JSON y XML:

- XML es una manera de estructurar datos, mientras que JSON es una manera de intercambiar datos.
- Un archivo XML es un conjunto de datos estructurado. Como tal, admite consultas (xpath), tiene una estructura fácilmente comprobable (DTD, XML Schema), puede visualizarse (CSS), procesarse (XSL), etc., mientras que un archivo JSON es un conjunto de datos agrupados, puede tener una estructura jerárquica, pero lo único que tienes son objetos, vectores, variables y valores.

JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, Microsoft, etc., que atienden a millones de usuarios) y cuando la fuente de datos es explícitamente de fiar.

Esta tecnología cuenta con diferentes APIs para Java y otros lenguajes que convierten en algo automático la lectura y composición de estos tipos de objetos. Hemos elegido la librería google-gson para utilizar del lado del cliente y en el caso del servidor se ha utilizo la provista por el Framework Grails (grails.converters.JSON, grails.converters.deep.JSON).

Grails permite redefinir la forma que se convierten los objetos a JSON a través de conversores, que permite adaptar la conversión a necesidades específicas. El siguiente es un conversor de nuestro `Entity` que se define en el archivo BootStrap.

BootStrap.Groovy

```
 1 ...
 2 grails.converters.JSON.registerObjectMarshaller(Entity) {
 2       def returnArray = [:]
 3       returnArray['id'] = it.id
 4       returnArray['name'] = it.name
 5       ...
 9       returnArray['geoPoint'] = it.geoPoint
10       returnArray['readerNews'] = it.readerNews
11       return returnArray
12 }
13 ...
```

6.9 - SUMINISTRO AUTOMÁTICO DE EVENTOS

Hoy en día la integración con redes sociales es una herramienta de muchísimo potencial en cualquier aplicación en la Red.

La presencia en las redes sociales ya no es una opción, sino la forma actual de posicionarse. Hoy por hoy, las personas pasan más tiempo conectadas al Internet que a cualquier otro medio y básicamente interactuando socialmente, los usuarios son cada vez más proactivos, expresan opiniones y marcan sus preferencias sobre productos, servicios, con una libertad de expresión e interacción, que los medios convencionales no les permitían.

Los usuarios son más amenos a la hora de requerir un servicio restringido por medio de redes sociales, así se evitan tediosos registros y validaciones de acceso. Del lado de la aplicación, es ventajoso porque se resuelven varios objetos de dominio, como registro de usuarios, automáticamente. También se beneficia del alto impacto de la "comunicación viral", donde son los propios usuarios los que se encargan de distribuir los contenidos. Aquí ahí que comprender la importancia de que dichos contenidos sean de un verdadero interés y motiven a los usuarios a compartirlo con sus contactos.

En las redes sociales, el contexto académico no es la excepción. No solo vemos la presencia oficial de los centros de estudios, sino también vemos como los propios alumnos se agrupan entre mismas unidades académicas o actividades en común.

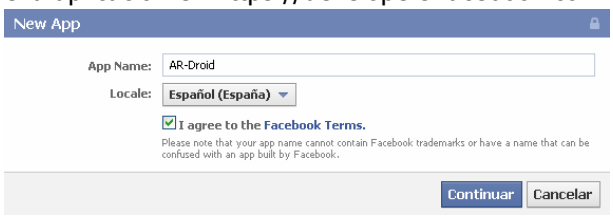
6.9.1 - Integración con Facebook

Uno de los lectores de actividades que implementaremos será el lector de eventos de la red social Facebook. Seleccionamos primeramente Facebook, porque tiene una estructura de eventos similar a la nuestra, y por sobre todo porque los eventos definidos pueden geolocalizarse, algo que resulta fundamental para nuestra aplicación.

Requerimos de dos pasos fundamentales para la integración: Crear una aplicación en Facebook, y desarrollar un cliente de integración con esta.

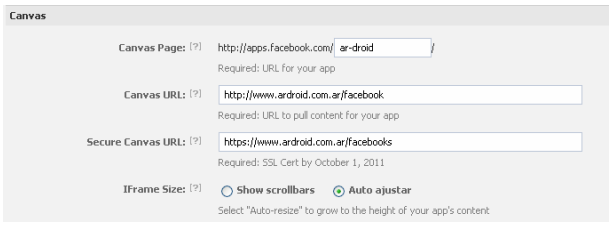
<u>Crear una aplicación de Facebook</u>

1. Crear una nueva aplicación en https://developers.facebook.com

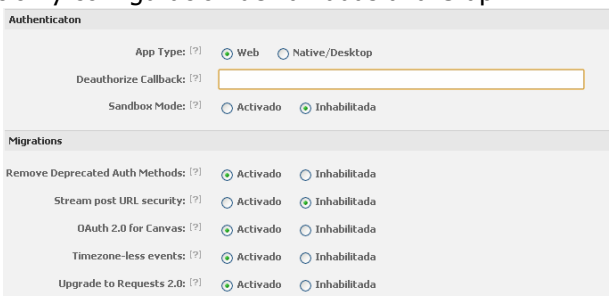


2. Una vez creada la aplicación, se nos asignara un App ID (código de aplicación) y un App Secret (código secreto) que serán nuestro parámetros de conexión a la aplicación.

3. Configuramos los parámetros de la aplicación, como pagina de entrada (*Canvas Page*) y pagina que visualizará la aplicación (*Canvas URL*)



4. Por ultimo revisamos las configuraciones avanzadas de la aplicación, tales como métodos de autenticación y configuración de llamadas a la Graph API.



El procedimiento de nos resultó sencillo y cuenta con mucha documentación disponible. El proceso es prácticamente automático y nos permitió crear la aplicación en un par de minutos.

Con esto ya tenemos configurada nuestra nueva aplicación y apuntada hasta nuestro servidor AR-Droid. Aclaramos ahora que nuestra aplicación con cara en Facebook no tendrá funcionalidad. Solo la creamos para poder interactuar con el API y permitir a nuestro administrador Web conectar con Facebook.

Ahora vamos a configurar y desarrollar la interacción con la aplicación por medio de nuestro administrador Web.

Como nombramos anteriormente, utilizaremos el plugin para Grails *facebook-graph* para interactuar con Facebook. Este plugin nos permite establecer una conexión de usuario y manipular la API de manera sencilla encapsulándola en un objeto Groovy.

Config.groovy

```
// facebook connection data
facebook.applicationSecret = 'd425a00090fccaa138bbeb7ba995f283'
facebook.applicationId = '224376114269839'
facebook.secure = false
```

Crearemos un enlace en nuestro administrador para autorizarlo a utilizar la aplicación que hemos creado en Facebook:

index.gsp

```
<script>
      FB.init({appId: '224376114269839', cookie: true, xfbml: true, status: true});
      function facebookLogin() {
            FB.getLoginStatus(function(response) {
                  if (response.session) {
                        // nada que hacer
                  }
            });
     }
</script>
<fb:login-button perms="email,publish_stream" onlogin="facebookLogin();"
size="large">Autorizar aplicaciones Facebook"</fb:login-button>
```

Lo que nos da por resultado un botón de acción:

Lo siguiente es desarrollar la integración con Facebook para solicitarle los eventos de determinado grupo. Nuestras entidades cuentan con la parametrización de que lector de actividades utilizaran. En el

caso de Facebook, se debe ingresar el *GID* (identificador de grupo) que identifica al grupo del cual queremos extraer los eventos. Por ejemplo en el caso del grupo "Facultad de Informática - UNLP", el gid es 34432157906 (este identificador es fácilmente reconocible porque esta en la URL de la pagina de entrada al grupo).

Leer los eventos de Facebook

Para leer los eventos de Facebook y poder incorporarlo al sistema lo primero que debemos hacer es autenticarnos, esto se logra integrando nuestra aplicación con Facebook como se describió anteriormente.

Al autenticarnos se generara un token de acceso que nos permitirá invocar a las APIs REST de Facebook como Grahp API (API REST) añadiendo el parámetro de acceso obtenido en la URL del requerimiento, ejemplo: https://graph.facebook.com/me?access_token=ACCESS_TOKEN.

El método de autenticación utilizado por la plataforma Facebook es OAuth (Open Authorization) es un protocolo abierto, que permite autorización segura de un API de modo estándar y simple para aplicaciones de escritorio, móviles, y Web.

Formato de eventos retornados por Facebook

Veamos a continuación como se recuperan los eventos de Facebook y se registran en el sistema.

ReaderActivityService.groovy

```
 1 class ReaderActivityService {
 2       def facebookGraphService
 3
 4       def synchronizeEventsFacebook(entityInstance, params){
 5               def jsonArray = facebookGraphService.api("method/events.get",
params.facebookdata,["uid":entityInstance.readerActivity.parameter,"format":"json"],"G
ET","api")
 6               jsonArray.each {
 7                       ...
 8                       def ev1 = Event.findByEid(it.eid)
 9                       if (ev1 == null){
10                               GeoPoint geoPoint = entityInstance.geoPoint
11                               if (it.venue.longitude){
12                                       def latitude = new
BigDecimal(it.venue.latitude)
13                                       def longitude = new
BigDecimal(it.venue.longitude)
14                                       geoPoint = new GeoPoint(latitude: latitude,
longitude: longitude, altitude: 0)
15                               }
16
17                               Event event = new Event(title: it.name, eid: it.eid,
description: it.description,
18                                                       place: it.location, typeEvent:
TypeEvent.get(1), geoPoint: geoPoint, entity: entityInstance,
19                                                       surveyTemplate:
SurveyTemplate.get(1), eventCalendar: new Daily(startDate: d1, endDate: d2))
20
21                               event.save(flush: true)
22                               println event.hasErrors()
23                       }
24               }
25       }
26
27       def synchronizeEventsDroid(entityInstance, params){
28               ...
29       }
30 }
```

Podemos ver la invocación al método API de la clase *facebookGraphService* que es el encargado de realizar la conexión al servicio y devolver el resultado en una representación JSON.

Los eventos retornados se registran en el sistema si es que no existen, este control se logra a través del campo *eid* que representa unívocamente los eventos de Facebook.

FacebookGraphService.groovy

```
class FacebookGraphService {
	def grailsApplication
	boolean transactional = false

	/**
	 * This method validates a facebook session
	 */
	def validateSession(facebookData) {
		...
	}

	/**
	 * Invoke the Graph API.
	 *
	 * @param String $path the path (required)
	 * @param String $method the http method (default 'GET')
	 * @param Array $params the query/post data
	 * @return the decoded response object
	 * @throws FacebookGraphException
	 */
	def api(path, facebookData, params = [:], method = 'GET',apiName='graph') {
		def exception
		def result
		params.method = method // method override as we always do a POST

		if(facebookData) { // without a facebook session we'll return null
			result = oauthRequest(getUrl(apiName, path), params,
facebookData)
			if(!result) throw new FacebookGraphException()
			else result = JSON.parse(result)

			log.error("Result: ${result}")

			def error = null
			if (result instanceof JSONArray){
				error = result.error[0]
			}
			else error = result.error

			// results are returned, errors are thrown
			if (error) {
				exception = new FacebookGraphException(error)
				if (exception.type == 'OAuthException') {
					invalidateFacebookData()
				}
				throw exception
			}
		}

		return result
	}

	/**
	 * Make a OAuth Request
	 */
	private def oauthRequest(url, params, facebookData) {
		if (!params['access_token']) {
			params['access_token'] = facebookData['access_token'];
		}

		// json_encode all params values that are not strings
		params.each{key,value ->
			if(!(value instanceof String)) params[key] = value as JSON
		}

		return makeRequest(url, params)
	}
	...

}
```

Lo que hace esta clase es, conectarse al servicio de Facebook, obtener la información solicitada y retorna la respuesta en formato JSON.

El servicio se identifica con el parámetro *API* que representa al servidor (por defecto se conecta a https://graph.facebook.com pero podría ser otro ya que no es el único que provee Facebook) y *path* que representa el servicio. Luego se incorpora al servicio los parámetros informados en *params* y el token de acceso, que como dijimos, es el que nos habilitara a consumir información de Facebook.

Por último se realiza la conexión al servidor de Facebook mediante un HTTP GET o POST (por defecto GET) y el resultado obtenido será convertido a formato JSON (convertidor Grails grails.converters.JSON)

6.9.2 - Definición de API de eventos

Definimos una API para obtener los eventos para una entidad de manera automatizada. Básicamente, la finalidad de esta es proveedor un mecanismo de alimentación de eventos para las entidades definidas en el sistema.

Podemos describir el API desde sus tres componentes principales:

- La configuración de la entidad
- Una definición XML
- La llamada al proveedor de eventos

Este mecanismo permite implementar con bajo costo un proveedor de eventos para la entidad, que llamaremos *proveedor externo* para con confundir con nuestro proveedor. El sistema de administración de AR-DROID sincronizará frecuentemente el contenido de los proveedores externos con el contenido local de eventos para la entidad, mediante una petición HTTP.

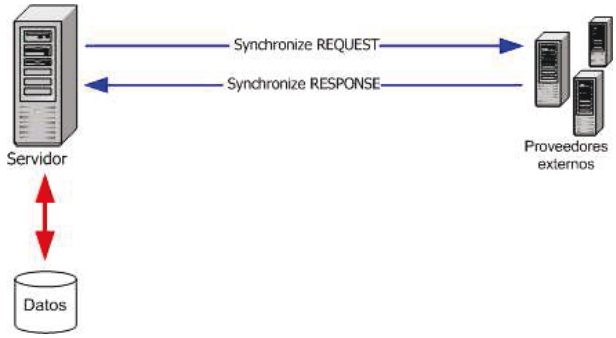


Figura 21. Arquitectura de sincronización por API de eventos

La configuración de la entidad requiere que esta utilice como lector de eventos el *API AR-DROID*, y que se indique a que dirección URL peticionar los eventos.

La definición XML se describe más abajo. Esta es una estructura que debe implementar el proveedor externo para alimentar a una entidad, que consta de una lista de eventos.

La llamada al proveedor externo se realiza periódicamente desde el administrador Web para verificar la existencia de nuevos eventos y sincronizarlos con la base de datos local. Esta llamada se hace enviando dos parámetros de configuración adicionales para chequeos por parte del proveedor externo:

- *last_time_synchronization*: timestamp(*) que representa la última vez que se sincronizó con éxito con el proveedor externo.
- *last_synchronization_id*: número entero que representa el mayor identificador que se sincronizo con éxito con el proveedor externo.

Cuando la llamada al proveedor externo tiene existo, se recupera la lista de nuevos eventos. Estos se convierten al modelo local de AR-DROID y se almacenan en la base de datos estando inmediatamente disponibles para los clientes. Al terminar esta tarea se actualizan los datos de control para la entidad: se toma el máximo ID recibido desde la lista de eventos y la fecha y hora en que termino el proceso de sincronización.

A continuación mostramos la estructura que debe cumplir la respuesta por parte del proveedor de servicios, que debe ser del tipo mime *text/xml*.

Estructura de definición de eventos

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE events SYSTEM "http://api.ardroid.com.ar/1.0/API.dtd">
<events xmlns="http://api.ardroid.com.ar/1.0/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://api.ardroid.com.ar/1.0/
http://api.ardroid.com.ar/1.0/facebook.xsd">
      <!-- id must be unique and asending numeric value -->
      <event id="{id}">
            <title><![CDATA[{title}]]></title>
            <description><![CDATA[{description}]]></description>
            <place><![CDATA[{place}]]></place>
            <!-- geoPoint is optional -->
            <geoPoint>
                  <latitude>{latitude}</latitude>
                  <longitude>{longitude}</longitude>
            </geoPoint>
            <!-- {eventCalendar_type} must be predefined constant -->
            <eventCalendar type="{eventCalendar_type}">
                  <startDate>{startDate}</startDate>
                  <endDate>{endDate}</endDate>
                  <dayOfMonth>{dayOfMonth}</dayOfMonth>
                  <dayOfWeek>{dayOfWeek}</dayOfWeek>
            </eventCalendar>
            <!-- {photo} must be absolute URL to an image - optional -->
            <photo><![CDATA[{photo}]]></photo>
            <!-- {typeEvent} must be predefined constant -->
            <typeEvent>{typeEvent}</typeEvent>
            <activities>
                  <!-- list of event activities -->
                  <activity>
                        <name><![CDATA[{name}]]></name>
                        <description><![CDATA[{description}]]></description>
                        <!-- geoPoint is optional -->
                  <!-- by default is the position of parent event -->
                        <geoPoint>
                              <latitude>{latitude}</latitude>
                              <longitude>{longitude}</longitude>
                        </geoPoint>
                        <!-- {typeActivity} must be predefined constant -->
                        <typeActivity>{typeActivity}</typeActivity>
                  </activity>
            </activities>
      </event>
</events>
```

A continuación describimos cada etiqueta XML:

- `<event id="{id}">`: *Numérico*. Es la cabecera de cada evento a sincronizar. El atributo `{id}` define el identificador único y ascendente para el eventos.
- `<title>`: *Alfanumérico*. El titulo del evento.
- `<description>`: *Alfanumérico*. La descripción del evento.
- `<place>`: *Alfanumérico*. Lugar particular (aula o sala) donde tendrá lugar el evento.
- `<geoPoint>`: *Compuesto*. Las coordenadas del evento. *Opcional*, si no se especifica se toma la posición de la entidad como lugar del evento. Las etiquetas que contienen `{latitude}` y `{longitude}` representan el lugar del evento en formato latitud y longitud respectivamente.
- `<eventCalendar type="{eventCalendar_type}">`: *Compuesto*. Tipo de periodicidad del evento. El atributo `{eventCalendar_type}`: define una constante de tipo de periodicidad, que puede ser una de estas [UNIQUE, DAILY, WEEKLY, MONTHLY]. Además esta entidad compuesta con los siguientes etiquetas:
  - `<startDate>`: *Timestamp*. Fecha y hora de inicio del evento.
  - `<endDate>`: *Timestamp*. Fecha y hora de fin del evento.
  - `<dayOfMonth>`: *Timestamp*. Día del mes en que se repite el evento. *Opcional*, solo se requiere si la periodicidad del evento es MONTHLY.
  - `<dayOfWeek>`: *Timestamp*. Día de la semana en que se repite el evento.

Opcional, solo se requiere si la periodicidad del evento es WEEKLY.

- `<photo>`: *Alfanumérico*. Dirección URL de la imagen representativa del evento. *Opcional*.
- `<typeEvent>`: *Numérico*. Identificador del tipo de evento parametrizado en el administrador.
- `<activities>`: *Compuesto*. Lista que contiene las actividades del evento. Esta formada por:
  - `<name>`: *Alfanumérico*. El nombre para la actividad.
  - `<description>`: *Alfanumérico*. Descripción para la actividad.
  - `<geoPoint>`: *Compuesto*. Ídem que el geoPoint del evento. *Opcional*, si no se especifica se toma la posición del evento padre.
  - `<typeActivity>`: *Numérico*. Identificador del tipo de actividad parametrizado en el administrador.

El implementador de proveedores de servicios debe tener en cuanta los parámetros que se envían *last_time_synchronization* y *last_synchronization_id* para evitar proveedor de eventos repetidos al administrador. Vale la aclaración ya que el administrador no realiza ningún tipo de chequeo de duplicidad, simplemente recoge los eventos y los agrega a la base de datos.

7 - IMPLEMENTACIÓN DEL CLIENTE MÓVIL

El cliente móvil es la pieza que completa la arquitectura AR-DROID. Es el software que sirve de cara al usuario final para consumir la información que e vuelca en el administrador Web.

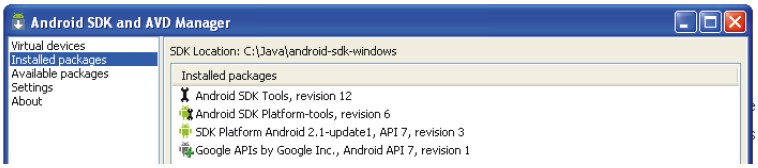
Dadas las restricciones de procesamiento y memoria de los dispositivos móviles, el administrador Web se encargará de recibir las peticiones del cliente, procesar la información (con el propósito de evitar que el cliente tenga que realizar cálculos o conexiones demandantes) y devolverle resultados estandarizados al cliente.

En este capítulo se presentará el diseño e implementación de los distintos componentes necesarios para crear el prototipo de cliente. Expondremos también las características más importantes que encontremos en el transcurso del desarrollo en cuando al desarrollo sobre la plataforma.

Aclaramos en este punto, que los códigos presentados pertenecen al prototipo real, pero han sido fragmentados y modificados a modo de demostrar las particularidades de de desarrollo que se encontraron en el proceso.

7.1 - CREANDO EL PROYECTO ANDROID

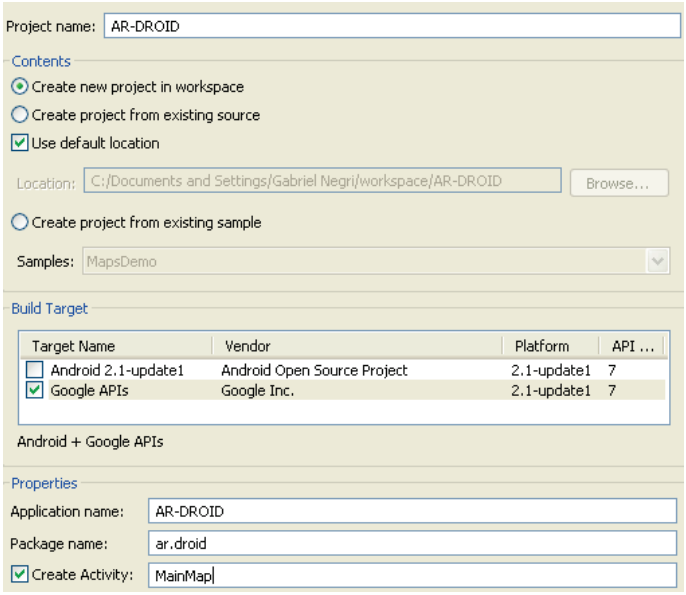
Android ofrece un plugin para Eclipse, el *ADT plugin*, que extiende la funcionalidad de éste y facilita el desarrollo de aplicaciones para Android. Una vez instalado (como se describe en el apéndice *A - PREPARACIÓN DE AMBIENTE*) debemos descargar las SDK que seleccionemos, y las herramientas para acompañar el desarrollo desde el *Android SDK and AVD Manager*.



Para crear nuestra aplicación Android desde Eclipse, debemos ir a "File" -> "New" -> "Project" y seleccionar "Android Project".

A la hora de crear un nuevo proyecto, debemos tener en cuenta diferentes aspectos en cuanto a la nomenclatura:

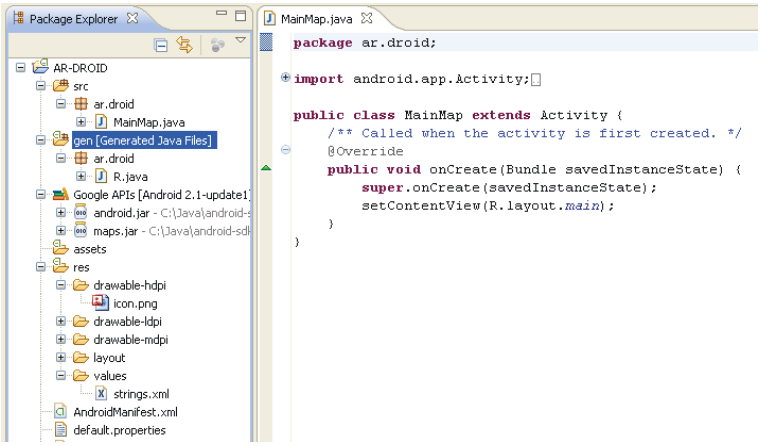
- Project Name: es un identificador para Eclipse; de él dependerá la carpeta en nuestro espacio workspace
- Build target: es la versión de SDK con el cual se construirá la aplicación que desarrollaremos.
- Application Name: será el nombre de nuestra aplicación, cómo aparecerá en el dispositivo.
- Package Name: el namespace inicial de nuestro proyecto será ar.droid
- Por último, tenemos la versión mínima del SDK para indicarnos qué dispositivos pueden abrir nuestra aplicación.



Creando el proyecto Eclipse Android

En nuestro caso utilizaremos un *build target* que contiene las librerías comunes de Google, distribuida por este mismo, incluyendo la librería de Google Maps nativa para Android.

La versión mínima de SDK debe ser igual o mayor al build target que hayamos seleccionado. Este número está en correspondencia con el API Level; por ejemplo, la versión 1.6 corresponde al nivel 4 de la API, y la 2.1 al nivel 7.



Estructura de nuestro nuevo proyecto Android

Una vez creado nuestro proyecto estamos en condiciones de empezar a desarrollar las distintas funcionalidades con las que contará el cliente.

En resumen, vemos que la estructura del proyecto contiene nuestras fuentes de nuestra aplicación, la clase *R* auto generada, la SDK de Google APIs, los recursos que utilizaremos, y el *AndroidManifiest.xml*.

Describimos este último archivo, que tiene una importancia significativa dentro de cualquier proyecto Android.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemás.android.com/apk/res/android"
        package="ar.droid" android:versionCode="1" android:versionName="1.0">
        <uses-sdk android:minSdkVersion="7" />

        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".MainMap" android:label="@string/app_name">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category
android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
        </application>

        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

Esta es nuestra primera versión del archivo de *manifiesto* de aplicación. En el definimos varias cuestiones de nuestra aplicación:

- `uses-sdk` indica que la aplicación correrá como mínimo sobre dispositivos con Android 2.1 (SDK ver. 7).
- `application` es la sección que describe las características básicas de la aplicación y sus actividades.
- `activity` define cada actividad que se implemente y pueda instanciarse en la aplicación
- `intent-filter` especifica los propósitos de una actividad o servicio. Dentro de este, la propiedad `category "android.intent.category.LAUNCHER"`, indica que esta será la actividad principal de la aplicación.
- `uses-permission` especifica que la aplicación tendrá acceso a determinados permisos (previamente aceptados por el usuario) a acceso a funcionalidades del dispositivo.

Recordamos que los recursos (*Resources*) son elementos externos que se quieren incluir y referenciar en la aplicación (imágenes, videos, audio, textos, layouts). Se incluyen declarativamente (carpeta */res*) accediéndose como *@<tipo>/<nombre>*.

7.1.1 - La actividad principal

Comenzamos por aquí porque es donde se define la actividad inicial que constituye la interfaz de usuario. La actividad principal es la cara visible del proyecto, punto de partida desde donde los usuarios realizan las peticiones que ponen en marcha las demás funcionalidades.

Las actividades son probablemente el elemento más común en una aplicación para Android. Una actividad se puede definir como una única pantalla o interfaz dentro de una aplicación. Las actividades mostrarán una interfaz de usuario basada en vistas (*Views*) y responderá a eventos.

La actividad principal de la aplicación cliente será de tipo mapa, para que la primera percepción que tenga el usuario sea una visión general del circuito académico. Y como lo hemos descrito, este mapa incluirá en una primera instancia, las entidades que se conocen en el área donde se encuentra el usuario.

Lo primero que necesitamos entonces es, utilizando la librería Google APIs nombrada anteriormente, la implementación de una actividad mapa. Para esto, primeramente describiremos que es esta librería y como integrarla desde nuestro proyecto.

7.1.2 - Google APIs

La utilización del popular servicio Google Maps es una de las posibilidades más atractivas que encontramos en Android. Justamente, nos topamos con un gran número de proyectos desarrollados que utilizan estas bibliotecas con fines muy distintos [48].

La librería es realmente completa, incluye todas las clases relacionadas con la carga y manejo de mapas directamente desde Google Maps.

Dentro del paquete *com.google.android.maps* podemos encontrar clases para todas las funcionalidades:

- *MapView*: obtiene el mapa solicitado y lo muestra en pantalla.
- *MapController*: gestiona el manejo de un mapa, como desplazamientos o zoom.
- *GeoPoint*: clase que representa un punto geográfico determinado del mapa, según su latitud y longitud.
- *Overlay*: capa superpuesta que permite dibujar y representar elementos sobre el mapa.
- *MapActivity*: clase muy relevante que extiende la clase base Activity, y permite crear una Activity específica para gestionar mapas.

Mostramos a continuación un breve diagrama de las relaciones que existen entre estas y otras clases del API de Google que nos serán útiles en el transcurso del desarrollo del prototipo cliente:

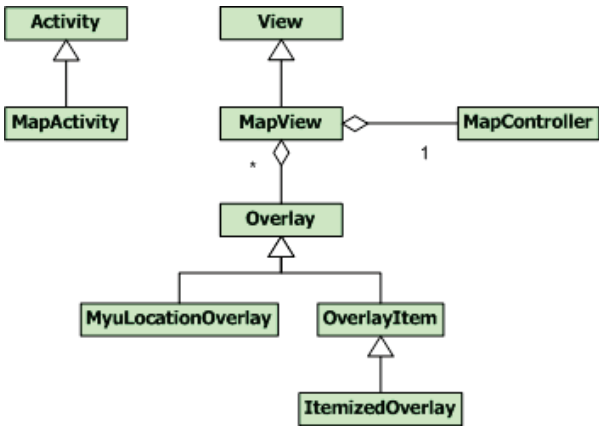


Figura 22. Diagrama de clases básico de la actividad principal AR-DROID

Previo a la utilización de este API para manipular mapas necesitamos *registrar* nuestra aplicación en Google Maps y aceptar los términos y condiciones de uso. Este registro se realiza a través de una clave *API Key*.

Obtener la API KEY

Toda aplicación en Android está acompañada de un certificado que asegura su autoría y la vincula con su desarrollador. Al utilizar el plug-in de Android para Eclipse, todas las aplicaciones están firmadas

por un certificado que permite a los desarrolladores poder crear y probar sus aplicaciones. Este certificado se obtiene a través del archivo de claves *debug.keystore*, presente por defecto en el SDK de Android.

El registro para tener acceso a Google Maps se hace a través de dicho certificado. Utilizando el archivo *debug.store*, tendremos que crear un certificado propio:

1. Obtenemos el MD5 del certificado que se va a usar para firma la aplicación donde usaremos mapas. El MD5 lo podemos obtener con la herramienta keytool presente en el SDK de Java:

```
> keytool -list -alias androiddebugkey -keystore debug.keystore

Huella digital de certificado (MD5):
0B:41:31:DC:4B:89:22:8E:A5:45:79:C6:13:DA:7E:D4
```

2. A continuación, registramos el MD5 obtenido en Google Maps, a través de la Web de registro, http://code.google.com/intl/es-ES/android/maps-api-signup.html. Es necesario disponer de una cuenta Google activa.
3. Google Maps nos proporciona una clave alfanumérica, llamada *API key*. Esta es la que debemos utilizar para manejar mapas desde Andorid, siendo además única y exclusiva para la aplicación firmada con certificado utilizado.

```
Tu clave es:
     1hT45Czgki8Q9NNmuPtK5olQ97m9mV3_Vt2CY9i

Esta clave es válida para todas las aplicaciones firmadas con el certificado
cuya huella dactilar sea:
     0P:54:31:DC:E4:89:32:8E:A4:45:88:C6:12:ad:7E:D1
```

Esta clave para acceder al API de Maps, la incluiremos posteriormente en la actividad que implementemos para la utilización de mapas.

Por último, necesitamos especificar en nuestra aplicación que vamos a necesitar en tiempo de ejecución de la librería de mapas, lo que hacemos introduciendo la siguiente línea en el archivo *AndroidManifest.xml*:

```
<uses-library android:name="com.google.android.maps" />
```

7.1.3 - Implementación de la actividad

Cada actividad se implementa como una única subclase que hereda de la clase *android.app.Activity*. De esta forma, no necesitamos inquietarnos de muchos de los aspectos relacionados con la tarea de las actividades ya que por herencia nuestra actividad contendrá ya esa funcionalidad.

Una vez disponemos de la *API key* con la que poder acceder a los mapas y servicios de Google Maps, nos queda implementar las clases necesarias del paquete *com.google.android.maps*.

La clase MainMap, será la clase principal y la que instancia gran parte de las demás funcionalidades del sistema. Esta clase hereda de *MapActivity*, que a su vez hereda *Activity*, por lo que finalmente nuestra clase será una actividad.

Entonces, para poder visualizar un map, necesitaremos básicamente dos objetos:

- Extender la clase MapActivity
- Un objeto MapView, que obtiene y representa el mapa

La interfaz principal del prototipo consiste en la vista completa de un mapa donde el usuario visualiza la ubicación de entidades, puede desplazarse libremente, realizar cambios en el zoom, o bien pulsar en una entidad para centrar el foco en ella.

MainMap.java

```
1 public class MainMap extends MapActivity {
2       private MapView mapView;
3
```

```
 4       @Override
 5       public void onCreate(Bundle savedInstanceState) {
 6               super.onCreate(savedInstanceState);
 7               setContentView(R.layout.mainmap);
 8
 9               // crear mapa
10               mapView = (MapView) findViewById(R.id.mapview);
11               mapView.displayZoomControls(true);
12               mapView.setBuiltInZoomControls(true);
13               mapView.getController().setZoom(17);
14       }
15
16       @Override
17       protected boolean isRouteDisplayed() {
18               return false;
19       }
20 }
```

Describimos las particularidades de esta simple clase:

- `onCreate` es el método que se ejecuta al crear la actividad.
- La clase MapActivity nos obliga a implementar el método `isRouteDisplayed` donde tenemos que indicar si el mapa esta mostrando una ruta.
- En la línea 7 vemos una llamada a `setContentView`, esto indica que vista utilizará la actividad, lo que explicaremos a continuación.
- Obtenemos y le damos aspectos básicos de visualización a una instancia de `MapView`, línea 10-13.
- `mapView`, contiene una instancia de `MapController`, otro objeto importante que nos ayudará a manipular el mapa.

7.1.4 - Creando la vista de la actividad

Mediante la llamada a `setContentView` de la clase Activity, le decimos a Android qué vista queremos mostrar en la pantalla. La vista se referencia mediante el archivo de recursos (*R.java*). En este archivo describimos la interfaz de usuario como fichero XML, de forma similar a como lo haríamos en un archivo HTML.

En este caso se utiliza un layout llamado `RelativeLayout` (que dispone todos los elementos de la interfaz se aliñen cada una con su padre o consigo misma) y dentro una etiqueta `com.google.android.maps.MapView`, que muestra un mapa en pantalla. Notemos que hay una larga lista de tipos de layouts y de componentes de interfaz, como botones, listas, etc...

La definición inicial de nuestro archivo de definición de vista quedaría de la siguiente manera:

mainmap.xml

```
<RelativeLayout xmlns:android="http://schemás.android.com/apk/res/android"
       android:id="@+id/mainlayout" android:orientation="vertical"
       android:layout_width="fill_parent" android:layout_height="fill_parent">

       <LinearLayout android:id="@+id/zoomview"
              android:layout_width="wrap_content" android:layout_height="wrap_content"
              android:layout_alignBottom="@id/mapview"
              android:layout_centerHorizontal="true" />

       <com.google.android.maps.MapView
              android:id="@+id/mapview" android:layout_width="fill_parent"
              android:layout_height="fill_parent" android:clickable="true"
              android:apiKey="1hT45Czgki8Q9NNmuPtK5olQ97m9mV3_Vt2CY9i" />
</RelativeLayout>
```

Nombramos las dos características más importantes de este tipo de archivos. En este caso particular, donde tenemos un elementos de interfaz `MapView`, debemos completar la propiedad `android:apiKey` con nuestro *API key* generado para nuestro ambiente de desarrollo.

En general, tenemos la propiedad `android:id="@+id/mapview"` que nos sirve para referencia

el recurso desde el archivo R.java.

Así es como se verá la aplicación una vez ejecutada, tanto en una maquina virtual, como en un dispositivo Android real.

Vemos arriba a la derecha un texto que nos indica nuestra dirección (al menos con un rango de menos de 25 metros). Esto lo hacemos determinando la a través del API *Geocoder* [49] de Google. Describiremos esta más adelante cuando hablemos de las funcionalidades extras implementadas.

7.1.5 - Menús de las actividades

Android nos permite definir tres tipos diferentes de menús:

- Menús Principales: Son los más habituales, aparecen en la zona inferior de la pantalla al pulsar el botón *menú* del teléfono.
- Submenús: Son menús secundarios que se pueden mostrar al pulsar sobre alguna opción desde el menú principal.
- Menús Contextuales: Son los que aparecen al realizar una pulsación *larga* sobre algún elemento de la vista. Resultan útiles en muchas ocasiones, al poder simular el botón derecho del Mouse.

Tenemos dos alternativas para crear nuestros menús de las actividades. La primera es definiendo el menú directamente en el código, y la segunda es mediante la definición del menú en un fichero XML. Nosotros utilizaremos la primera, ya que es la manera más limpia y la recomendada por Google.

res/menu/menu_map.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <menu xmlns:android="http://schemás.android.com/apk/res/android">
 3       <item android:id="@+id/menu_ar" android:icon="@drawable/icon_ra"
 4               android:title="Vista RA" />
 5       <item android:id="@+id/menu_position" android:icon="@drawable/icon_posicion"
 6               android:title="Mi posición" />
 7       <item android:id="@+id/menu_find" android:icon="@drawable/icon_buscar"
 8               android:title="Buscar" />
 9       <item android:id="@+id/menu_events" android:icon="@drawable/icon_mostrar"
10               android:title="Eventos" />
11       <item android:id="@+id/menu_entities" android:icon="@drawable/icon_event"
12               android:title="Entidades" />
13       <item android:id="@+id/menu_close" android:icon="@drawable/icon_cerrar"
14               android:title="Salir" />
15 </menu>
```

Vemos que la definición del menú es bastante comprensiva, varios elementos `<item>` bajo un elemento `<menu>`. En cada elemento ítem especificamos:

- `android:id`: (obligatorio) es el identificador único dentro del menú.
- `android:icon`: (opcional) ID del icono representativo del menú. Como todos los recursos, los iconos utilizados deberán estar en la carpeta *res/drawable*.
- `android:title`: (obligatorio) Es el valor textual que mostrar la opción en el menú. Esta puede ser una cadena de texto, o un ID de recurso tipo *string* para soportar traducción.

En nuestra actividad, debemos ahora implementar dos métodos de la superclase *Activity*. Estos eventos nos van a permitir por un lado indicar que menú mostrar, y por otro que acciones tomar al seleccionarse una opción.

```
 1 @Override
 2 public boolean onCreateOptionsMenu(Menu menu) {
 3       MenuInflater inflater = getMenuInflater();
 4       inflater.inflate(R.menu.menu_map, menu);
 5       return true;
 6 }
 7
 8 @Override
 9 public boolean onOptionsItemSelected(MenuItem item) {
10       switch (item.getItemId()) {
11       case R.id.menu_close:
12                Log.d(TAG, "Salir");
13                int pid = android.os.Process.myPid();
14                android.os.Process.killProcess(pid);
15                return true;
16       case R.id.menu_ar:
17                ...
18                return true;
19       case R.id.menu_find:
20                ...
21                return true;
22
23       ...
24       default:
25                return super.onOptionsItemSelected(item);
26       }
27 }
```

En el evento `onCreateOptionsMenu` deberemos "inflar" el menú de forma parecida a cómo ya hemos hecho anteriormente con el tipo layout. Primero obtendremos una referencia al `inflater` mediante el método `getMenuInflater` y posteriormente generaremos la estructura del menú llamando a su método `inflate` pasándole como parámetro el ID del menú definido en XML. Por último devolveremos el valor `true` para confirmar que debe mostrarse el menú.

La implementación de cada una de las opciones se incluirá en el evento `onOptionsItemSelected` de la actividad que mostrará el menú. Este evento recibirá como parámetro el ítem de menú que ha sido seleccionado, cuyo ID recuperamos con el método `getItemId`. Según este ID podremos saber qué opción ha sido pulsada y ejecutar una acción u otra.

Se observa en las líneas 11-15, por ejemplo, que si el usuario selecciona la opción Salir de la aplicación, se ejecutará una rutina estándar de finalización de aplicaciones en Android.

7.2 - UBICANDO AL CLIENTE

El prototipo utiliza la señal GPS para conocer la ubicación actual del cliente, pudiendo así tanto dibujarla en el mapa.

El SDK de Android incluye un soporte primario para prestar servicios de localización, que encontramos en el paquete *android.location*.

Este paquete contiene varias clases relacionadas con los servicios de localización y, además, incluye un servicio, *LocationManager*, el cual proporciona una *API* para determinar la localización del dispositivo. Las clases básicas para obtener la señal de GPS son *LocationManager* para controlar el dispositivo, *LocationListener* para escuchar sus cambios y *Location* para guardar la información de localización.

LocationManager no puede ser instanciado directamente sino que debemos obtener un controlador para este. Entonces, para acceder a este o a cualquier otro servicio integrado en el dispositivo móvil, se utiliza el método `getSystemService` de la clase *Activity* y sus clases derivadas.

```
LocationManager locationManager =
    (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

Además, necesitamos otorgar a la aplicación el permiso de aplicación para acceder al dispositivo de localización mediante la declaración en AndroidManifest.xml de la etiqueta correspondiente:

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

Cuando se obtiene el control de un *LocationManager*, podemos realizar las siguientes acciones:

- Lanzar consultas a los *LocationProviders* disponibles para conocer la última posición conocida del dispositivo.
- Registrar un componente para que reciba actualizaciones periódicas de la ubicación desde un *LocationProvider*.

Antes de poder conseguir la posición actual a través de `locationManager`, Android necesita que definamos una clase que permita gestionar los eventos asociados al GPS, como pueden ser: la posición cambie (exista un desplazamiento del usuario), el dispositivo de localización se active o desactive.

Para ello, es necesario que se defina una clase que implementa la interfaz *LocationListener*. En nuestro caso esta clase será una implementación de nuestra propia `LocationListenerGPS`.

LocationListenerGPS.java

```
 1 public class LocationListenerGPS implements LocationListener {
 4       protected Location lastKnowLocation;
 5       protected Context context;
 6       private Activity activity;
 7
 8       public LocationListenerGPS(Context context, Activity activity, Location
lastKnowLocation){
 9               this.context = context;
10               this.activity = activity;
11               this.lastKnowLocation = lastKnowLocation;
13       }
14
15       @Override
16       public void onLocationChanged(Location location) {
17               Log.d(TAG, "Posición recibida de GPS");
18               lastKnowLocation = location;
19               this.showAddress();
20       }
21
22       ...
37
38       protected void showAddress() {
39               if(lastKnowLocation == null)
40                       return;
41
```

```
42                // obtener ubicación
43                String addressName = "";
44                try{
45                        Geocoder geocoder = new Geocoder(this.getContext(),
Locale.getDefault());
46                        List<Address> lsAddress =
geocoder.getFromLocation(lastKnowLocation.getLatitude(),
lastKnowLocation.getLongitude(), 1);
47                        if(lsAddress.size() > 0){
48                                // imprimir ubicación
49                                ...
59                        }
60                }
61                ...
69        }
70 }
```

Vemos que la interfaz nos hace método `onLocationChanged`, donde recibimos la posición que nos envía el dispositivo GPS. Aquí por ahora estamos implementando una funcionalidad que imprime, utilizado la librería `Geocoder` de Google, la dirección donde se encuentra el usuario.

La interfaz completa del listener, implementa una serie de métodos asociados a los distintos eventos que podemos recibir del proveedor, y se completa con:

- `onProviderDisabled(provider)`: Lanzado cuando el proveedor se deshabilita.
- `onProviderEnabled(provider)`: Lanzado cuando el proveedor se habilita.
- `onStatusChanged(provider, status, extras)`: Lanzado cada vez que el proveedor cambia su estado.

Con esto ya es posible acceder a la localización del cliente utilizando el método `onLastKnownLocation()` del objeto `LocationManager`, que es el que tiene el control del GPS. La localización vendrá dada en un objeto `Location`, y de ella se puede obtener, entre otras cosas, la longitud y latitud con los métodos `getLongitude()` y `getAltitude()`, respectivamente.

7.2.1 - Mostrando la ubicación del usuario

Android no ofrece a través de la clase `MyLocationOverlay` una vista superpuesta que dibuja la actual posición GPS sobre el un mapa. Utilizando este nos olvidamos de gestionar objetos como *LocationManager* y *Overlays* para mostrar la ubicación del cliente. Sino que basta con definir el objeto y asociarlo al mapa.

Visualmente nos muestra un área azul transparente sobre el mapa, que se actualiza automáticamente de acuerdo a la posición que recibe de algún proveedor de localización y la circunferencia de la forma denota la precisión que se tiene sobre el proveedor actual.

Entonces añadiremos un *Overlay* en el mapa de nuestra actividad principal que muestre la posición actual del usuario:

MainMap.java

```
 1 public class MainMap extends MapActivity {
 2       private MapView mapView;
 3
 4       @Override
 5       public void onCreate(Bundle savedInstanceState) {
 6               super.onCreate(savedInstanceState);
 7               ...
 8               this.initMap();
 9       }
10
11       private void initMap() {
12               Log.d(TAG, "Inicializando localización");
13               myLocationOverlay = new MyCustomLocationOverlay(this, mapView);
14               mapView.getOverlays().add(myLocationOverlay);
15               myLocationOverlay.enableCompass();
16               myLocationOverlay.enableMyLocation();
```

```
17                myLocationOverlay.runOnFirstFix(new
MyLocationOverlayFirstRun(mapView,
18                        myLocationOverlay));
19        }
20  }
```

Se muestra en el código como se inicializa un `MyCustomLocationOverlay`. Se crea el *Overlay* de posición (línea 13), se habilita el compás para el mapa (línea 15) y la impresión de la localización (línea 16) y por último se indica que la primera actualización se haga a través de un *thread*, el `MyLocationOverlayFirstRun` para no demorar la carga del mapa (línea 17).

Pero notamos que el nombre de la clase nos hace suponer que esta una implementación personal la que utilizamos. Esto es, porque además de mostrar la ubicación actual del cliente, también queríamos mostrar la orientación del mismo, funcionalidad no soportada actualmente por el objeto incluido en el API.

MyCustomLocationOverley.java

```
 1 public class MyCustomLocationOverlay extends MyLocationOverlay {
 2       private Context mContext;
 3       private float mOrientation;
 4
 5       public MyCustomLocationOverlay(Context context, MapView mapView) {
 6               super(context, mapView);
 7               mContext = context;
 8       }
 9
10       @Override
11       protected void drawMyLocation(Canvas canvas, MapView mapView, Location
lastFix, GeoPoint myLocation, long when) {
12               Point screenPts = mapView.getProjection().toPixels(myLocation, null);
13               Bitmap arrowBitmap =
BitmapFactory.decodeResource(mContext.getResources(), R.drawable.arrow);
14               Matrix matrix = new Matrix();
15               matrix.postRotate(this.getOrientation());
16               Bitmap rotatedBmp = Bitmap.createBitmap(arrowBitmap, 0, 0,
arrowBitmap.getWidth(), arrowBitmap.getHeight(), matrix, true);
17               canvas.drawBitmap(rotatedBmp, screenPts.x - (rotatedBmp.getWidth() /
2), screenPts.y - (rotatedBmp.getHeight() / 2), null);
18               mapView.postInvalidate();
19       }
20 }
```

Esta implementación es una subclase de `MyLocationOverlay`, que básicamente sobrescribe el método `drawMyLocation()`:

- Primeramente obtiene la posición (en píxeles) de pantalla de la ubicación actual de usuario sobre el mapa (linea12).
- En la línea 13, creamos un *Bitmap*(*) que será el que se muestre como nuestra posición.
- Obtenemos la orientación actual del dispositivo cliente, obtenemos la orientación actual y creamos un objeto `Matrix` con esa orientación (línea 14 y 15).
- Luego, creamos un nuevo *Bitmap* con la imagen original, y con la rotación adecuada.
- Por último, en la línea 18 y a través del método `postInvalidate()`, redibujamos el mapa para reflejar los cambios.

7.3 - COMUNICACIONES CON EL SERVIDOR

En el capitulo anterior ya expusimos como el servidor despliega los datos que serán consumidos por los clientes. Entonces, dijimos que esa información viajaba a través de la red en formato JSON sobre el protocolo REST y explicamos este último.

Vale aclarar que la API de Maps nos hace transparente cualquier gestión de conexión al servidor que necesite para visualizar los mapas. Y si bien existe en las últimas versiones de la aplicación *Google Maps* para móviles la posibilidad de visualizar mapas sin conexión, esto es exclusivo de la propia aplicación y no esta soportado en la API.

Previamente a todo diseño, tenemos que decir que para que la aplicación tenga acceso a Internet, esto se hace en AndroidManifest.xml:

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

De esta manera, nuestra aplicación podrá acceder a Internet, ya sea por WIFI o 3G.

Android cuenta con muchas clases dedicados en exclusiva a las diferentes cuestiones de comunicaciones. Por ejemplo, de Java hereda, entre otros, *javax.net*. Dentro de sus paquetes exclusivos como plataforma, incluye una amplísima familia de paquetes bajo el nombre raíz *android.net.** y sobre todo *org.apache.http.**, de la que se hará uso en este prototipo.

Utilizamos un cliente HTTP estándar, con la clase `DefaultHttpClient`. Ésta cuenta con el método `execute()`, que precisa como parámetros la petición a enviar y se encarga de ejecutar la conexión. Para recibir la correspondiente respuesta, se define un objeto `HttpResponse` que recibe el resultado de `execute()`.

Para finalizar, la respuesta desde el servidor se procesa mediante clases JSON y para poder ser finalmente convertidas al modelo local. La implementación que utilizamos para JSON es *Gson* [50], una librería de Google para serializar y deserializar objetos JSON.

Desde la aplicación móvil, implementamos la interfaz de un cliente REST, que será el encargado de establecer la conexión con el servidor de AR-DROID e intercambiar datos.

RESTClient.java

```
 1 public class RESTClient {
 2
 3       public static void doPut(final JsonObject data, final String url) {
 4               HttpClient httpClient = new DefaultHttpClient();
 5               HttpConnectionParams.setConnectionTimeout(httpClient.getParams(),
10000);
 6               Thread thread = new Thread() {
 7                       public void run() {
 8                               try {
 9                                       HttpPut httpPut = new HttpPut(url);
10                                       httpPut.addHeader("Accept",
"application/json");
11                                       httpPut.addHeader("Content-Type",
"application/json");
12                                       StringEntity entity = new
StringEntity(data.toString(), "UTF-8");
13                                       entity.setContentType("application/json");
14                                       httpPut.setEntity(entity);
15                                       httpClient.execute(httpPut);
16                               } catch (Exception e) {
17                                       Log.e("Error", "doPut", e);
18                                       e.printStackTrace();
19                               }
20                       }
21               };
22               thread.start();
23       }
24
25       public static InputStream doGet(String url) {
26               HttpClient httpClient = new DefaultHttpClient();
27               HttpConnectionParams.setConnectionTimeout(httpClient.getParams(),
10000);
28               HttpGet httpGet;
```

```
29              try {
30                      httpGet = new HttpGet(new URI(url.toString()));
31                      HttpResponse response;
32                      response = httpClient.execute(httpget);
33                      HttpEntity entity = response.getEntity();
34                      if (entity != null) {
35                              InputStream instream = entity.getContent();
36                              return instream;
37                      }
38              } catch (Exception e) {
39                      Log.e("Error", "doGet", e);
40                      e.printStackTrace();
41              }
42              return null;
43      }
44 }
```

Resumimos el comportamiento de este código. Primeramente vemos que la clase implementa dos métodos, `doPut` y `doGet`, ya que necesitaremos tanto recibir, como también enviar datos hacia el servidor.

El comportamiento del método `doGet` es:

- Creamos un cliente HTTP de la interfaz `HttpClient`, implementado por `DefaultHttpClient` (línea 26).
- Creamos una petición HTTP de tipo GET, mediante `HttpGet` (línea 30) para la URL recibida como parámetro.
- Con `HttpResponse`, recibimos el resultado de ejecutar la petición creada a través del cliente HTTP (línea 32).
- Si el resultado de la petición no es nulo, devolvemos el contenido en un `InputStream` (línea 35).

El método `doPut` es similar al descrito, salvo que la comunicación es en el otro sentido. En esta petición remitimos los datos (la entidad) al servidor. Veremos para que lo vayamos a utilizar más adelante.

Esto es todo lo que necesitamos implementar a "bajo nivel". Vemos que la SDK nos provee el objeto `HttpClient` que nos simplifica las cuestiones de conexión y sockets, y los objetos `HttpPut` y `HttpGet` que implementan peticiones HTTP.

7.3.1 - Representación del modelo en el cliente

Como describimos en el capitulo de especificación del prototipo, tenemos un modelo de datos que incluye las entidades que forman el sistema. Este modelo esta representado en ambas partes de la aplicación, tanto en la implantación del servidor, como en la del cliente.

En general, estas implementaciones son parecidas, aunque difieren en particularidades propias de cada plataforma. En el administrador Web la implementación se hace en Groovy, por lo que no es compatible con la del cliente, en Java nativo.

A continuación veremos como se intercambian datos entre la aplicación cliente y el servidor, pero antes tenemos que entender que existen dos implementaciones distintas para el modelo de datos.

Entonces, poniendo como ejemplo un evento para una entidad, este en primera instancia será creado y persistido en la base de datos. Cuando desde el cliente se solicita este evento para ser visualizado, se crea un objeto *Event* (de la implementación del servidor) con sus propiedades leídas desde la base de datos. Posteriormente este objeto evento es trasformado a JSON por medio de una estrategia definida, para ser enviado al cliente a través de una conexión de red. Por último, el cliente decodificara la información recibida a través de una estrategia de implementación propia, para mostrar el evento en el dispositivo móvil.

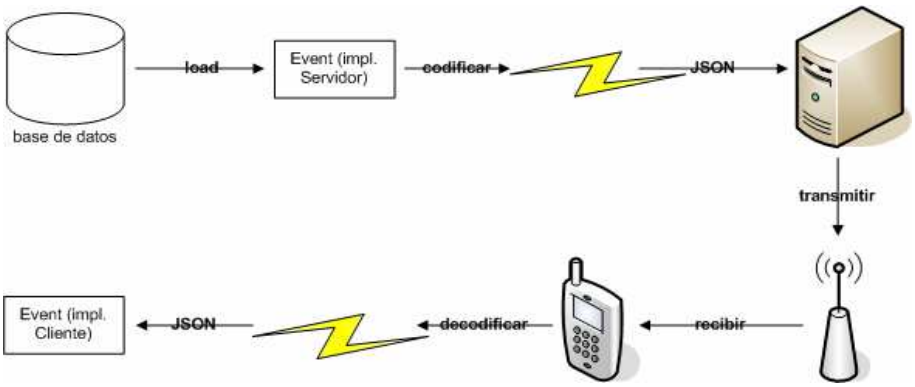


Figura 23. Diagrama de intercambio de información AR-DROID

Los datos se reciben desde el servidor y son deserializados por Gson, a través de sus constructores y deserealizadores. Veremos como trabaja esta librería y de que manera convierta la respuesta en objetos del modelo local a continuación.

7.3.2 - Recuperando las entidades y eventos

En el capitulo anterior describimos como el administrador Web expone la funcionalidad para brindar la informaciones de entidades y eventos a los cliente. Ahora veremos como el cliente recupera esa información. Describiremos también como se implementan algunas de las implementaciones más importantes que utilizan esos datos, a saber, representar entidades y eventos, y visualizar las novedades para una entidad.

Ya dijimos que el intercambio de datos se hace por medio de JSON, en la forma de comunicación REST. Ahora veremos como se realiza este intercambio técnicamente, desde el lado del cliente.

IResourceHelper.java

```
1 public interface IResourceHelper {
2       public List<Entity> getEntities();
3       public List<Event> getEvents(Entity entity);
4       public List<Event> getEvents();
5       public List<Event> getEvents(String option);
6       public void saveResponse(SurveyResponse surveyResponse);
7 }
```

Primeramente definimos una interfaz para obtener las entidades y eventos, esto es, porque el prototipo se comunicará como ya describimos al servidor. Pero quizás el agregado de una fuente de datos, o el almacenamiento local de la información, requieran una nueva implementación de esta interfaz que difiera de la que describiremos a continuación.

Por ejemplo, una posible implementación sería que al descargar datos de entidades y eventos estas se guardasen en una base de datos del dispositivo. Si la próxima ejecución de la aplicación no dispone de una conexión de red, podrían utilizarse los datos almacenados para brindarle la información al usuario.

A continuación definiremos una clase que nos ayudará a en cuestiones básicas de trabajo con JSON que, entre otras cosas, necesita alguna clase de conversión entre textos y strems, para adaptar nuestros resultados a las librerías que nos brinda la plataforma.

ResourceHelper.java

```
 1 public abstract class ResourceHelper implements IResourceHelper {
 2
 3       protected Reader loadManySerialized(String url) {
 4
 5               InputStream instream = RESTClient.doGet(url);
 6               if (instream != null) {
 7                       return new InputStreamReader(instream);
 8               }
 9               return null;
10       }
11 }
```

Con esto ya estamos en condiciones de implementar la funcionalidad para convertir la información que nos da el servidor a la implementación del modelo de datos local al cliente. Esta clase será la encargada de proveer a los servicios y actividades de nuestra aplicación móvil de los elementos a visualizar.

ResourceHelperJSON.java

```
 1 public class ResourceHelperJSON extends ResourceHelper {
 2
 3       public List<Entity> getEntities() {
 4               String urlServer =
ARDROIDProperties.getInstance().getProperty("ar.droid.server");
 5               Reader inputStream = loadManySerialized(urlServer +
Request.GET_ENTITIES);
 6               Type listType = new TypeToken<ArrayList<Entity>>() {
 7               }.getType();
 8               GsonBuilder gsonBuilder = new GsonBuilder();
 9               gsonBuilder.registerTypeAdapter(GeoPoint.class, new
GeoPointDeserializer());
10               gsonBuilder.registerTypeAdapter(ReaderNews.class, new
ReaderNewsDeserializer());
11               Gson gson = gsonBuilder.create();
12
13               List<Entity> xLsResult = gson.fromJson(inputStream, listType);
14               return xLsResult;
15       }
16
17       @Override
18       public List<Event> getEvents(Entity entity) {
19               String urlServer =
ARDROIDProperties.getInstance().getProperty("ar.droid.server");
20               Reader inputStream = loadManySerialized(urlServer +
Request.GET_EVENTS + "/" + entity.getId());
21               Type listType = new TypeToken<ArrayList<Event>>() {
22               }.getType();
23
24               GsonBuilder gsonBuilder = new GsonBuilder();
25               ...
28               Gson gson = gsonBuilder.create();
29
30               List<Event> events = gson.fromJson(inputStream, listType);
31
32               Iterator<Event> it = events.iterator();
33
34               while (it.hasNext()) {
35                       Event event = (Event) it.next();
36                       event.setEntity(entity);
37               }
38               return events;
39       }
40
41       public void saveResponse(SurveyResponse surveyResponse) {
42               ...
45               Gson gson = gsonBuilder.create();
46               RESTClient.doPut(gson.toJsonTree(surveyResponse).getAsJsonObject(),
urlServer + "/request/feedback");
47
48       }
49
50       @Override
51       public List<Event> getEvents() {
52               return getEvents("");
53       }
54
55       @Override
56       public List<Event> getEvents(String option) {
57               ...
81       }
82 }
```

Describamos a nivel más comprensivo el código anterior: Esta es una implementación del la

interfaz que comentábamos, que obtiene los eventos desde el servidor y a través de JSON.

Tomemos como ejemplo el método `getEntities`, que lo que haces es devolvernos un listado de objetos `Entity`, quede representa a todas las entidades registradas en el sistema. Vamos línea a línea:

- Primeramente guardamos en `urlServer` desde la parametrización de la aplicación la URL al servidor (línea 4).
- Creamos un lector de stream que contendrá los bytes devueltos por la petición al servidor en `inputStream` (línea 5).
- En la línea 8, una instancia de `GsonBuilder` será la encargada de crear instancias `Gson` (línea 11) con algunas configuraciones particulares.
- Tenemos varios algoritmos deserializadores, que actúan como estrategias de conversión al encontrar objetos de determinados tipos. Ejemplo de estos son las clases `GeoPointDeserializer` y `ReaderNewsDeserializer`.

Conozcamos un poco más acerca de estos desearilizadores. Si observamos en detalle el código, vemos que se recibe el objeto que contiene la respuesta JSON, y se trata como tal, obteniendo los atributos de un objeto con `getAsJsonObject().get("nombre_atributo")`.

Esto hace que la conversión sea automática, con relación 1 a 1, lo que nos permite convertir entre las dos implementación del modelo de datos sin más que estos adaptadores.

GeoPointDeserializer.java

```
 1 public class GeoPointDeserializer implements JsonDeserializer<GeoPoint> {
 2
 3       @Override
 4       public GeoPoint deserialize(JsonElement json, Type type,
JsonDeserializationContext context)
 5                       throws JsonParseException {
 6               ...
 9               double lat = json.getAsJsonObject().get("latitude").getAsDouble();
10               double lng = json.getAsJsonObject().get("longitude").getAsDouble();
11               double alt = 0;
12               if (!json.getAsJsonObject().get("altitude").isJsonNull())
13                       alt = json.getAsJsonObject().get("altitude").getAsDouble();
14               GeoPoint p = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6), alt);
15               return p;
16       }
17 }
```

En este caso necesitamos convertir los `GeoPoint` (ubicación con latitud, longitud y altitud) ya que los valores de estas difieren en las *APi JavaScript* y *Google API* para Android de los mapas de Google.

Mientras que en la primera se trabaja con una representación de grados (de 0 a 90 para la latitud y de -180 a 180 para lo longitud), el sistema para representar coordenadas en Android se especifica en microgrados (de 0 a 90.000.000 para la latitud). Por lo que la conversión se hace multiplicando o dividiendo por la constante `1E6`.

Así mismo, la clase `GeoPoint` expuesta no es la que provee la API de mapas de Android, sino que es una extensión que implementamos, puedo que esta no proveía la propiedad `altitude`, que necesitaremos más adelante para la implementación de Realidad aumentada.

7.4 - VISUALIZANDO ENTIDADES Y EVENTOS

Ya describimos que el cliente interactúa con el servidor y solicita las entidades y eventos. Una vez obtenidas, estas se deben plasmar visualmente en el mapa para poder recorrerlas y manipularlas.

Hemos decidido que las entidades se muestren diferenciando su tipo, para una lectura más rápida y colorida. Por esto, cada entidad se representa con el icono de su tipo de entidad, definido en el administrador Web.

La clase `Overlay` y sus derivadas están disponibles en el API de Maps para hacer realmente sencillo el uso de superposiciones graficas en los mapas. Con estas estructuras no tenemos que preocuparnos de manejar el *canvas*(*) del mapa, redimensionamiento de marcadores, movilidad de los mismos para mantener su ubicación exacta, etc. De todo este comportamiento se encarga el API.

Estas no sólo dan acceso a dibujar elementos asociados a un determinado mapa, sino que también permite controlar los eventos relacionados con ellos como, por ejemplo, que un marcador sea pulsado por el usuario.

MainMap.java

```
1  public class MainMap extends MapActivity implements IDirectionsListener {
2
3       @Override
4       public void onCreate(Bundle savedInstanceState) {
5                ...
6                drawEntities();
7                ...
8       }
9
10      private void showEntities() {
11               // Se recupera las entidades a mostrar en el mapa
12               List<Entity> entities = Resource.getInstance().getEntities();
13               showEntities(entities);
14      }
15
16      private void drawEntities(final List<Entity> entities) {
17               mapView.getOverlays().clear();
18               initMap();
19
20               Runnable viewOrders = new Runnable() {
21                        public void run() {
22                                 // se agrupan entidades por tipo de entidad
23                                 Iterator<Entity> itEnt = entities.iterator();
24                                 Map<TypeEntity, List<Entity>> types = new
HashMap<TypeEntity, List<Entity>>();
25                                 while (itEnt.hasNext()) {
26                                          Entity entity = (Entity) itEnt.next();
27                                          List<Entity> listEntities = new
ArrayList<Entity>();
28                                          if (types.get(entity.getTypeEntity()) !=
null) {
29                                                   listEntities =
types.get(entity.getTypeEntity());
30                                          }
31                                          listEntities.add(entity);
32                                          types.put(entity.getTypeEntity(),
listEntities);
33                                 }
34
35                                 // se crean los overlays con la info de las
entidades
36                                 Iterator<TypeEntity> itTypesEnt =
types.keySet().iterator();
37                                 while (itTypesEnt.hasNext()) {
38                                          TypeEntity typeEntity = itTypesEnt.next();
39                                          itEnt = types.get(typeEntity).iterator();
40
41                                          // se recupera el icono a mostrar para el
tipo de entidad
42                                          Drawable iconTypeEntity =
ImageHelperFactory.createImageHelper().getIconTypeEntity(typeEntity);
```

```
43                                          iconTypeEntity = scaleImage(iconTypeEntity,
2);
44                                          ...
49                                          MapEntityItemizedOverlay mapEntityOverlay =
new MapEntityItemizedOverlay(iconTypeEntity, mapView,
50                                                          MainMap.this);
51                                          while (itEnt.hasNext()) {
52                                                  Entity entity = (Entity)
itEnt.next();
53                                                  EntityOverlayItem overlayitemEntity
= new EntityOverlayItem(entity.getGeoPoint(),
54      entity.getName(), entity.getDescription(), entity);
55      mapEntityOverlay.addOverlay(overlayitemEntity);
56
57                                          }
58                                          ...
63                      }
64              };
65              Thread thread = new Thread(null, viewOrders, "agentcargaentidades");
66              thread.start();
67      }
68 }
```

Vemos que al crear la actividad, mostramos las entidades a través de la llamada a `showEntities()`.

Describimos a grandes rasgos esta funcionalidad, ya que es fácil de entender:

- Las líneas 10-14 definen el método `showEntities()`, este obtiene el listado de entidades, y hace una llamada a `drawEntities(entities)`, que se encarga de dibujar estas entidades.
- `drawEntities` crea un hilo (`viewOrders`) para no interrumpir con la carga y desplazamiento de mapa, y así no generar congelamientos de pantalla.
- Cuando empieza la ejecución del hilo (línea 26) primeramente agrupamos las entidades por tipo en un mapa.
- A continuación recorremos este mapa en orden y empezamos a graficar las entidades (línea 36). Para cada grupo de tipos, se crea el icono que representará a las entidades en `iconTypeEntity`. Para posteriormente escalarla a la resolución del dispositivo a través de `scaleImage()`.
- Para cada entidad de se crea un objeto `EntityOverlayItem`, con la posición, nombre, descripción y una instancia de la entidad (línea 56).
- Por último, se agrega el *Overlay* al mapa para dibujarlo.

A través de `EntityOverlayItem` subclasificamos `OverlayItem`, clase que representa un elemento añadido al mapa. Este elemento puede proporcionar a sus marcadores propiedades propias, o utilizar los valores predeterminados para la instancia de superposición.

Algo que nos parece importante mencionar, es que después de crear un objeto `MapOverlay` (`mapEntityOverlay`) éste se asocia al mapa representado por nuestra instancia de `MapOverlay`, utilizando los métodos `getOverlays().add`. Se puede intuir que un mismo mapa permite tener varios objetos `Overlay` asociados, de forma que se pueden pintar elementos de forma independiente a través de diferentes clases que se implementen.

En nuestro caso, el mapa mostrado por nuestro prototipo solamente tendrá un `Overlay` asociado, encargado de dibujar las entidades y eventos.

También mostramos en el fragmento de código anterior, que para cada instancia de `EntityOverlayItem` creamos un `MapEntityItemizedOverley` asociado que es una subclasificación de `BalloonItemizedOverley`. Esta última es un View capaz de dibujar una ventana emergente y responder el evento `onClick`. Lo utilizamos como punto de acceso a la información extendida de la entidad.

Esta ventana primeramente muestra el nombre y descripción resumida de la entidad, con un evento para responder al clic sobre el.

MapEntityItemizedOverley.java

```
 1 public class MapEntityItemizedOverlay extends
BalloonItemizedOverlay<EntityOverlayItem> {
 2
 3       private List<OverlayItem> mOverlays = new ArrayList<OverlayItem>();
 4       private Context mContext;
 5       private MapView mapView;
 6       private Activity activity;
 7
 8       public MapEntityItemizedOverlay(Drawable defaultMarker, MapView mapView,
Activity activity) {
 9               super(boundCenterBottom(defaultMarker), mapView);
10               mContext = mapView.getContext();
11               this.mapView = mapView;
12               this.activity = activity;
13               populate();
14       }
15
16       ...
17
18       @Override
19       protected boolean onBalloonTap(int index, EntityOverlayItem item) {
20               hideBalloon();
21               Intent i = new Intent(activity.getApplicationContext(),
EntityTabWidget.class);
22               i.putExtra("idEntity", item.getEntity().getId());
23               activity.startActivityForResult(i, 0);
24               return true;
25       }
26 }
```

Vemos que en el constructor de la ventana emergente pasamos todo el contexto de la aplicación. Necesitamos el `MapView` y el `Activity` para poder soportar la funcionalidad de *Balloon (globo)* y poder instanciar la actividad de visualización de la entidad.

En las líneas 19-24 sobrescribimos el método `onBalloonTap`, para que al clickear sobre la ventana emergente, nos despliegue la actividad para visualizar la información de la entidad. La tarea la realizamos a través de un `Intent`, técnica utilizada a lo largo del desarrollo.

Mostraremos luego como se despliega la información completa de cada entidad al seleccionar una desde el mapa.

Eventos

Los eventos se muestran de forma similar. Para distinguirlos de las entidades hemos elegido iconos circulares en colores, representando cada color un tipo de evento (nombramos aquí que los tipos de eventos son parametrizables a través del administrador Web).

En la vista mapa, por defecto la aplicación muestra las entidades cercanas a la ubicación del usuario. Para visualizar los eventos en el radio del usuario, debemos ir a la opción "Eventos" del menú de la aplicación.

Esta acción nos desplegará el dialogo de opciones mostrado debajo. Las opciones para visualizar eventos están identificadas para filtrar la cantidad de eventos a dibujar sobre el mapa:

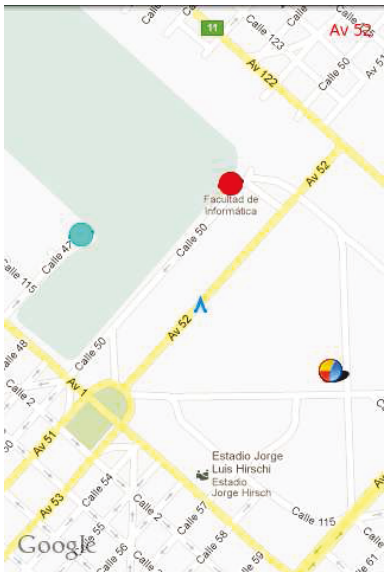
- Hoy: muestra los eventos para hoy.
- 7 días: muestra los eventos para los próximos 7 días.
- 30 días: dibuja los eventos para el próximo mes.

Este filtro funciona simplemente enviando la petición al servidor de los eventos que se quieren mostrar. La resolución de estos se hace enteramente del lado del servidor.

El servidor recibe la petición y recorre la lista de eventos definidos para todas las entidades, e interroga a la periodicidad del mismo si corresponde a la opción seleccionada por el usuario. A Modo de ejemplo, resumimos la siguiente situación: Si el usuario escoge filtrar los eventos para hoy, se mostrarán:

- Todos los eventos Únicos que correspondan a la fecha actual.
- Todos los eventos con periodicidad Diaria.
- Todos los eventos con periodicidad Semanal donde el día de la semana parametrizado sea el día hoy que corresponde.
- Todos los eventos con periodicidad Mensual donde el día del mes parametrizado sea el día de la fecha actual.

Una vez seleccionado el rango de eventos a mostrar se desplegarán en el mapa los eventos que coincidan con el rango seleccionado.



Notamos en la imagen anterior dos iconos de distintos colores y un icono multicolor, esto representa los distintos tipos de eventos, tal como se parametriza en el servidor. El icono multicolor define más de un evento para esa ubicación.

Hicimos esta distinción porque es probable que en una entidad se definan más de una evento un mismo día o semana. Presionando este último, se desplegará la lista de eventos para esa entidad.

7.4.1 - Información de las entidades

Representamos cada entidad con una actividad de tipo *pestañas*, para una mejor visualización y distribución de la información para cada uno.

Por la limitación de tamaño existente en los dispositivos móviles, cuando la cantidad de datos a proporcionarle al usuario puede ser mucha, es mejor hacerlo de forma fragmentada para evitar que el usuario deba desplazarse a lo largo de la pantalla para llegar al contenido de su interés.

Para crear una interfaz de usuario con pestañas, debemos utilizar un `TabHost` y `TabWidget`. El `TabHost` actúa como el nodo raíz de disposición, que contiene un `EntityabWidget` para la visualización de las pestañas como un `FrameLayaut` para mostrar el contenido de cada una de ellas.

Tenemos dos maneras de implementar el contenido de una pestaña: usando las pestañas para cambiar entre distintas *Views* dentro de la misma actividad, o utilizando las pestañas para cambiar entre actividades totalmente independientes. El método de actividades independiente encaja mejor en nuestra necesidad, ya que podemos administrar mejor varias actividades en grupos, en lugar de una gran actividad con un diseño complejo.

Entones, tenemos que la visualización de los datos de una entidad son implementados con tres distintas actividades. Las describiremos a continuación:

- `EntityActivity`: Muestra los datos básicos de la entidad, como nombre, imagen, descripción y pagina Web.
- `ListNewsActivity`: Muestra las novedades de la entidad. Describiremos esta activad mas adelante.
- `ListEventsActivity`: Muestra la lista de los eventos próximos definidos para la

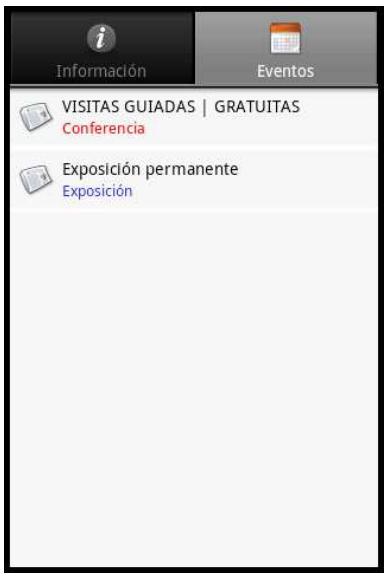
aplicación.

EntityTabWidget.java

```
 1 public class EntityTabWidget extends TabActivity implements IReaderListener {
 2
 3       @Override
 4       protected void onCreate(Bundle savedInstanceState) {
 5               super.onCreate(savedInstanceState);
 6               Entity entity =
Resource.getInstance().getEntity(getIntent().getExtras().getLong("idEntity"));
 7               setContentView(R.layout.tab_host);
 8               TabHost tabHost = getTabHost();
 9               Intent intent =  new Intent().setClass(this, EntityActivity.class);
10
        intent.putExtra("idEntity",getIntent().getExtras().getLong("idEntity"));
11
12               TabHost.TabSpec spec =
tabHost.newTabSpec(TAB_INFO).setIndicator(getResources().getString(R.string.ic_tab_ent
ity_main),
getResources().getDrawable(R.drawable.ic_tab_entity_main)).setContent(intent);
13               tabHost.addTab(spec);
14               entity.getReaderNews().reader(this);
15               ...
16               // tabs de eventos
17               tabHost.setCurrentTab(0);
18
19       }
20
21       @Override
22       public void doReaderTweeter() {
23               tabReaderNews(R.drawable.ic_tab_entity_news_twitter);
24       }
25
26       private void tabReaderNews(int ic_tab_selector){
27               ...
28       }
29 }
```

Primeramente obtenemos la entidad que vamos a visualizar a través de los parámetros adicionales a la llamada a la actividad (línea 6). Luego creamos un *Intent* para la actividad `EntityActivity`, y a través de la creación de una pestaña `tabHost.newTabSpec(TAB_INFO)` parametrizamos la actividad que contendrá y otros aspectos de control y visualización. A la pestaña recién creada en `spec`, le indicamos la actividad que contendrá a través de la llamada a `setContent(intent)`, pasándole el *Intent* que creamos para la actividad contenida.

De manera general, así es como creamos la actividad de pestañas. Luego de creada la pestaña para la información básica creamos la actividad que mostrará la lista de eventos, pero antes, vemos que en la línea 14 hacemos `entity.getReaderNews().reader(this)`. Esta es la forma de crear la pestaña para visualizar las novedades de la entidad. Lo que hacemos a través de un *double dispatching*(*) es decirle al tipo de lector de noticias que cree la pestaña para mostrar la lista. Dependiendo la implementación del lector, este le indicará a la actividad que tipo de pestaña crear. En el caso de Twitter, el lector hará una llamada a `doReaderTweeter` (líneas 22-24), donde la actividad instanciará la actividad para mostrar la lista de noticias. Este método también creara una pestaña con el selector que contiene los iconos representativos de Twitter.



En la segunda imagen no vemos la pestaña Noticias, pues la parametrización de la entidad no incluye un lector de noticias.

Vemos que el selector es configurable para mostrar los iconos de las pestañas inactivas en blanco y negro. Esto se hace a través de un selector definido en XML, donde se indica que icono utilizar para cada estado de la pestaña.

ic_tab_entity_main.xml

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/ic_info"
          android:state_selected="true" />
    <item android:drawable="@drawable/ic_info_gray" />
</selector>
```

La actividad `EntityActivity` define el esquema de los datos informativos de cada entidad. Describamos de manera textual como cumplimos con esta tarea: Primeramente definimos un *layout* para la actividad como lo hicimos anteriormente, a través de un XML.

A continuación implementamos la actividad y en el `onCreate` le asignamos un valor a cada dato que queremos mostrar de la misma. Debajo se muestra el código para la actividad, con comentarios que hacen referencia a cada dato que se setea.

EntityActivity.java

```
 1 public class EntityActivity extends Activity {
 2       private Entity entity;
 3
 4       protected void onCreate(Bundle savedInstanceState) {
 5               super.onCreate(savedInstanceState);
 6               // setear layout
 7               setContentView(R.layout.view_entity);
 8               // recupero la entidad
 9               entity =
Resource.getInstance().getEntity(getIntent().getExtras().getLong("idEntity"));
10               TextView title = (TextView) this.findViewById(R.id.title);
11               title.setText(entity.getName());
12               // recuperar la imagen descripctiva de la entidad
13               Drawable photoEntity =
ImageHelperFactory.createImageHelper().getImageEntity(entity);
14               photoEntity.setBounds(0, 0, photoEntity.getIntrinsicWidth(),
photoEntity.getIntrinsicHeight());
15
16               ImageView image = (ImageView) this.findViewById(R.id.imageEntity);
17               image.setImageDrawable(photoEntity);
18
19               // setar URL
20               TextView url = (TextView) this.findViewById(R.id.site);
```

```
21                url.setText(Html.fromHtml("<a href=\'" + entity.getUrl() + "\'>" +
entity.getUrl() + "</a>"));
22                url.setMovementMethod(LinkMovementMethod.getInstance());
23
24                // mostrar descripción
25                TextView descr = (TextView) this.findViewById(R.id.description);
26                descr.setText(entity.getDescription());
27        }
28
29        ...
30 }
```

7.4.2 - Información de los eventos

La ventana de información para los eventos esta implementada de igual manera que implementamos la visualización de las entidades, por lo que no describiremos el proceso.

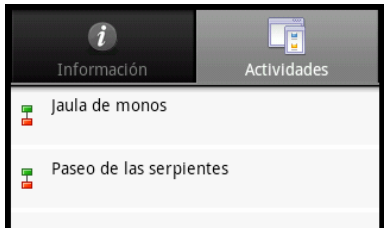
Si vamos a nombrar la particularidad con la que cuenta la actividad `EventActivity`, que es la de contar con dos botones de acción que después de mostrar los datos informativos del evento. Estos botones son: "Comentar" y "Compartir".

El primero, nos dará la posibilidad de comentar el evento, que dependerá del de tipo de feedback parametrizado para el vento. El segundo, nos permitirá compartir el evento con otras personas. Describiremos estas funcionalidades en los capítulos siguientes.

7.4.3 - Listar las actividades de un evento

Cada evento puede definir una lista de actividades a desarrollar. Esto es útil en eventos de tipo conferencia, por ejemplo, donde se define un cronograma de actividades a desarrollarse en una jornada.

La lista de actividades se muestra en la solapa Actividades de cada evento, y al seleccionar alguna, se mostrará la descripción de la misma en un popup.



7.4.4 - Filtrar los resultados

Cada vista del Mapa, ya sea que se muestren entidades o eventos, tiene la posibilidad de filtrar los resultados dependiendo de su tipo.

Estando en la vista de entidades, podemos visualizar solo un tipo específico de entidad mediante el menú *Tipos de Entidad*. Mostramos como creamos este dialogo para seleccionar el tipo de entidad a visualizar.

MainMap.java

```
 1 ...
 2 private void showOptionsTypeEntities() {
 3       AlertDialog.Builder builder = new AlertDialog.Builder(this);
 4       builder.setTitle("Capas - Entidades");
 5       List<TypeEntity> typeEntity = Resource.getInstance().getTypeEntities();
 6       final ArrayAdapter<TypeEntity> itemlist = new TypeEntityAdapter(this,
typeEntity);
 7       builder.setSingleChoiceItems(itemlist, -1, new
DialogInterface.OnClickListener() {
 8               @Override
 9               public void onClick(DialogInterface dialog, final int which) {
10                       TypeEntity typeEntity = itemlist.getItem(which);
11                       dialog.dismiss();
12                       List<Entity> entityFilter = new ArrayList<Entity>();
13                       for (Entity entity : Resource.getInstance().getEntities()) {
14                               if (entity.getTypeEntity().equals(typeEntity)) {
15                                       entityFilter.add(entity);
16                               }
17                       }
18                       showEntities(entityFilter);
19               }
20       });
21       AlertDialog alert = builder.create();
22       alert.getListView().setBackgroundColor(Color.WHITE);
23       alert.show();
24 }
25 ...
```

El fragmento de código anterior muestra la acción que se ejecuta al presionar la opción de para filtrar los tipos de entidades. Función similar sucede al filtrar tipos de eventos.

Esta funcionalidad de *Capas* es muy común en este tipo de aplicaciones, la mayoría de las aplicaciones reconocidas que hemos analizado la implementan de alguna manera. Le da valor a la aplicación, ya que le permite al usuarios seleccionar solo la información que le interesa y descartar aquella que no.

El resultado de la implementación se visualiza en las siguientes capturas del prototipo. La acción despliega una lista para seleccionar que tipo de entidad (o de evento, según corresponda) queremos visualizar. Vale aclarar que estas listas se arman con los tipos que son parametrizados desde el Administrador Web.

7.4.5 - Noticias de las entidades

Ya describimos como se visualizan entidades y eventos. Como comentamos a lo largo de la especificación del sistema, cada una de las entidades del sistema tiene definido un "alimentador" de noticias y novedades. Dijimos entonces, que una entidad puede consumir automáticamente noticias desde Twitter, Facebook, RSS, o desde una página Web.

También describimos en el capitulo anterior como se parametrizaban estos alimentadores para poder consumirlos desde el cliente. Veamos ahora como están definidos e implementados cada uno.

Primeramente decimos que el modelo de clases del cliente es similar al del servidor, por lo que tendremos una jerarquita de lectores de noticias, donde la base el la clase abstracta `ReaderNews`.

ReaderNews.java

```
1  public abstract class ReaderNews {
2        private String parameter;
3
4        public String getParameter() {
5                return parameter;
6        }
7        public void setParameter(String parameter) {
8                this.parameter = parameter;
9        }
10
11       public abstract List<Message> getMessages();
12       public abstract String getName();
13       public abstract void reader(IReaderListener reader);
14 }
```

Todos los lectores de noticias deberían extender esta clase para que su implementación se acople al modelo actual del prototipo. En esta clase tenemos tres métodos que sus subclases deben implementar:

- `getMessages`: será el encargado de obtener las noticias desde el proveedor que implementa.
- `getName`: texto que representa el nombre del proveedor.
- `reader`: este método será el encargado de indicarle a la UI cual es el lector que implementar la entidad, para que se armen las vistas correspondientes.

El método recibe un `IReaderListener` como parámetro. Esta es una interfaz que debe implementar la vista, para definir el comportamiento al recibir el tipo de lector de la entidad. En el prototipo, el componente de la UI `EntityTabWidget` (que veremos más adelante) es el que implementan esta interfaz.

Solo mostraremos en detalle la implementación de uno de los proveedores de noticias. El resto lo

describiremos brevemente diferenciándolos del descripto a continuación: el lector de noticias desde Twitter.

Comentamos ahora que la actividad de visualización de entidades es la que, ya conocida la entidad a visualizar, pide los mensajes al lector de noticias y le dice a la interfaz que tipo de lector va a mostrar.

ReaderNewsTwitter.java

```
 1 public class ReaderNewsTwitter extends ReaderNews {
 2
 3       @Override
 4       public List<Message> getMessages() {
 5               return
ResourceNewsFactory.createResourceNewsTwitter().getMessages(getParameter());
 6       }
 7
 8       @Override
 9       public String getName() {
10               return "Twitter";
11       }
12
13       @Override
14       public void reader(IReaderListener reader) {
15               reader.doReaderTwitter();
16
17       }
18 }
```

Arriba se muestra la clase completa que realiza la alimentación de novedades mediante Twitter. Vemos al la implementación de `getMessages` utiliza `ResourceNewsFactory` para crear un objeto que será el encargado de leer y devolver las noticias desde Twitter para el usuario parametrizado. Vemos a continuación que es objeto es una instancia de la clase `TwitterREST`.

ResourceNewsFactory.java

```
 1 public class ResourceNewsFactory {
 2
 3       public static ResourceNews createResourceNewsRSS() {
 4               return new SAXParserRSS();
 5       }
 6
 7       public static ResourceNews createResourceNewsFacebook() {
 8               return new FacebookREST();
 9       }
10
11       public static ResourceNews createResourceNewsTwitter() {
12               return new TwitterREST();
13       }
14 }
```

La consumación de la conexión y lectura de noticias termina siendo atendida en última instancia por `SAXParserRSS`. Esta clase es una implementación personalizada de un lector RSS ordinario. Esto es porque Twitter nos brinda un mecanismo para obtener las actualizaciones de estado en formato RSS a través del URI `http://twitter.com/statuses/user_timeline.rss`, por lo que utilizando un cliente HTTP podemos obtener las actualizaciones sin mayores alineaciones.

TwitterREST.java

```
1 public class TwitterREST extends ResourceNews {
2
3       @Override
4       public List<Message> getMessages(String uid) {
5               // uid usuario de twitter
6               SAXParserRSS saxParserRSS = new SAXParserRSS();
7               return
saxParserRSS.getMessages("http://twitter.com/statuses/user_timeline.rss?id=" + uid);
8       }
9 }
```

Una vez obtenidas las novedades desde la fuente de datos, estas se muestran en un listado implementado con un componente View de la SDK de Android, `ListActivity`.

Para implementar un listado dentro de la actividad que visualiza la entidad, utilizamos directamente el `ListView`, un componente de interfaz de usuario. Para esto, creamos una actividad que incrustaremos en la vista principal, que llamamos `ListNewsActivity` heredando `ListActivity`, que ofrece un manejo simplificado de las listas. Por ejemplo usted tiene un método predefinido, si alguien hace clic en un elemento de la lista.

Este tipo de actividad contiene un `ListAdapter`, que se encarga de gestionar los ítems que contendrá la lista. Este adaptador se debe establecer en el `onCreate` de la actividad a través de la llamada al método `setListAdapter()`. Capturamos la selección del usuario en la lista implementando el evento `onItemClik()`, quien tiene acceso al elemento seleccionado.

ListNewsActivity.java

```
 1 public class ListNewsActivity extends ListActivity {
 2       private ProgressDialog progressDialog = null;
 3       private Entity entity;
 4       private List<Message> messages;
 5       private ArrayAdapter<Message> adapter;
 6
 7       @Override
 8       protected void onCreate(Bundle savedInstanceState) {
 9               super.onCreate(savedInstanceState);
10               entity =
Resource.getInstance().getEntity(getIntent().getExtras().getLong("idEntity"));
11               adapter = new MessageNewsAdapter(getApplicationContext(),
R.layout.list_item_news, messages);
12               setListAdapter(adapter);
13               Runnable viewOrders = new Runnable() {
14                       public void run() {
15                               getMessages();
16                       }
17               };
18               Thread thread = new Thread(null, viewOrders, "MagentoBackground");
19               thread.start();
20               progressDialog = ProgressDialog.show(this, "", "Cargando
Noticias....");
21               getListView().setOnItemClickListener(this);
22       }
23       ...
49
50       @Override
51       public void onItemClick(AdapterView<?> arg0, View arg1, int position, long
arg3) {
52               Message message = messages.get(position);
53               Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse(message.getLink()));
54               startActivity(i);
55       }
56 }
```

Vemos como al recibir la entidad a la cual se quieren obtener las novedades, creamos un adaptador para la lista y, mientras mostramos un mensaje para mantener paciente al usuario, creamos un hilo para recuperarlas a través de `getMessages()`. Luego, a través de `setOnItemClickListener()`, indicamos que va a ser la misma actividad la que atienda a la selección de un ítem a través de `onItemClick()`.

Por último, al seleccionar un ítem de la lista, lo que se haces es ir una URL definida en la propiedad link de la clase `Message`, que representa una noticia de la entidad.

Describimos y diferenciamos como se muestras las noticias de cada tipo de lector. Como dijimos, tenemos otras tres implementaciones de lectores de noticias: RSS, Facebook y Página Web.

RSS

El lector de noticias a través de RSS, es de similar implementación al lector Twitter, ya que este primero dijimos que podía implementarse a través del estándar RSS. Desde el lado visual de la aplicación, se comporta igual, implementándose a través de la actividad `ListNewsActivity`.

El parámetro que se define en la definición de la entidad, se utiliza como URL donde conectar al servicio RSS para obtener las novedades.

Web

El lector de novedades Web difiere en su implementación a los anteriores. En este no mostramos las novedades en una lista, sino que se muestra en un componente tal cual la devuelve el servidor. Este tipo de lector fue necesario para aquellas fuentes que no tienen un formato estandarizado.

El componente es un `WebView`, un elemento de interfaz de usuario que permite visualizar paginas HTML con soporte para JavaScript(*).

Mostramos a continuación como implementamos la actividad que incrustaremos y como cargamos la URL a mostrar en el componte.

WebNewsActivity.java

```
 1 public class WebNewsActivity extends Activity {
 2       private ProgressDialog progressDialog;
 3       private WebView webView;
 4       Entity entity;
 5
 6       @Override
 7       protected void onCreate(Bundle savedInstanceState) {
 8               super.onCreate(savedInstanceState);
 9               setContentView(R.layout.view_news_web);
10               webView = (WebView) this.findViewById(R.id.webview);
11               entity =
Resource.getInstance().getEntity(getIntent().getExtras().getLong("idEntity"));
12               progressDialog = ProgressDialog.show(this, "Por favor espere..",
"Cargando Noticias....");
13               Thread thread = new Thread() {
14                       @Override
15                       public void run() {
16                               // cargar url
17
        webView.loadUrl(entity.getReaderNews().getParameter());
18                               try {
19                                       while (webView.getProgress() < 100)
20                                               sleep(100);
21                                       progressDialog.dismiss();
22                               } catch (InterruptedException e) {23
24                               }
25                       }
```

```
26                  };
27                  thread.start();
28          }
29  }
```

Vemos como implementamos el cargamos la URL parametrizada para el lector de la entidad a través del método `loadUrl()` (línea 17) .

Como lo hicimos anteriormente, lo hacemos en un thread y mostramos una ventana de progreso mientras no se haya cargado completamente la pagina (línea 17 a 21).

7.5 - BUSCADOR DE EVENTOS

En este punto describiremos como incluimos un buscador de eventos en nuestra actividad Mapa. Este buscador lo hemos implementado de acuerdo a la interfaz de búsqueda definida para los dispositivos con Android.

La mayoría de estos dispositivos cuenta con un botón de acción de hardware para Buscar. Cada actividad de cada aplicación puede definir el comportamiento que necesite al presionarse este botón.

En nuestro caso, desplegaremos el cuadro de búsqueda estándar que nos brinda Android, y mostraremos los resultados de búsqueda mediante una petición al servidor.

Describimos las acciones necesarias para implementar esta funcionalidad, resumiendo de manera descriptiva los pasos a seguir:

- Necesitaremos una configuración para el buscador, que incluya datos descriptivos, si permite buscador por voz, idioma de entrada, proveedor de palabras sugeridas, etc. Esta configuración la definimos en un archivo XML.
- Tendremos que definir la actividad que se encargue de realizar la búsqueda y mostrar los resultados.
- Por último, debemos definir la manera en que se invoque al buscador de la aplicación y de que manera.

Searchable.java

```
 1 public class Searchable extends ListActivity implements OnItemClickListener {
 2       static String TAG = Searchable.class.getName();
 3       List<Event> events;
 4
 5       @Override
 6       public void onCreate(Bundle savedInstanceState) {
 7               super.onCreate(savedInstanceState);
 8               setContentView(R.layout.search);
 9
10               Intent intent = getIntent();
11               if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
12                       String query = intent.getStringExtra(SearchManager.QUERY);
13                       this.setTitle("Resultados de búsqueda de eventos para '" +
query + "'");
14                       doSearch(query);
15               } else
16                       Searchable.this.finish();
17               getListView().setOnItemClickListener(this);
18       }
19       ...
20 }
```

Vemos arriba la definición de la actividad encargada de la búsqueda.

Primeramente obtenemos el `Intent` con el que fue creada la actividad, y le pedimos el parámetro adicional `SearchManager.QUERY`, que contiene el texto a buscar ingresado por el usuario. Luego, en la línea 14 hacemos la llamada al método `doSearch(query)`, que se encarga de hacer la llamada al servidor y mostrar los resultados a modo de lista.

Luego debemos modificar el archivo de manifiesto de la aplicación para que la actividad MainMap utilice el buscador definido:

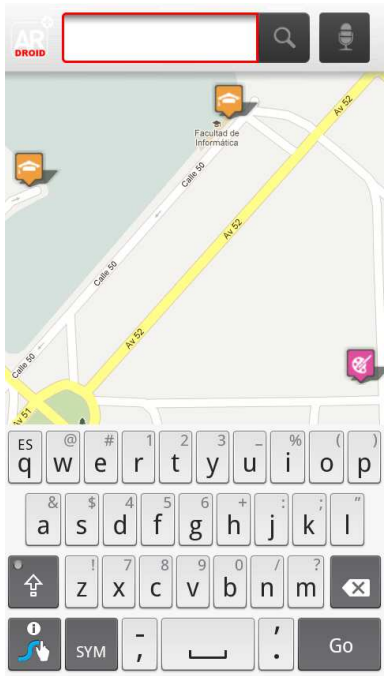
AndroidManifest.xml

```
<activity android:name=".Searchable" android:label="@string/app_name">
       <intent-filter>
              <action android:name="android.intent.action.SEARCH" />
       </intent-filter>
       <meta-data android:name="android.app.searchable"
              android:resource="@xml/search" />
</activity>
<activity android:name=".MainMap" android:label="@string/app_name">
       <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.DEFAULT" />
```

```
        </intent-filter>
        <meta-data android:name="android.app.default_searchable"
                android:value=".Searchable" />
</activity>
```

Vemos que tenemos una nueva actividad definida que es la recientemente implementada, que define como filtro de intent `android.intent.action.SEARCH`. A través de un `meta-data` en la definición de nuestra actividad de mapa, le indicamos que utilice el buscador definido anteriormente.

El resultado de esto es una interfaz agregable, con un método de entrada fácil de utilizar que sigue el estándar de Android.



7.6 - COMO LLEGAR A UN DESTINO

Una de las funcionalidades que nos sentíamos obligados a implementar, era la de poder determinar y dibujar una ruta a las entidades o eventos desde la ubicación actual de usuario. Las personas en el ámbito universitario están habituadas a asistir a cursos y conferencias en entidades que tal vez le son desconocidas, por lo que deben conocer la forma de llegar.

De las aplicaciones analizadas la mayoría no contaba con esta funcionalidad y, desde nuestro punto de vista, le da valor agregado al prototipo al tener la posibilidad de dirigir al usuario al destino. Aunque vale decir no es la finalidad del mismo reemplazar un dispositivo GPS capaz de hacer una navegación en tiempo real, por lo que solo nos limitaremos a imprimir una ruta sobre el mapa desde la posición actual del usuario hasta el destino seleccionado.

El menú de las ventanas de información de las envides y eventos cuenta con una opción de cómo llegar:

La implementación la hicimos a través de la *API directions* [51] de Google y ha resultado algo compleja de lo que en principio pensábamos pues no está disponible para Android, sino que tuvimos que acceder a ella a través de su versión Web. Aunque el hecho de que la interacción se hace a través de JSON nos ha facilitado la tarea. Contamos sin más detalle como implementamos esta funcionalidad, ya que la codificación requeriría la descripción de la API de direcciones, y no es la finalidad del prototipo.

Conociendo la ubicación actual del usuario, debemos llegar dibujar una ruta sobre el mapa hasta algún destino. Primeramente obtendremos una sesión con el servicio de direcciones por medio de la clase `DriveingDirections` que se encarga por medio de `doConnectToService(startPoint, endPoint, mode)` de conectar y hacer la petición al servicio. Luego se interpreta la respuesta JSON y la mapeamos a un modelo local, implementado por puntos, caminos y *steps*. Ya en este punto, si existe un recorrido hacia el destino, decimos que tenemos un objeto `Route` para determinar el camino hacia la ubicación seleccionada.

Definimos una interfaz a implementar por la actividad que lo requiera para realizar acciones al momento de recibidas las indicaciones de cómo llegar:

IDirectionsListener.java

```
1 public interface IDirectionsListener {
2
3       public void directionAvailable(Route route);
4       public void directionNotAvailable();
5
6 }
```

Hicimos que la actividad mapa implementara esta interfaz, por lo que el usuario al momento de querer llegar a una entidad, por ejemplo, se determina la ruta y se imprime la ruta:

MainMap.java

```
 1 public class MainMap implements IDirectionsListener {
 2
 3       ...
 4       @Override
 5       public void directionAvailable(Route route) {
 6
 7               Iterator<Leg> xIterator = route.getLegs().iterator();
 8               while (xIterator.hasNext()) {
 9                       Leg leg = xIterator.next();
10                       Iterator<Step> xItSteps = leg.getSteps().iterator();
11                       while (xItSteps.hasNext()) {
12                               Step step = xItSteps.next();
13                               for (int i = 1; i <
step.getPolyline().getPolylines().size(); i++) {
```

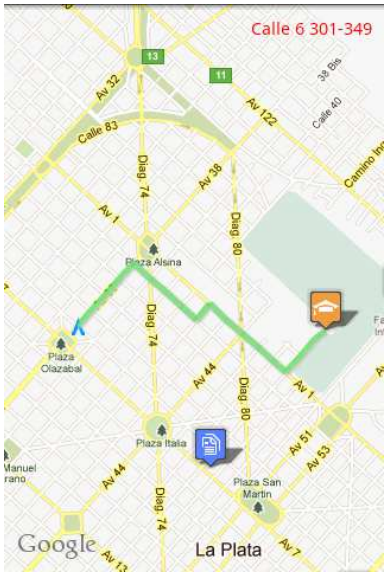
```
14                                          RouteOverlay routeOverlay = new
RouteOverlay(null, mapView.getContext(), step.getPolyline()
15                                                  .getPolylines().get(i - 1),
step.getPolyline().getPolylines().get(i), Color.BLUE);
16                                          mapView.getOverlays().add(routeOverlay);
17                                  }
18                          }
19                  }
20                  mapView.postInvalidate();
21          }
22          ...
23
24  }
```

El objeto `Route` esta compuesto por una lista de `steps` que básicamente contienen un punto de localización y una descripción. Utilizamos únicamente los la ubicación para crear `Overlay` de tipo línea para dibujar la ruta completa.

La interfaz obliga a implementar un método en el caso de que no se encuentre o no se pueda determinar un recorrido hacia el destino. Aquí mostramos al usuario un mensaje de lo ocurrido.

```
1           ...
2           @Override
3           public void directionNotAvailable() {
4                   progressDialog.dismiss();
5                   Toast.makeText(getApplicationContext(), "No se encontró un recorrido
hasta el destino", Toast.LENGTH_LONG).show();
6           }
7           ...
```

El prototipo es capaz de dibujar dos distintos modos de rutas dependiendo la selección del usuario. El modo de ruta en auto, y el modo caminando. Esto es parametrizable desde la pantalla de configuración del prototipo y para distinguir el modo seleccionado, se muestran las rutas a pie en azul, mientras que las rutas en auto se dibujan en verde.



Hemos simplificado el dibujado de la ruta a la primera posible. Además, esta no se actualiza al moverse el usuario por ella, sino que permanece estática.

Cuando el usuario decida ya no mostrar la ruta, incluimos una opción en el menú la opción *Borrar ruta* para realizar esta acción, que solo se presenta si es que hay una ruta impresa.


Av 1
74
Diag 80
Facultad de
Tipos de Eventos
Mi posición
Borrar ruta
Volver

7.7 - FEEDBACK DE LOS EVENTOS

La funcionalidad de retroalimentación del prototipo fue uno de los principales puntos definidos por su importancia. Tener la posibilidad de visualizar opiniones y recibir comentarios de los participantes o asistentes a un evento o lugar es significativo en las áreas de marketing.

La posibilidad de recibir un feedback(*) por parte de los usuarios de un producto o servicio permite mejorar este y tener una visión mas descifrada de la aceptación de estos.

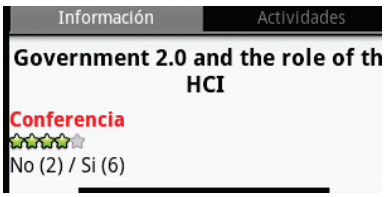
Definimos e implementamos para el prototipo 4 tipos de feedback posibles:

- *Me gusta*: Permite que el usuario de una simple opción para un evento, por Si o No.
- *Múltiples opciones*: Presenta al usuario una lista acotada de opciones seleccionables para responder a una pregunta en particular.
- *Comentarios*: Permite al usuario responder a una pegunta con libertad.
- *Calificación numérica*: Brinda al usuario emitir su opinión sobre un evento mediante un rango numérico.

Me gusta

Este tipo de Feedback es el estándar predeterminado para todo nuevo evento que se defina. Define un simple diálogo con las opciones Si y No. La acumulación de respuestas se muestra en la ventana de información para el evento.

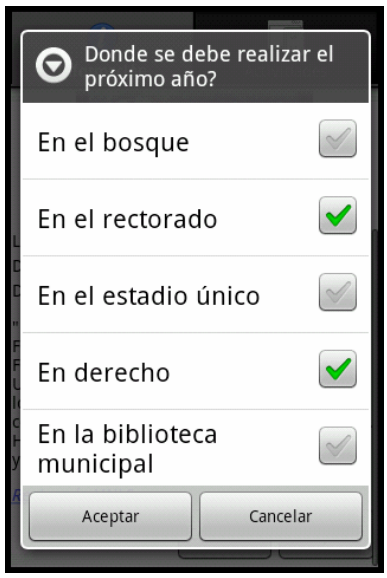
Veamos este tipo de feedback como una subclasificación del tipo múltiples opciones. Al tener dos opciones preestablecidas, podemos definir un promedio entre esas respuestas y mostrar en forma de barra la cantidad de calificaciones positivas que recibió un evento.



Múltiples opciones

El tipo de parametrización permite seleccionar una o varias opciones desde una lista de opciones definidas.

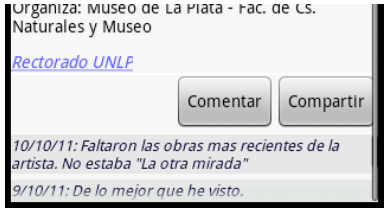
Tanto en funcionalidad como implementación es muy similar al tipo anterior, aunque al no tener un número predefinido de opciones y no contar con una distinción entre positivas y/o negativas, no es posible hacer un promedio de opiniones como en el caso de "Me gusta".



Comentarios

El tipo de feedback de preguntas textuales permite responder a una pregunta de forma abierta. Las respuestas se envían al servidor para luego mostrarse en forma de lista debajo de la información del evento. El numero de comentarios a visualizar lo limitamos a 5. Por lo que si hay mas comentarios, se agregará debajo de la lista un enlace "Ver mas" que llevará a la Web donde se visualiza la lista completa de respuestas.

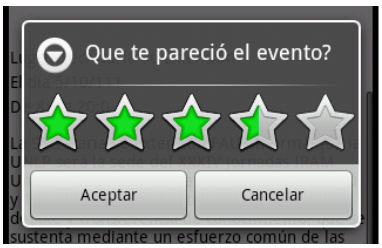
Este tipo de feedback no brinda ningún tipo de información estadística, pero si puede obtener opiniones mas concretas sobre determinado tema, por lo que puede ser útil en determinados tipos de eventos.



Calificación numérica

Este tipo de feedback permite obtener una valoración tipo barra. Se define un rango numérico, comúnmente 1-10, y desde los clientes se reciben respuestas en ese rango a través de la utilización de una barra de estrellas.

Visiblemente, es más vistoso que una simple entrada de número. Al ser respuestas numéricas dentro de un rango, estas se pueden cuantificar y realizar un promedio de todas las votaciones que se recibieron para el evento, y mostrar el ranking.



Describimos como implementamos esté dinamismo en la obtención de un feedback ejemplificándolo con la implementación estándar de "Me gusta".

Encontramos dos piezas implementadas en esta funcionalidad, una es la descripción y muestreo de las estadísticas, y otra es la confección del dialogo para obtener el feedback del usuario. Ambas están implementadas en la actividad `EventActivity`.

EventActivity.java

```
 1 public class EventActivity extends Activity implements
android.view.View.OnClickListener, IQuestionListener {
 2       private Event event;
 3       private Entity entity;
 4
 5       @Override
 6       protected void onCreate(Bundle savedInstanceState) {
 7               super.onCreate(savedInstanceState);
 8               ...
 9               TextView rank = (TextView) this.findViewById(R.id.rank);
10               rank.setText(summary.getDescription());
11               RatingBar ratingbar = (RatingBar) this.findViewById(R.id.ratingBar);
12               if (summary.getRating() != -1) {
13                       ratingbar.setRating(summary.getRating());
14               }
15
16               ListView comments = (ListView) this.findViewById(R.id.comments);
17               if (!summary.getComments().isEmpty()) {
18                       rank.setAutoLinkMask(0);
19                       rank.setMovementMethod(LinkMovementMethod.getInstance());
20                       ...
21                       comments.setAdapter(new ArrayAdapter<String>(this,
R.layout.row_list_comment, R.id.comment, summary
22                                       .getComments()));
23               } else
24                       comments.setVisibility(View.INVISIBLE);
25               ...
26               Button btbCommnet = (Button) findViewById(R.id.buttonComment);
27               if (event.getSurveyResponse() == null) {
28                       btbCommnet.setOnClickListener(new View.OnClickListener() {
29                               public void onClick(View v) {
30                                       showDialogCommentEvent(v);
31                               }
32                       });
33               } else {
34                       btbCommnet.setVisibility(View.INVISIBLE);
35               }
36       }
37
38       protected void showDialogCommentEvent(View v) {
39               if (!event.getSurveyTemplate().getQuestions().isEmpty()) {
40
       event.getSurveyTemplate().getQuestions().get(0).onQuestion(this);
41               }
42       }
43       ...
44 }
```

Primeramente se arma la estructura de resultados de las respuestas al crear la actividad. Se tiene un objeto que `SummaryResponse` que contiene el resumen de promedios y cantidad de respuestas. A través de `summary.getDescription()` se obtiene el texto descriptivo. El objeto también contiene, si

corresponde al tipo de feedback, los comentarios o el promedio calculado.

Si corresponde a un tipo calculado, se setea el promedio (líneas 11-14) en un objeto de vista `RatingBar`. Si corresponde a un tipo de comentarios, se completa la lista de comentarios (líneas 16-24).

Por último, si el evento tiene parametrizado un tipo de retroalimentación, se configura el botón para comentar (líneas 26-35). La acción del botón hace la llamada a `showDialogCommentEvent()`. Este método ejecuta `onQuestion` para el tipo de encuesta parametrizado, y así a través de un *double distpaching* este objeto le dirá a la actividad que acción realizar.

En el caso del tipo "Me gusta", la labor del método anterior es simplemente `callback.doMultipleChoiceQuestion(this);`

EventActivity.java

```
 1 public class EventActivity extends Activity implements
android.view.View.OnClickListener, IQuestionListener {
 2       ...
 3
 4       @Override
 5       public void doMultipleChoiceQuestion(final MultipleChoiceQuestion
multipleChoiceQuestion) {
 6               AlertDialog.Builder builder = new AlertDialog.Builder(this);
 7               builder.setTitle(multipleChoiceQuestion.getQuestion());
 8
 9               final String options[] = new
String[multipleChoiceQuestion.getOptions().size()];
10               final boolean selected[] = new
boolean[multipleChoiceQuestion.getOptions().size()];
11
12               Iterator<Choice> option =
multipleChoiceQuestion.getOptions().iterator();
13               int i = 0;
14               while (option.hasNext()) {
15                       Choice choice = (Choice) option.next();
16                       options[i] = choice.getDescription();
17                       selected[i] = false;
18                       i++;
19               }
20               builder.setPositiveButton("Aceptar", new
DialogInterface.OnClickListener() {
21                       public void onClick(DialogInterface dialog, int whichButton)
{
22                               // guardar la opción elegida
23                               SurveyResponse surveyResponse = new
SurveyResponse();
24                               surveyResponse.setEvent(event);
25                               MultipleChoiceResponse multipleChoiceResponse = new
MultipleChoiceResponse();
26
        multipleChoiceResponse.setMultipleChoiceQuestion(multipleChoiceQuestion);
27                               for (int j = 0; j < selected.length; j++) {
28                                       if (selected[j]) {
29
        multipleChoiceResponse.addOption(multipleChoiceQuestion.getOptions().get(j));
30                                       }
31                               }
32                               surveyResponse.addResponse(multipleChoiceResponse);
33
        ResourceHelperFactory.createResourceHelper().saveResponse(surveyResponse);
34                               event.setSurveyResponse(surveyResponse);
35                       }
36               });
37               ...
38
39               AlertDialog alert = builder.create();
40               alert.show();
41       }
42       ...
43 }
```

El método `doMultipleChoiceQuestion` se encarga de construir y abrir un dialogo para

capturar la opción del usuario (líneas 20-40). Pero antes, entre las líneas 12-19, se arma la lista de respuestas posibles, que en este caso son "Si" y "No".

Una vez obtenida la respuesta del usuario al presionar este el botón Aceptar, se instancia un objeto `MultipleChoiceResponse` que luego se guarda a través de la llamada a `ResourceHelperFactory.createResourceHelper().saveResponse(surveyResponse);`.

Esta llamada incluye la serialización del objeto a JSON para luego a través del cliente HTTP que se encarga de la comunicación con el servidor, se envíe el requerimiento PUT para ser atendido por el servidor.

Para los otros tipos de feedback el procedimiento es el mismo. Solo cambian de acuerdo a la parametrización como se arma el diálogo para capturar datos y como se arma el objeto `SurveyResponse` que se envía al servidor.

7.7.1 - Compartiendo eventos

Describimos en este punto la funcionalidad de "Compartir" con la que cuentan todos los eventos definidos para las entidades.

Nos encontramos con una problemática a la hora de implementar la funcionalidad "Compartir", tradicional en este tipo de aplicaciones. Esta era, el costo de incluir todas las APIs para comunicarse con los distintos servicios.

Queríamos permitir que los eventos de nuestro prototipo se puedan compartir a través de Twitter, Facebook, Google+, pero también queríamos evitar implantar cada una de las autenticaciones necesarias para publicar contenido en las cuentas del usuario, y así evitarle al usuario iniciar sesión en un servicio para compartir un enlace. Un análisis rápido de la plataforma nos hizo entender la potencia de los filtros Intents (intenciones). Hemos utilizado los *Intents* a lo largo del desarrollo, por lo que daremos aquí una breve descripción de los mismos.

Los filtros de *Intents* son, desde nuestro punto de vista, una de las características más interesantes de Android. Los hemos utilizado en gran parte de la implementación del prototipo. Nos permiten reutilizar componentes de otras aplicaciones usando muy pocas líneas de código.

Entonces, solo tuvimos que implementar un par de líneas para obtener un completo selector de aplicaciones o servicios con que compartir nuestro contenido.

EventActivity.java

```
 1 public class EventActivity extends Activity implements OnClickListener,
IQuestionListener {
 2       ...
 3       protected void showDialogShare() {
 4                ARDROIDProperties properties = ARDROIDProperties.getInstance();
 5                Intent sendIntent = new Intent(Intent.ACTION_SEND);
 6
 7                sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Evento " +
this.event.getTitle());
 8                sendIntent.putExtra(
 9                                 Intent.EXTRA_TEXT,
10                                 "Hola! Voy a asistir al evento " +
this.event.getTitle() + ". Mas información en "
11                                                                 +
properties.getProperty("ar.droid.server") + "/event/evento/" + this.event.getId() +
".");
12                sendIntent.setType("text/plain");
13                startActivity(Intent.createChooser(sendIntent, "Compartir con"));
14       }
15       ...
16 }
```

Una vez creado el `Intent`, lo que se hace es comenzar una nueva actividad con la función `startActivity` y le pasamos como parámetro un `createChooser` (esto creará una ventana flotante de elección de aplicaciones) con el texto que queramos que salga en esa ventana flotante.

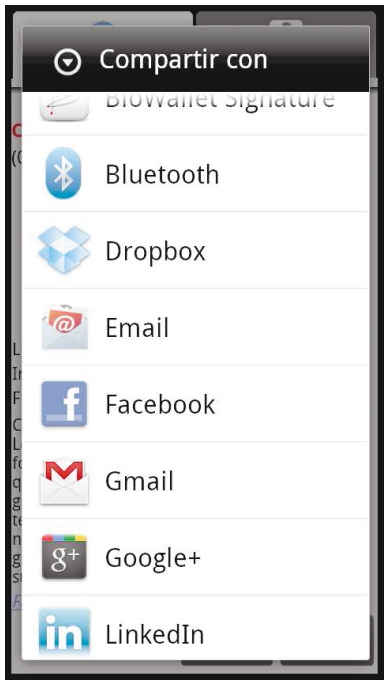
Vemos en la línea 5, que la "intención" que creamos es del tipo `Intent.ACTION_SEND`. Este tipo de acción permite al usuario enviar el contenido a una persona o sistema, aunque todavía no esta definido como se enviará esa información. En las siguientes líneas (7 a 11) definimos los parámetros

particulares de `Intent`:

- `Intent.EXTRA_SUBJECT`: En este parámetro indicamos mediante un String el asunto de la información a enviar.
- `Intent.EXTRA_TEXT`: Indicamos mediante un String con el contenido de la información.

En la última línea, con `Intent.createChooser()`, estamos creando un *Intent* de tipo `EXTRA_CHOOSER`. Se utiliza este método por convención en vez de crearlo de la manera habitual (new Intent(...)). Lo que conseguimos es lanzar una pantalla en la que podemos elegir una *Activity*, que en nuestro caso será la *Activity* encarga de compartir información.

Esta puede ser la activad para enviar un correo, un mensaje de Twitter, Facebook, o de cualquier otra que se haya registrado como para ser capaz de compartir información.



7.8 - IMPLEMENTACIÓN DE REALIDAD AUMENTADA

Ya dimos una definición y describimos la arquitectura de un sistema de realidad aumentada, vamos a detallar cómo encajamos todo y cada uno de los componentes de Android que nos brinda Android y los aprovechamos en la implementación de RA.

Básicamente, la implementación de RA consiste en utilizar las API de la cámara, las API de gráficos, y las API del sensor de la superposición. Y a esto le agregamos los datos de aumento sobre la captura de video en tiempo real para crear una experiencia aumentada.

Describiremos los cuatro componentes fundamentales en todo sistema de RA, que son:

- *Captura de video*: Visualización de la captura en vivo desde la cámara
- *Datos de ubicación*: Capacidad de determinar la ubicación del dispositivo
- *Datos de los sensores*: Los sensores son importantes para las implementaciones de RA. Conocer la orientación del teléfono es fundamental cuando se trata de mantener los datos sincronizados con la transmisión de la cámara.
- *Superposición gráfica*: Por supuesto, todo el sentido de la RA consiste en dibujar algo sobre la captura de la cámara que, además, *aumenta* lo que el usuario está viendo en vivo.

La implementación de la RA es compleja por la cantidad de algoritmos y estructuras que se utilizan para el cálculo de posición y orientación del dispositivo. Por lo que describiremos los aspectos funcionales más relevantes de este módulo de prototipo, sin evocarnos demasiado en estos algoritmos. También mostraremos pantallas de esta funcionalidad, aunque incompletas, pues los programas de captura no tomaban la captura de video en el backend.

En un vistazo general, la implementación del modulo puede verse en el siguiente diagrama:

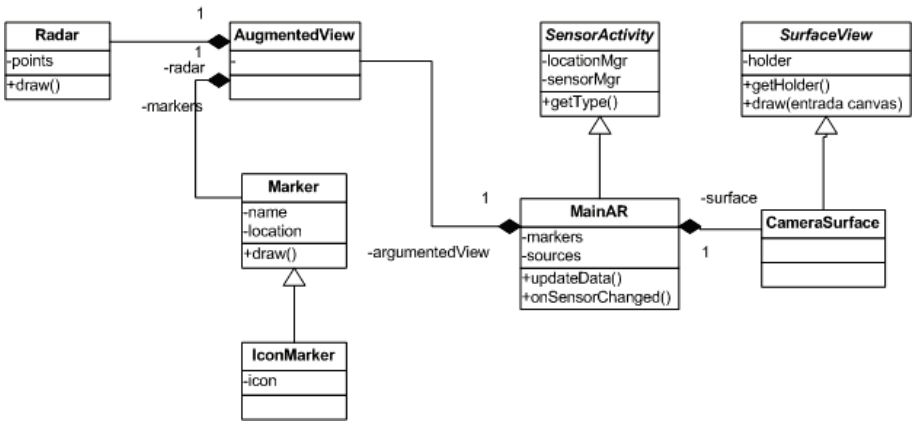


Figura 24. Diagrama de clases básico del módulo de RA

Damos una descripción explicativa de cada una de las clases que componen el diagrama anterior, mas adelante veremos algo más en detalle como funcionan:

- *MainAR*: Es la actividad principal que implementa la realidad aumentada.
- *SensorActivity*: Clase que se encarga de la inicialización y atención de los distintos sensores necesarios.
- *CameraSurface*: Implementa una vista para la captura de video.
- *ArgumentedView*: Es la clase que parametriza y controla los elementos de realidad aumentada a superponer a la captura de video.
- *Radar*: Representa el objeto radar a incluir en la vista de RA.
- *Marker*: Es la representación de una entidad a visualizar en pantalla.
- *IconMarker*: Implementa una entidad que a la que se le asocia una imagen para mostrar en pantalla.

En principio, la funcionalidad del modelo comienza con la captura de video, tarea que veremos mas adelante. Luego obtenemos los elementos que mostraremos sobre el video, y para esto hemos creado una interfaz a modo de fuente de datos a implementar, que debe devolver las entidades a mostrar.

Datasource.java

```
1 public abstract class DataSource {
2       protected static final int MAX = 10;
3
4       public abstract List<Entity> getEntities(double lat, double lon, double alt);
5 }
```

En nuestro caso, y para simplificar la tarea, solo mostraremos las entidades definidas en el servidor, dejando de lado los eventos. Por lo que nuestra implementación `ARDroidDataSource`, se limita a devolver las entidades leídas desde el servidor.

ARDroidDatasource.java

```
1 public class ARDroidDataSource extends DataSource {
2
3       @Override
4       public List<Entity> getEntities(double lat, double lon, double alt) {
5               // retorno las entidades cacheadas
6               return Resource.getInstance().getEntities();
7       }
8 }
```

Definimos la clase `ARData` para centralizar las tareas a realizar y no ensuciar demasiado la codificación de las actividades. La clase `ARData` se encarga, entre otras cosas, de almacenar los *markers* a mostrar, actualizar las superposiciones con los cambios de localización y orientación.

ARData.java

```
 1 public abstract class ARData {
 2       private static final Map<Long, Marker> markerList = new
ConcurrentHashMap<Long, Marker>();
 3       public static void setCurrentLocation(Location currentLocation) {
 4               ARData.currentLocation = currentLocation;
 5               onLocationChanged(currentLocation);
 6       }
 7
 8       public static void addMarkers(List<Marker> markers) {
 9               for (Marker ma : markers) {
10                       if (!markerList.containsKey(ma)) {
11
       ma.calcRelativePosition(ARData.getCurrentLocation());
12                               markerList.put(ma.getEntity().getId(), ma);
13                       }
14               }
15       }
16
17       public static void onLocationChanged(Location location) {
18               for (Marker ma : markerList.values()) {
19                       ma.calcRelativePosition(location);
20               }
21       }
22
23       public static Collection<Marker> getMarkers() {
24               return markerList.values();
25       }
26
27       public static Marker getMarker(int index) {
28               Long key = (Long) markerList.keySet().toArray()[index];
29               return markerList.get(key);
30       }
31       ...
32 }
```

Entonces, ya tenemos las entidades a mostrar y la vista previa de video, solo nos resta mostrar las entidades en los momentos que correspondan, y esto es tarea de la clase `AugmentedView`.

AugmentedView.java

```
1  public class AugmentedView extends View {
2        private static Radar radar = null;
3
4        public AugmentedView(Context context) {
5                super(context);
6                if (radar == null)
7                        radar = new Radar();
8        }
9
10       @Override
11       protected void onDraw(Canvas canvas) {
12               // Dibujar los marcadores RA!
13               for (Marker marker : ARData.getMarkers()) {
14                       marker.draw(canvas);
15               }
16               radar.draw(canvas);
17       }
18 }
```

Esta es toda la implementación de la clase. Su finalidad es dibujar las superposiciones sobre la vista previa de video. Pero notamos que por su simplicidad, no es quien realmente implementa la superposición, sino que el `onDraw` realmente delega a los *markers* que se dibujen sobre un objeto `Canvas`. Veremos mas adelante que esta tarea se realiza repetidamente, siempre que la orientación del dispositivo cambie, y sea necesario redibujar las entidades.

De manera general, esta es la implementación de la RA. El ciclo, a excepción de la obtención de las entidades, se repite continuamente.

Veamos que la actividad tiene un menú para volver a la vista mapa, además del resto de los elementos que componen la vista de RA. Arriba a la izquierda, imprimimos lo que es el radar de RA. Este incluye la orientación actual del dispositivo, el ángulo de visión, el alcance actualmente configurado, y las entidades que podemos encontrar en ese alcance.

Veremos a continuación los aspectos más relevantes de implementación de la funcionalidad de Realidad Aumentada.

7.8.1 - Captura de video

La captura de video la hacemos a través de componentes que nos brinda la plataforma.

Antes que nada, necesitamos incluir un par de permisos a nuestro AndroidManifest.xml para tener acceso a la cámara del dispositivo:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
```

Como vimos a lo largo de este trabajo, Android tiene un conjunto de clases que para acceder y manipular el hardware con el que cuentan los dispositivos. La cámara de video es un elemento común en los móviles de hoy, por lo que se ha hecho que el acceso a ella sea realmente sencillo. Extendemos la clase `SurfaceView` para las personalizaciones necesarias el *preview* de video. Lo que hacemos con esta

La clase de `Camera` se utiliza para configurar los ajustes de captura de imágenes, de arranque / parada de vista previa, y recuperar los marcos para la codificación de vídeo. Esta clase es un cliente para el servicio de la cámara, que gestiona el hardware de la cámara real. Es requerimiento que también contemos con un `SurfaceHolder`, sin este, no podremos inicializar la vista previa de video.

CameraSurface.java

```
1  public class CameraSurface extends SurfaceView implements SurfaceHolder.Callback {
2      private static SurfaceHolder holder = null;
3      private static Camera camera = null;
4
5      public CameraSurface(Context context) {
6          super(context);
7
8          try {
9              holder = getHolder();
10             holder.addCallback(this);
11             holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
12         } catch (Exception ex) {
13             logger.info("Exception: "+ex.getLocalizedMessage());
14         }
15     }
16
17     public void surfaceCreated(SurfaceHolder holder) {
18         ...
19     }
20
21     public void surfaceDestroyed(SurfaceHolder holder) {
22         ...
23     }
24
25     public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
26         ...
27     }
28 }
```

Como se ve arriba, existe una seria de llamadas callback que debemos implementar, para cuando la captura es creada, destruida, o cambia (rotación de pantalla, por ejemplo). Esta implementación de `SurfaceView` la utilizamos en nuestra actividad principal de RA:

MainAR.java

```
1  public class MainAR extends SensorsActivity {
2       private static CameraSurface camScreen = null;
3       private static AugmentedView augmentedView = null;
4       private static List<Marker> markers = null;
5
6       @Override
7       public void onCreate(Bundle savedInstanceState) {
8               super.onCreate(savedInstanceState);
9               camScreen = new CameraSurface(this);
10              setContentView(camScreen);
11              ...
12      }
13      ...
14 }
```

Notamos que al crear la actividad creamos un `CameraSurface` y lo establecemos como la vista de la actividad. Esto nos permitirá tener una captura de video a pantalla completa. Como pudimos suponer, esta clase esta dentro de la jerarquía de `View`.

En este punto nos hemos encontrado con algunos problemas de compatibilidad en algunos dispositivos. Estos pueden tener diferentes especificaciones de hardware, tales como las calificaciones y las capacidades de enfoque automático. Para que nuestra aplicación sea compatible con todos los dispositivos, no hicimos suposiciones sobre las especificaciones y la vista previa es básica.

7.8.2 - Datos de ubicación

La ubicación del dispositivo para la realidad aumentada la hacemos de la misma manera que la ubicación de la actividad mapa, a través de un *listener* de GPS, tal como se describió anteriormente en "Ubicando al cliente".

Al tener distintas tareas que realizar al obtener una nueva ubicación, implementamos `LocationListenerGPSAR`, que básicamente es igual a `LocationListenerGPS`, con la diferencia antes nombrada.

LocationListenerGPSAR.java

```
 1 public class LocationListenerGPSAR implements LocationListener {
 2       protected Location lastKnowLocation;
 3       protected MainAR activity;
 4       private TextView locationText;
 5
 6       public LocationListenerGPSAR(Activity activity, TextView locationText,
Location lastKnowLocation) {
 7               this.activity = (MainAR) activity;
 8               this.lastKnowLocation = lastKnowLocation;
 9               this.locationText = locationText;
10               this.showAddress();
11       }
12
13       @Override
14       public void onLocationChanged(Location location) {
15               ARData.setCurrentLocation(location);
16               this.lastKnowLocation = location;
17               this.activity.updateData(location.getLatitude(),
18                       location.getLongitude(), location.getAltitude());
19               // mostrar ubicación
20               this.showAddress();
21       }
22
23       public void showAddress() {
24               ...
25       }
26 }
```

Notamos que la diferencia esta en la definición de `onLocationChanged`. Vemos que al recibir una nueva ubicación se hace un llamado `updateData()`. Veremos mas adelante cual es la finalidad de este método.

7.8.3 - Datos de los sensores

Describimos en este punto como tomamos la orientación del dispositivo para conocer hacia a donde esta mirando y determinar que entidades debemos mostrar hacia esa dirección.

Utilizaremos los sensores de orientación y el acelerómetro para cumplir con esta tarea. Como nombramos al principio, esta funcionalidad la implementamos sobre `SensorsActivity` para que nuestra clase `MainAR` quede más simple.

SensorsActivity.java

```
 1 public class SensorsActivity extends Activity implements SensorEventListener,
OnTouchListener {
 2       ...
 3
 4       @Override
 5       public void onStart() {
 6               super.onStart();
 7               ...
 8               try {
 9                       sensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);
10                       sensors = sensorMgr.getSensorList(Sensor.TYPE_ACCELEROMETER);
11                       if (sensors.size() > 0)
12                               sensorGrav = sensors.get(0);
```

```
13
14                          sensors =
sensorMgr.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
15                          if (sensors.size() > 0)
16                                  sensorMag = sensors.get(0);
17
18                          sensorMgr.registerListener(this, sensorGrav,
19                                          SensorManager.SENSOR_DELAY_UI);
20                          sensorMgr.registerListener(this, sensorMag,
21                                          SensorManager.SENSOR_DELAY_UI);
22
23                          ...
24                          } catch (Exception ex) {
25                                  logger.info("Excepción en sensores! " + ex);
26                          }
27                  } catch (Exception ex1) {
28                          ...
29                  }
30          }
31          ...
32  }
```

Vemos en las línea 9 como obtenemos un manejador para los sensores, y recordamos que es de la misma forma que obtenemos un manager para cuando nos tuvimos que registrar a las actualizaciones de GPS. Esto es porque el terminal GPS no es más que otro tipo de sensor. Es por eso que Android facilita las cuestiones centralizando todo tipo de servicios soportados por el móvil en manejadores que se obtienen desde `getSystemService`.

En las líneas siguientes obtenemos el primero de los sensores registrados para cada tipo, y luego registramos a la actividad como *listener* a través de la llamada a `registerListener` (líneas 18-20). Al registrar la actividad como *listener*, debemos implementar la interfaz `SensorEventListener`, que nos dicta implementar `onSensorChanged`.

SensorsActivity.java

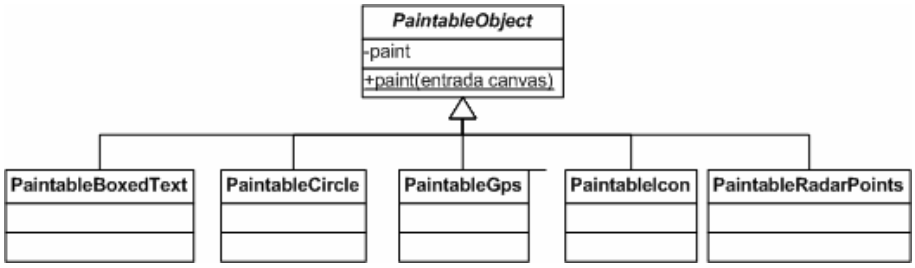
```
 1 public class SensorsActivity extends Activity implements SensorEventListener,
OnTouchListener {
 2          ...
 3          @Override
 4          public void onSensorChanged(SensorEvent evt) {
 5                  if (evt.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
 6                          grav[0] = evt.values[0];
 7                          grav[1] = evt.values[1];
 8                          grav[2] = evt.values[2];
 9                  } else if (evt.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
10                          mag[0] = evt.values[0];
11                          mag[1] = evt.values[1];
12                          mag[2] = evt.values[2];
13                  }
14
15                  SensorManager.getRotationMatrix(RTmp, null, grav, mag);
16                  SensorManager.remapCoordinateSystem(RTmp, SensorManager.AXIS_X,
17                                  SensorManager.AXIS_MINUS_Z, Rot);
18                  ...
19                  ARData.setRotationMatrix(smoothR);
20          }
21          ...
22  }
```

Cuando recibimos una actualización desde los sensores, evaluamos de cual se trata y obtenemos los datos (líneas 5-13). Una vez obtenidas los datos y luego de aplicar algunos cálculos sobre los mismos, delegamos a `ARData` la actualización en la visualización de las entidades en pantalla.

7.8.4 - Superposición gráfica

Sabemos en este punto que las entidades se dibujan en un objeto *canvas* por sobre la captura de video. Veremos mas en detalle este componente, empezando primeramente por mostrar la jerarquía de

objetos principales que intervienen.



Las clases que representan los objetos dibujar extienden de `PaintableObject`, que define el método abstracto `paint(Canvas canvas)` a implementar por las subclases.

Mostramos, a modo de ejemplificar la utilización del *canvas*, la implementación de `PaintableCircle`, que su finalidad es dibujar un circulo en pantalla. Vemos como en la línea 12 se hace la llamada a `drawCircle` del `canvas` recibido. La clase `Canvas` de Android contiene muchas y muy variadas formas de dibujar sobre un canvas.

PaintableCircle.java

```
 1 public class PaintableCircle extends PaintableObject {
 2
 3     public PaintableCircle(int color, float radius, boolean fill) {
 4       set(color, radius, fill);
 5     }
 6
 7     @Override
 8     public void paint(Canvas canvas) {
 9       if (canvas==null) return;
10          setFill(fill);
11          setColor(color);
12          canvas.drawCircle(0, 0, radius);
13     }
14     ...
15 }
```

Mostramos a continuación la implementación de la clase `Marker`, que contiene mucha funcionalidad en cuanto a manejo de vectores al ser la responsable de determinar a que distancia y orientación del usuario esta, dibujarse sobre la pantalla y saber si fue presionada por el usuario.

Para dibujarse, un `Marker` utiliza varios objetos *paintables*, ya que debe dibujarse sobre el radar de RA, y su visualización incluye una imagen sobre un texto (el nombre de la entidad sobre el icono que representa el tipo de entidad).

Marker.java

```
 1 public class Marker implements Comparable<Marker> {
 2     ...
 3     private Entity entity;
 4
 5       public Marker(Entity entity) {
 6               set(entity.getName(), entity.getGeoPoint().getLatitudeE6()/1E6,
entity.getGeoPoint().getLongitudeE6()/1E6,entity.getGeoPoint().getAltitude(),
Color.YELLOW, entity);
 7       }
 8       ...
 9       private void populateMatrices(MixVector originalPoint, CameraModel cam, float
addX, float addY) {
10       ...
11       }
12     private void updateDistance(Location location) {
13       ...
14     }
15       public void calcRelativePosition(Location location) {
16               ...
17       }
18       public void draw(Canvas canvas) {
```

```
19                if (canvas==null) return;
20            update(canvas,0,0);
21            if (!isVisible) return;
22            drawIcon(canvas);
23            drawText(canvas);
24        }
25        public boolean clickedMe(float x, float y) {
26        ...
27        }
28        ...
29 }
```

Vemos que, entre otros métodos definidos, están `populateMatrices`, `updateDistance`, `calcRelativePosition`, que cumplen con la finalidad de actualizar y comprobar orientación y distancia, y métodos de visualización e interacción, como `draw` que pinta el merker sobre la pantalla y `clickedMe` que determina si la pulsación de determinada posición en la pantalla coincide con la posición de el mismo.

7.8.5 - Interacción con los elementos

Cada entidad superpuesta en la vista previa de video permite una mínima interacción. Esto es, al presionar sobre una, se despliega un dialogo semitransparente donde se muestra el nombre de la misma con su descripción acotada.

También un botón a la entidad, que nos permite abrir la actividad de visualización de la entidad para ver sus datos completos.

MainAR.java

```
1  public class MainAR extends SensorsActivity {
2        ...
3        @Override
4        public boolean onTouchEvent(MotionEvent me) {
5                entitySelected = this.getMarkerAt(me.getX(), me.getY());
6                ...
7                Button button = (Button) dialog.findViewById(R.id.button_e_ar);
8                button.setOnClickListener(new View.OnClickListener() {
9
```

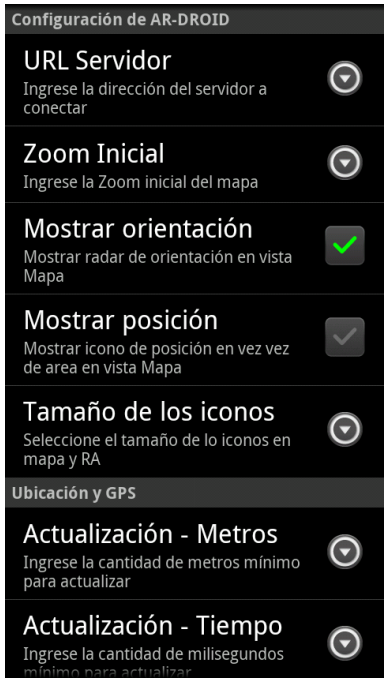
```
10                      @Override
11                      public void onClick(View v) {
12                              Intent i = new Intent(getApplicationContext(),
EntityTabWidget.class);
13                              startActivityForResult(i, 0);
14                              dialog.dismiss();
15                      }
16              });
17              dialog.show();
18              return true;
19      }
20
21      private Entity getMarkerAt(float x, float y) {
22              ...
23      }
24 }
```

La implementación de la actividad responde a los eventos de touch sobre la misma, por lo que no es que queda *marker* tiene su propio evento. Mas bien, a presionar sobre la pantalla, la actividad busca si hay un *marker* en esa posición a través de `getMarkerAt` (línea 5). Si existe, lo siguiente es crea la actividad dialogo para mostrar la información de la entidad contenida en el marker.

7.9 - PREFERENCIAS DE LA APLICACIÓN

Incluimos en el prototipo una actividad para configurar opciones básicas del comportamiento del mismo. A esta opción se accede a través del menú *Configuración* de la vista de mapa.

Mostramos a continuación un pantallaza de las opciones disponibles. Entre ellas, podemos encontrar preferencias de visualización, como mostrar radar de orientación en mapa, el zoom inicial del mismo, tamaño de los iconos en pantalla, y otras configuraciones relevantes al sensor GPS y vista de RA.



La implementación de esta actividad la realizamos con una de las *best practices*(*) que nos brinda la plataforma Android.

Primeramente debemos definir un XML que contendrá los elementos de configuración que queramos añadir a la aplicación.

Mostramos a continuación el resultado de nuestro prototipo:

preferences.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
 3       <PreferenceCategory android:title="Configuración de AR-DROID">
 4               <EditTextPreference android:summary="Ingrese la dirección del
servidor a conectar"
 5               android:defaultValue="http://www.gabrielnegri.com.ar:8080/ardroid"
 6                       android:title="URL Servidor" android:key="urlServerPref"
 7                       android:persistent="true" />
 8               <EditTextPreference android:inputType="number"
 9                       android:summary="Ingrese la Zoom inicial del mapa"
10                       android:defaultValue="14" android:title="Zoom Inicial"
android:key="zoomMapPref"
11                       android:persistent="true" />
12               <CheckBoxPreference android:summary="Mostrar radar de orientación en
vista Mapa"
13                       android:defaultValue="true" android:title="Mostrar
orientación"
14                       android:key="oriMapPref" android:persistent="true" />
15               <CheckBoxPreference
16                       android:summary="Mostrar icono de posición en vez de area en
vista Mapa"
17                       android:defaultValue="true" android:title="Mostrar posición"
18                       android:key="posMapPref" android:persistent="true" />
19               <ListPreference android:title="Tamaño de los iconos"
20                       android:summary="Seleccione el tamaño de lo iconos en mapa y
RA"
```

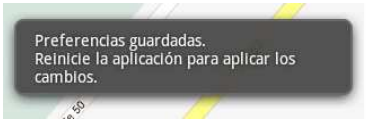
```
21                      android:key="iconSizePref" android:defaultValue="2"
android:entries="@array/iconSize"
22                      android:entryValues="@array/iconSize_values"
android:persistent="true" />
23      </PreferenceCategory>
24      ...
25 </PreferenceScreen>
```

Este XML, luego se definirá como vista de una actividad que extienda PreferenceActivity a través de la llamada a `addPreferencesFromResource` en nuestro caso, la actividad Config.

Config.java

```
1 public class Config extends PreferenceActivity {
2       static String TAG = Config.class.getName();
3
4       @Override
5       public void onCreate(Bundle savedInstanceState) {
6               super.onCreate(savedInstanceState);
7               addPreferencesFromResource(R.layout.preferences);
8       }
9 }
```

Y eso es todo. La plataforma se encarga automáticamente del salvado de las preferencias del usuario. La edición de las preferencias se hace *in line*, sin necesidad de guardar los cambios a través de algún comando del tipo Guardar. Simplemente saliendo de la actividad mediante el botón back de nuestro dispositivo se salvan los cambios. Por esto, hemos agregado un aviso a nuestro prototipo, cuando al salir de la actividad de configuración nos de una aviso.



Las propiedades que define el usuario se recuperan a través de una instancia de `SharedPreferences`.

Para simplificar el uso y lectura de las preferencias, creamos una clase `ARDroidPreferences` que implementa el patrón de diseño *Singleton*(*), para minimizar las lecturas de las configuraciones de usuario y obtener las configuraciones necesarias en los distintos lugares del prototipo de una manera más fácil.

```
 1 public class ARDroidPreferences {
 2       private static SharedPreferences preferences = null;
 3
 4       public static SharedPreferences createPreferences(Context context) {
 5               if (preferences == null) {
 6                       preferences =
PreferenceManager.getDefaultSharedPreferences(context);
 7               }
 8               return preferences;
 9       }
10
11       private static SharedPreferences getPreferences() {
12               return preferences;
13       }
14
15       public static String getString(String key, String defValue) {
16               if (ARDroidPreferences.getPreferences() == null)
17                       return defValue;
18               return ARDroidPreferences.getPreferences().getString(key, defValue);
19       }
20       ...
21 }
```

Cuando necesitamos una propiedad para condicionar alguna acción o vista del prototipo, utilizamos

la siguiente línea de código:

```
ARDroidPreferences.getBool(”oriMapPref”, true)
```

El objeto `SharedPreferences` nos brinda distintos métodos para obtener las distintas propiedades según su tipo: `getString`, `getBool`, `getFloat`, `getInt`. Utilizamos `getBool` cuando queremos leer una propiedad de tipo `CheckBoxPreference`. Utilizamos `getFloat` cuando queremos leer una propiedad de tipo `EditTextPreference android:inputType="number"`.

7.9.1 - URL del servidor

Nombramos una de las opciones más importantes con las que cuenta el prototipo, que es la *URL Servidor*. Aquí podemos configurar a que servidor apuntar para obtener los datos. Esto nos brinda la posibilidad de utilizar diferentes servidores de datos, dándole independencia al cliente del servidor.

Con un mismo cliente podemos utilizar la aplicación para conectarnos a un servidor que nos de información de entidades y eventos del ámbito académico, pero un cambio de servidor, podemos conectarnos a uno que nos brinde información de entidades y eventos

Por supuesto, el servidor que se utilice debe respetar el formato del modelo de datos de AR-DROID, respondiendo a todos lo eventos de interacción del modo REST que definimos en el capitulo anterior.

7.10 - FUNCIONALIDADES ADICIONALES

Mostramos aquí otras funcionalidades que hemos incluido en la aplicación o que por su simplicidad no detallaremos demasiado.

Por supuesto que no son las únicas, el prototipo cuanta con muchos y variados piezas de código que en sincronía cumplen con todas las funcionalidades implementadas, pero como dijimos al principio de este capitulo, la finalidad de el mismo era mostrar las capacidades de la plataforma mostrando pequeños fragmentos codificados.

7.10.1 - SplashScreen de la aplicación

Para evitar la pantalla negra que da la aplicación al inicio, donde principalmente se tiene una demora en la inicialización del mapa y captura inicial de las entidades desde el servidor, implementamos una actividad a modo de pantalla de bienvenida.

La finalidad de la misma es mostrar una imagen y presentación de la aplicación mientras de fondo se recuperan las entidades a mostrar.

SplashScreen.java

```
 1 public class SplashScreen extends Activity {
 2       ...
 3       public void onCreate(Bundle savedInstanceState) {
 4                ...
 5                // crear hilo para mostrar el splash screen
 6                Thread splashTread = new Thread() {
 7                        @Override
 8                        public void run() {
 9                                try {
10                                        int waited = 0;
11                                        while ((waited < _splashTime) ||
_waitLoaded) {
12                                                sleep(100);
13                                                waited += 100;
14                                        }
15                                }
16                                finally {
17                                        Intent myIntent = new
Intent(SplashScreen.this, MainMap.class);
18                                        SplashScreen.this.startActivity(myIntent);
19                                }
20                        }
21                };
22                splashTread.start();
23                try {
24                        Resource.getInstance().getEntities();
25                        _waitLoaded = false;
26                } catch (Exception e) {
27                        showError();
28                }
29       }
30
31       ...
32 }
```

Cumplimos esta tarea a través de un `Thread` (líneas 6-21) que empieza a correr previamente a la llamada a `Resource.getInstance().getEntities()` (línea 24), responsable de recuperar y almacenar localmente las entidades. Una vez recuperadas las entidades, seteamos el *flag* `_waitLoaded` (línea 25) para que continúe la carda de la aplicación y se inicie la actividad principal.

El resultado hace la carga del prototipo más vistosa de cara al usuario.

También cumple con la función de informar si no se puede conectar al servidor e iniciar la pantalla de preferencias de la aplicación para comprobar la URL del servidor.

7.10.2 - Sonidos de la aplicación

A modo de mostrar las varias funcionalidades de la plataforma, hemos decidido incluir sonidos al prototipo. Estos sonidos se reproducen al interactuar con diálogos y botones en las diferentes actividades de la aplicación.

Para simplificar las llamadas, creamos una clase para delegar la reproducción de un sonido, que se encarga de leerlo desde los recursos de la aplicación, y reproducirlo.

SoundManager.java

```
 1 public class SoundManager {
 2       private static boolean soundEnable = AppPreferences.getBool(
 3                         "soundEnable", false);
 4       private static Context context;
 5
 6       public static void createInstance(Context cxt) {
 7                context = cxt;
 8       }
 9
10       public static void playSound(final int resource) {
11                if (!soundEnable)
12                         return;
13
14                Thread runnable = new Thread() {
15                         @Override
16                         public void run() {
17                                  MediaPlayer.create(context, resource).start();
18                         }
19                };
20                runnable.start();
21       }
22 }
```

La plataforma no brinda una clase `MediaPlayer`, con el que podemos reproducir sonidos con solo una línea de código (ver línea 17). Vemos el sonido se reproduce en un `Thread` para no demorar las operaciones mientras se suena el sonido.

La utilización de esta clase la hacemos en distintas actividades con esta simple llamada a `SoundManager`:

```
SoundManager.playSound(R.raw.message);
```

En nuestro caso utilizamos sonidos OGG(*), pero la plataforma es compatible con otros muchos

formatos. Estos se deben guardar en la carpeta */res/raw* de nuestro proyecto Android. De allí se auto compila y se generar automáticamente los identificadores en la clase R.

7.10.3 - Visualizar dirección actual

En la vista de mapa como así también en RA mostramos en la esquina superior derecha un texto que nos indica nuestra dirección. Esto, como dijimos al inicio de este capítulo, lo hacemos determinando la a través del API *Geocoder*.

LocationListenerGPS.java

```
 1 public class LocationListenerGPS implements LocationListener {
 2       ...
 3       protected void showAddress() {
 4                if (lastKnowLocation == null)
 5                        return;
 6                // obtener ubicación
 7                String addressName = "";
 8                try {
 9                        Geocoder geocoder = new Geocoder(this.getContext(),
Locale.getDefault());
10                        List<Address> lsAddress =
geocoder.getFromLocation(lastKnowLocation.getLatitude(),
11                                        lastKnowLocation.getLongitude(), 1);
12                        if (lsAddress.size() > 0) {
13                                Address address = lsAddress.get(0);
14                                if (address.getAddressLine(0) != null)
15                                        addressName = address.getAddressLine(0);
16                                else if (address.getThoroughfare() != null)
17                                        addressName = address.getThoroughfare();
18                                else if (address.getFeatureName() != null)
19                                        addressName = address.getFeatureName();
20                                else if (address.getLocality() != null)
21                                        addressName = address.getLocality();
22                                else
23                                        addressName = "Ubicación desconocida";
24                        }
25                } catch (Exception e) {
26                        Log.d(TAG, "Error obteniendo ubicación", e);
27                        addressName = "Ubicación desconocida";
28                }
29                // armar texto
30                TextView textView = (TextView) activity.findViewById(R.id.address);
31                textView.setText(addressName);
32                textView.invalidate();
33       }
34 }
```

Conociendo la ubicación actual del usuario en un objeto `Location`, creamos un objeto `Geocoder` y obtener la información de la ubicación actual. Este objeto tiene varias propiedades de la misma, como calle, numero, localidad, punto de interés, etc.

Entre las líneas 13-23 obtenemos una de estos textos por prioridad, sabiendo que es más exacto mostrar calle y numero, pero no siempre están disponibles. Por último, obtenemos el objeto `TextView` de la actividad e imprimimos el texto descriptivo que obtuvimos desde la el servicio de geolocalización.

Este mecanismo se repite cada vez que se recibe un nuevo punto de localización desde el dispositivo GPS.

8 - CONCLUSIONES Y TRABAJO FUTURO

Daremos ahora un repaso general del proyecto, presentando nuestras conclusiones finales en función de los resultados obtenidos y señalaremos posibles trabajos futuros que puedan presentarse como consecuencia de este.

8.1 - Conclusiones

Para verificar que el proyecto ha obtenido los resultados esperados, basta con comprobar que se han alcanzado los objetivos que se pautaron al inicio del proyecto.

Hemos introducido fundamentos teóricos de los temas necesarios para la implementación del prototipo inicialmente propuesto. Como pudo apreciarse, el marco teórico estuvo bien identificado y tratamos de centramos en conceptos introductorios a los diferentes temas, seguido después un hilo de descripción general de cada uno de estos.

Resumiremos las conclusiones de cada uno, haciendo hincapié en lo que estas tecnologías podrán hacer en el futuro.

Las capacidades actuales de los móviles explican el fenómeno de hoy en día. La oferta de smartphones es cada vez más grande. Los móviles modernos tienen capacidades que los hacen ideales para desarrollos de de aplicaciones interactivas. La llegada de sensores como GPS, brújulas electrónicas, banda ancha y la alta capacidad computacional, han convertido a los celulares en el perfecto campo de juego para las aplicaciones de geolocalización y RA.

A estas capacidades las fusionamos sobre un sistema operativo de evolución, como es Android, y la combinación resultó apasionante.

En Android nos encontramos con una plataforma de código abierto creada por "la" firma de Internet, Google. A solo cuatro años de ver la luz, se ha confirmado el éxito de Android en los móviles de Google y de de otros fabricantes que han conseguido que la plataforma de Google sea una de las más usadas, hasta codearse con el rey de los dispositivos móviles, iPhone. Esto significa que el "robot verde" es un rival de peso para Apple.

Google Android es un desarrollo emocionante en el mundo de los sistemas operativos móviles. No es tanto por lo que hace por si mismo, sino también por lo que los desarrolladores del mundo pueden crear cuando tienen el código en sus manos.

Android incluye todo el software para que funcione un teléfono móvil, pero sin los obstáculos propietarios que han impedido la innovación en el teléfono móvil. El resultado en última instancia será un ritmo de innovación más rápido y mejor que proporcionará a los clientes móviles aplicaciones y capacidades hasta ahora difíciles de conseguir. Android es una estrategia importante para acercarnos al objetivo de proporcionar acceso a la información a los usuarios dondequiera que estén.

En los últimos años ha crecido en número y funcionalidad el desarrollo de aplicaciones y servicios basados en LBS. Hoy debemos ver a los prestaciones LBS no sólo como una categoría en si misma, sino considerar que brinda valor agregado a los servicios existentes, incrementando la utilidad y valor para sus usuarios. Debemos partir de las preferencias y hábitos que hoy tiene los usuarios y a esto aplicarle las tecnologías existentes para dar nuevos usos.

Nuestro prototipo permite servir información geolocalizada y catalogada. Brinda mecanismos de integración con redes sociales y un fácil acceso de adaptación para información fidedigna, provenientes de fuentes no estandarizadas, como sitios Web a medida, y blogs anticuados.

A esto le sumamos elementos de realidad aumentada, y si bien el resultado ha sido algo pobre, ya que algunas de las decisiones a la hora de implementación no fueron de lo más acertadas, conocimos las ventajas de su aplicación. Revelamos una ola de innovación, que permite a los usuarios interactuar virtualmente con su entorno, aunque sigue siendo una tecnología incipiente.

El potencial a futuro de la realidad aumentada es muy grande. No solo por su funcionalidad, sino más bien por su capacidad de mejorar la experiencia sobre otras tecnologías actuales (y futuras).

Finalizamos esta conclusión mencionado algunos de los problemas que se nos presentaron durante el desarrollo del mismo.

El objetivo de crear un prototipo funcional en el transcurso de este trabajo, provocó la poca profundización en alguno de los métodos que se estábamos utilizando, por ejemplo en la localización, o la representación gráfica, lo que provocó la reescritura de muchas partes del código en revisiones

posteriores.

Otra de las grandes dificultades que hemos encontrado esta en habernos enfrentado a tecnologías novedosas, en algunos casos con poca maduración. Como ejemplo de esta podemos nombrar la Realidad Aumentada.

Sin embargo, estamos seguros que el resultado ha sido muy positivo. A pesar de las ansias en la funcionalidad del prototipo y de unas expectativas iniciales quizás demasiados optimistas, creemos que se pueden extraer lecciones muy valiosas de lo aprendido de este proyecto.

8.2 - Resultados obtenidos

A lo largo de este trabajo de investigación hemos expuesto distintos conceptos y temas relacionados a aplicaciones móviles, servicios basados en localización y realidad aumentada. También hemos realizado una introducción a las principales características de la plataforma Android, sus APIs y entorno de desarrollo principal, para luego concluir con el desarrollo de un prototipo de aplicación para dispositivos móviles llamado AR-DROID, que aplica al ámbito académico.

El prototipo cuenta con un administrador Web que forma parte del backend de la aplicación, actuando como parte servidor de la arquitectura. Está desarrollado en Java sobre el Framework Groovy on Grails. El administrador Web permite registrar los distintos puntos de interés (entidades, eventos y actividades) así como también ver estadísticas sobre los eventos registrados, con la finalidad de tener conocimiento de las visitas y comentarios que los usuarios realizan sobre los eventos registrados.

AR-DROID permite ver información geolocalizada (entidades y eventos) utilizando la vista de mapas en la cual utilizamos Google Maps, o a través de la vista de de Realidad Aumentada que hace uso de la cámara del dispositivo móvil. El usuario puede consultar individualmente la información completa de las entidades y eventos registrados, con la posibilidad de interactuar con el prototipo a modo de red social, completando encuestas manifestando su opinión sobre los eventos. AR-DROID también permite visualizar noticias o últimas novedades sobre las entidades ya que integra diferentes canales de información o redes sociales (RSS, Facebook, Twitter, etc).

En AR-DROID se utilizan diferentes componentes básicos de la plataforma Android, como son las actividades, mapas o menús. Hicimos uso de gestores que controlan elementos del dispositivo móvil como el GPS, esencial para conocer la ubicación del usuario, se captura datos de sensores (acelerómetro, compás) y se hace uso de la cámara del dispositivo para captura de video. Además, se trabajo con interfaces de usuarios, conexiones HTTP, declaración de manifiesto de aplicación y un manejo de recursos tanto internos como remotos.

En conclusión, hemos desarrollado un prototipo estable y funcional. AR-DROID no es dependiente del negocio, por lo que podría adaptarse fácilmente a otro ámbito que no sea el universitario.

8.3 - Trabajo futuro

Como hemos dicho a lo largo de este trabajo, el resultado obtenido es un prototipo de aplicación, un punto de partida para crear sistemas completos con funcionalidades que utilicen geolocalización de recursos. Quedan muchas mejoras que se pueden hacer sobre este prototipo, a conocer desde el la aceptación de los usuarios y necesidades de negocio especificas.

La implementación de esta primera versión del prototipo ha dejado ver puntos débiles que deberían ser mejorados o añadidos que generarían nuevos aspectos funcionales del mismo:

- Refactorización general del código: Hemos puesto empeño en la desarrollar funcionalidades en el prototipo dejando de lado cuestiones estéticas de código fuente. Algunas implementaciones carecen de buenas practicas de desarrollo, y puede que afectar la performance de la aplicación.
- Más fuentes de alimentación de eventos: Es posible, por la interfaz que hemos implementado, agregar mas fuentes de alimentación de eventos para las entidades, implementado una interfaz definida. Dichas implementaciones podrían incluir productos como Google Calendar, o Google+.
- Mejorar el administrador Web: La carga de entidades y eventos desde el administrador Web ha mostrado una utilización fácil e intuitiva, pero no así la confección de encuestas.
- Acoplar distintos elementos de feedback: El prototipo final presentado incluye solo un tipo de feedback para los eventos. Lo ideal para obtener una opinión confiable para un evento,

consistiría en permitir armar encuestas de opiniones más significativas, que incluyan cuestionarios con varias preguntas de distintos tipos.

- Elementos sensibles al contexto: El prototipo incluye información geolocalizada de puntos de interés, pero no reacciona cuando un usuario esta en uno de esos puntos o cerca de el. Podría incluirse algún tipo de funcionalidad que indique al usuario que está sucediendo cerca de su ubicación, o consultar la agenda de actividades del usuario para advertirle a través de un recordatorio de las mismas al acercase al lugar.
  El teléfono, con su compás y su GPS, sabe donde estamos y qué estamos mirando, por lo que podríamos decir cualquier cosa antes de que se el usuario interactué con la aplicación.

El crecimiento que están teniendo las tecnologías de geolocalización y realidad aumentada en el mercado de las aplicaciones móviles ha dado lugar al nacimiento de *startups* y empresas de desarrollo. La proliferación de eventos sobre geolocalización muestra el interés que despiertan estas tecnologías en todo el mundo. Es por esto que este tipo de aplicaciones se perfilan como promotores del marketing móvil.

Su aplicación cubre varios sectores, tanto públicos como empresariales. Podría pensarse en aplicaciones de geolocalización de obras públicas, que permitan fomentar lugares de esparcimiento, patrocinados por unidades gubernamentales. También puede presentarse como una alternativa a un mapa de ruta de una empresa de turismo, asistiendo al turista sobre lugares turísticos, hospedajes, excursiones, con la posibilidad de extraer datos cualitativos a partir de la opinión de los turistas. Otro gran sector es el inmobiliario, donde el éxito está asegurado para aquella empresa que localice sus propiedades en un catálogo móvil geolocalizado.

Las grandes empresas están empezando a mostrar sus productos y centros de atención mediante aplicaciones móviles, esto les da reputación y publicidad por lo novedoso y el interés que este tipo de aplicaciones todavía genera entre los usuarios. De seguro, apostamos que nos espera un futuro con nuevas e interesantes oportunidades en el mundo móvil.

REFERENCIAS BIBLIOGRÁFICAS

[1] GUY, Retta. Mobile Learning: Pilot Projects and Initiatives. 1ª Ed. US, Informing Science Press, 2010.
[2] FLING, Brian. Mobile Design and Development. 1ª Ed.US, O'Reilly Media Inc., 2009. Cap 2.
[3] MOBILEN 50AR. Facts about the Mobile. A Journey through Time. [en línea] <http://www.mobilen50ar.se/eng/FaktabladENGFinal.pdf>.
[4] LISTER, John y HARRIS, Bronwyn. What Is 4G Mobile Technology? [en línea] <http://www.wisegeek.com/what-is-4g-mobile-technology.htm>.
[5] FORMAN, G. y ZAHARJON, J. The challenges of mobile computing. IEEE Computing, Vol. 27, No. 4.
[6] Ref [2], Cap 5.
[7] AZUMA, Ronald, C. A Survey of Augmented Reality. Hughes Research Laboratories. Hughes Research Laboratories, 1997.
[8] MILGRAM, Paul, KISHINO, Fumio. Augmented Reality: A class of displays on the reality-virtuality continuum. ATR Communication Systems Research Laboratories, 1994.
[9] Gartner, Inc. (NYSE: IT) [en línea] <http://www.gartner.com/technology/about.jsp>.
[10] BRUGGE, B. y REICHER, R. Distributed user tracking concepts for augmented reality applications. TU Munich, Dept. of Computer Science, 2002.
[11] CHOUDARY, O. y CHARVILLAT, V. MARCH: Mobile Augmented Reality for Cultural Heritage. IRIT, University of Toulouse, 2009.
[12] LORENSEN, W. y CLINE, H. Enhancing Reality in the Operating Room. GE Corporate Research and Development, 2003.
[13] KIYOKAWA, K. An introduction to head mounted displays for augmented reality. Colorado School of Mines, 2007.
[14] SUTHERLAND, I. A Head-Mounted Three Dimensional Display. ACM, 1968.
[15] CAUDELL, T. y MIZELL, D. Augmented reality: an application of heads-up display technology to manual manufacturing processes. Boeing Comput. Services, 1992.
[16] FEINER, S. y HÖLLERER, T. Exploring MARS: Developing Indoor and Outdoor User Interfaces to a Mobile Augmented Reality System. Columbia University, New Yor, 1997.
[17] KHARMA: A KML/HTML Architecture for Mobile Augmented Reality Applications [en línea] <https://research.cc.gatech.edu/polaris/content/home>.
[18] Jochen Schiller, Agne`s Voisard. Location-Based Services. 2004 by Elsevier Inc.
[19] Ian Koeppel. The definition of location services (2002), what is location services from a GIS perspective, ESRI
[20] Kirsi Virrantaus. Developing GIS-Supported Location-Based Services.[en línea] <http://www.cs.jyu.fi/ai/papers/WGIS-01.pdf>.
[21] Tumasch Reichenbacher. Tesis. Mobile Cartography - Adaptive Visualisation of Geographic Information on Mobile Devices. 2004. Technical University, Munich. [en línea] <http://tumb1.biblio.tu-muenchen.de/publ/diss/bv/2004/reichenbacher.pdf>.
[22] Funcation of LBS. [en línea] <http://www.e-cartouche.ch/content_reg/cartouche/LBSbasics/en/html/index.html>.
[23] Allan Brimicombe, Chao Li. Location-Based Services and Geo-Information Engineering. 2009 by John Wiley & Sons Ltd.
[24] Shu Wang, Jungwon Min and Byung K. Yi. Location Based Services for Mobiles: Technologies and Standards, IEEE International Conference on Communication (ICC) 2008, Beijing, China.
[25] IETF - The Internet Engineering Task Force. [en línea] <http://www.ietf.org/>.
[26] 3GPP -3rd Generation Partnership Project. [en línea] <http://www.3gpp.org/>.
[27] 3GPP2 - 3rd Generation Partnership Project 2. [en línea] <http://www.3gpp2.org/>.
[28] OMA - Open Mobile Alliance. [en línea] <http://www.openmobilealliance.org/>.
[29] WI-FI Positioning System. [en línea] <http://www.avesnocturnas.es/2009/12/%C2%BFcomo-funciona-el-posicionamiento-por-wifi/>.
[30] Introducción A Android. [en línea] <http://developer.android.com/index.html>.
[31] Android Open Handset Alliance. [en línea] <http://www.openhandsetalliance.com/android_overview.html>.
[32] Historia de Android. [en línea] < http://materiageek.com/2009/11/android-cumple-2-anos-breve-recorrido-por-su-corta-historia/>.

[33] Android: The Open Source Cell Phone. [en línea] <http://google-opensource.blogspot.com/2008/10/android-open-source-cell-phone.html>.
[34] Condiciones de la Licencia Apache versión 2.0. [en línea] <://www.apache.org/licenses/LICENSE-2.0.html>.
[35] Android. What is?. [en línea] <http://developer.android.com/guide/basics/what-is-android.html>.
[36] Android Historia. [en línea] <http://androidzone.org/especial-por-los-3-anos-de-android-un-poco-de-historia/]>.
[37] Professional Android Application Development. Wiley Publishing, Inc. 2009
[38] Android. Using DDMS. [en línea] <http://developer.android.com/guide/developing/debugging/ddms.html>.
[39] Android Debug Bridge. [en línea] <http://developer.android.com/guide/developing/tools/adb.html>.
[40] Android ADT plugin. [en línea] <http://developer.android.com/sdk/eclipse-adt.html>.
[41] Android Managing Virtual Devices. [en línea] <http://developer.android.com/guide/developing/devices/index.html>.
[42] William Brogden, "REST versus SOAP – the REST Story" [en línea] <http://searchsoa.techtarget.com/tutorial/Representational-State-Transfer-REST-Tutorial>.
[43] Roger L. Costello, "Building Web Services the REST Way". xFront website. [en línea] <http://www.xfront.com/REST-Web-Services.html>.
[44] Web services. [en línea] <http://www.w3.org/TR/ws-arch/#whatis>.
[45] implementación de Web services REST en Grails. [en línea] <http://grails.org/doc/1.0.x/guide/13.%20Web%20Services.html>.
[46] Introducing JSON. <http://www.json.org/>.
[47] JSON: The Fat-Free Alternative to XML. [en línea] <http://www.json.org/xml.html>.
[48] ADC: Android Developer Challenge [en línea] <http://code.google.com/intl/es-ES/android/adc/>.
[49] Geocoder API by Google [en línea] <http://code.google.com/intl/en/apis/maps/documentation/geocoding/>.
[50] google-gson: Librería de conversión JSON a string y viceversa de Google. [en línea] <http://code.google.com/p/google-gson/>.
[51] Google API Directions: API de geocodificación de direcciones y POIs [en línea] <http://code.google.com/intl/en/apis/maps/documentation/directions/>.

GLOSARIO

- Smartphone: Teléfono móvil que ofrece capacidades de computación y conectividad más avanzada y características contemporáneas.
- GSM: Sistema global para las comunicaciones móviles.
- CDMA (code division multiple access o acceso múltiple por división de código): Permite que múltiples terminales compartan el mismo canal de frecuencia, identificándose el "canal" de cada usuario mediante (secuencias PN).
- CDMA2000: Estándares de telecomunicaciones móviles de tercera generación (3G) que utilizan CDMA.
- GPS: El Sistema de Posicionamiento Global es una zona basada en el sistema mundial de navegación por satélite (GNSS) que proporciona información sobre ubicación y hora cerca de la Tierra, donde hay una línea de visión sin obstáculos a tres o más satélites.
- NMT: NMT (Telefonía Móvil Nórdica, en Inglés) es el primer sistema de telefonía celular totalmente automático.
- AMPS: "Advanced Mobile Phone System" fue el análogo estándar de sistema de teléfono móvil desarrollado por Bell Labs, y presentado oficialmente en América en1983.
- SMS: Servicio disponible en los teléfonos móviles que permite el envío de mensajes cortos entre teléfonos móviles, teléfonos fijos y otros dispositivos de mano.
- TDMA: La multiplexación por división de tiempo (TDM) es una técnica que permite la transmisión de señales digitales y cuya idea consiste en ocupar un canal (normalmente de gran capacidad) de transmisión a partir de distintas fuentes, de esta manera se logra un mejor aprovechamiento del medio de transmisión.
- CDMA: La multiplexación por división de código, acceso múltiple por división de código o CDMA (del inglés Code Division Multiple Access) es un término genérico para varios métodos de multiplexación o control de acceso al medio basado en la tecnología de espectro expandido.
- 3G: Es la abreviación de tercera-generación de transmisión de voz y datos a través de telefonía móvil.
- GPRS: General Packet Radio Service (GPRS) o servicio general de paquetes vía radio es una extensión de GSM para la transmisión de datos no conmutada (o por paquetes).
- EDGE: Es una tecnología de la telefonía móvil celular, que actúa como puente entre las redes 2G y 3G. EDGE se considera una evolución del GPRS.
- 3GPP: 3rd Generation Partnership Project es un acuerdo de colaboración en tecnología de telefonía móvil que fue establecido en diciembre de 1998. Esta cooperación es entre Europa, Japon, China, América del Norte y Corea del Sur.
- Qualcomm: Compañía estadounidense fundada en 1985 que produce los chipsets para la tecnología móvil CDMA.
- SDK: Un kit de desarrollo de software (software development kit) es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.
- Open Handset Alliance: OHA es una alianza comercial de 78 compañías para desarrollar estándares abiertos para dispositivos móviles. Algunos miembros son Google, HTC, Dell, Intel, Motorola, entre otros.
- Framework: Estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado
- Drill-down: Refiere a cualquiera de las diversas operaciones y transformaciones relacional y multidimensional de datos tabulares.
- Widget: Un widget es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine.
- Código QR: Un código QR (Quick Response Barcode) es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensional.
- algoritmos para tratamiento de imágenes: Son algoritmos utilizados en captura por computador para extraer ciertos tipos de características y deducir el contenido de una imagen.

- A-GPS: A-GPS fue desarrollado e introducido para mejorar el funcionamiento del sistema GPS, y se suele usar en teléfonos y dispositivos móviles tipo PDA.
- 802.11b: Es una modificación de la Norma IEEE 802.11 que amplia la tasa de transferencia hasta los 11 Mbit/s usando la misma banda de 2.4 GHz.
- RFID: Sistema de almacenamiento y recuperación de datos remotos, que utiliza dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID.
- GPU: La unidad de procesamiento gráfico es un procesador dedicado al procesamiento de gráficos u operaciones de coma flotante.
- UTF-8: Es un formato de codificación de caracteres Unicode e ISO 10646 utilizando símbolos de longitud variable.
- CDATA: Es un componente de XML, que permite introducir caracteres especiales sin que sean interpretados por el navegador como código ejecutable.
- GEOSpot: (puntos geo-posicionados) son puntos identificables sobre un mapa.
- SOAP: SOAP es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.
- Geolocalización: Conocer la ubicación geográfica de un objeto o persona.
- Geotagging: Se llama al proceso de agregar información geográfica en los metadatos de archivos de imágenes, vídeos, sonido, sitios web, etc. que sirva para su geolocalización, por lo general estos datos suelen ser coordenadas que definen la longitud y latitud.
- RFC: Son una serie de notas sobre Internet que comenzaron a publicarse en 1969.
- UMTS: Es una de las tecnologías usadas por los móviles de tercera generación (3G, también llamado W-CDMA), sucesora de GSM.
- RSS: Es una familia de formatos de fuentes Web utilizada para publicar información actualizada en un formato estandarizado.
- API: Conjunto de especificaciones que los programas de software siguen para comunicarse entre sí. Define una interfaz entre programas y facilita su interacción.
- Scaffolding: Técnica de apoyo de algunos MVC, en los que el programador puede escribir una especificación que describe cómo la base de datos es utilizada por la aplicación.
- MVC: Modelo-vista-controlador es una arquitectura de software, que actualmente se considera un patrón arquitectónico usado en ingeniería de software.
- HSQLDB: Sistema de gestión de bases de datos relacionales escrito en Java.
- XML: Conjunto de normas de codificación de documentos en forma legible por máquina.
- Timestamp: Marca de tiempo en una secuencia de caracteres, que indica la fecha y/o el tiempo en que ocurre un determinado evento.
- JSON: (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos.
- Bitmap: Formato de imagen que organiza los bits sobre un mapa, casi sin compresión.
- Canvas: Un Canvas es un control apropiado para dibujar en su superficie.
- double dispatching: (doble envío) es una forma espacial de envío multiple en una llamada a una pieza de software.
- JavaScript: JavaScript es un prototipo basado en el lenguaje de scripting que es dinámico, con tipos débiles y tiene funciones de primera clase.
- Feedback: Feedback describe información sobre una acción que vuelve a la fuente de esas acciones.
- best practices: Son métodos o técnicas que han demostrado consistentemente resultados superiores a los obtenidos con otros medios, y que se utilizan como punto de referencia.
- Singleton: Es un patrón de diseño utilizado para implementar el concepto matemático de un "producto único", al restringir la creación de instancias de una clase a un objeto.
- OGG: Es un formato contendor multimedia, de código libre, mantenido por la Fundación Xiph.Org.

Apéndice A - PREPARACIÓN DEL AMBIENTE DE DESARROLLO

Como podemos deducir de capítulos anteriores, las tecnologías utilizadas para llevar a cabo el prototipo pueden resumirse en la siguiente lista:

- Java Development Kit
- Eclipse
- SpringSource Tool Suite (STS)
- Grails y Groovy on Grails
- Apache Tomcat
- Android SDK
- Plugin SVN
- Maven
- MySQL

En este anexo detallaremos los pasos necesarios para instalar las herramientas necesarias y montar un ambiente de desarrollo para ejecutar el prototipo.

Instalación de Java Development Kit

Primeramente, por tratarse de un proyecto Java, es imprescindible instalar el JDK. Podemos obtenerlo desde http://www.oracle.com/technetwork/java/javase/downloads/index.html. En nuestro caso, la versión utilizada es la más reciente hasta el momento, *JDK 6 Update 24*.

Instalación de Eclipse

Con la JDK instalada, ya podemos correr el ambiente de desarrollo Eclipse. Por compatibilidad de plugins, que necesitaremos más adelante, utilizamos la versión 3.5 SR2 (alias *Galileo*), en su versión *Eclipse IDE for Java EE Developers*. La instalación de Eclipse se resume únicamente a descomprimir el archivo descargado.

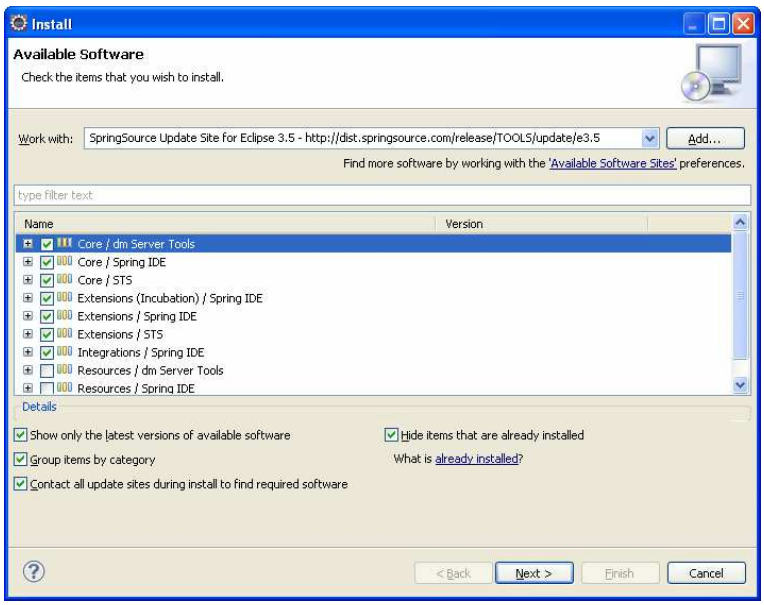
Instalación de SpringSource Tool Suite (STS)

Con el Eclipse en marcha, es hora de instalar algunos plugins que necesitaremos para dar soporte a la funcionalidad a implementar. Unos de los primeros es STS. Para esto necesitaremos agregar algunos *Software Sites* que contienen los plugin que instalaremos[1]. Primeramente agregaremos estos sitios, para esto, nos dirigimos a *Window -> Preferences -> Install/Update -> Available Software Sites*. Con el botón *Add…* agregamos los siguientes sitios:

Name: SpringSource Update Site for Eclipse 3.5 (Release)
Location: http://dist.springsource.com/release/TOOLS/update/e3.5

Name: SpringSource Update Site for Eclipse 3.5 (Release + Dependencies)
Location: http://dist.springsource.com/release/TOOLS/composite/e3.5

Agregados los sitios nos dirigimos a *Help -> Install New Software*, y seleccionamos del combo *Working with* el primer sitio agregado, SpringSource Update Site for Eclipse 3.5 (Release). Seleccionamos los primeros siete componentes como muestra la siguiente figura:

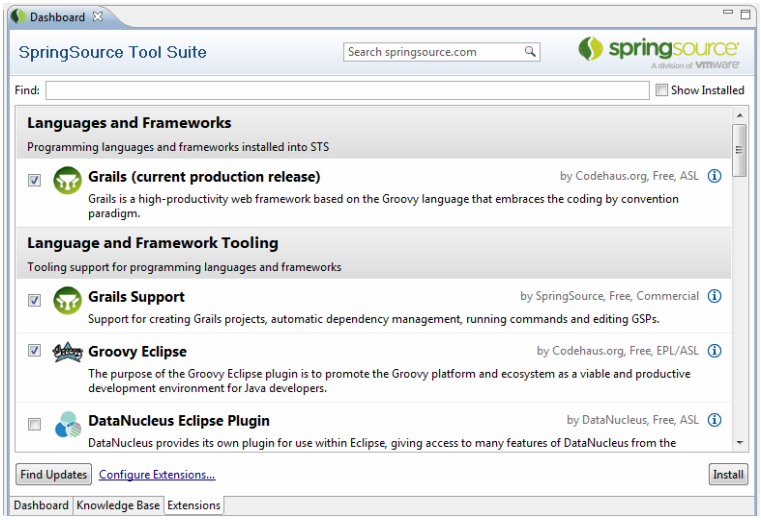


Al terminar de descargar e instalar los nuevos componentes nos solicitara reiniciar nuestro Eclipse para aplicar los cambios, lo permitimos.

Instalación de Grails y Groovy on Grails

Instalador el plugin STS, agregaremos un par de extensiones a este: soporte para Grails, y plugin para Groovy. Para esto, nos dirigimos a *Help -> Dashboard* y seleccionamos la solapa *Extensons*. Veremos la lista de extensiones para el plugin STS. Seleccionaremos de la sección *Lenguaje and Framwork Tooling* tres extensiones: *Grails (current production release)*, *Grails Support* y *Groovy Eclipse*.

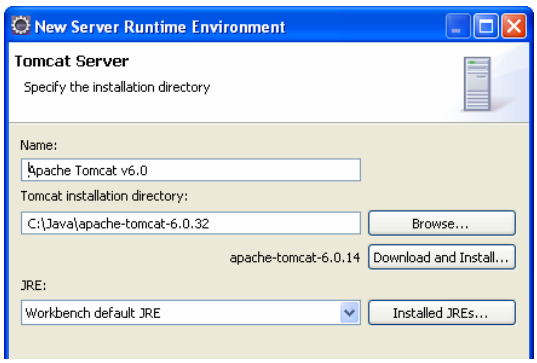
Seleccionaremos estas tres de la lista de extensiones y a continuación presionamos el botón *Install*. Al terminar de descargar e instalar los nuevos componentes nos solicitara reiniciar nuestro Eclipse para aplicar los cambios, lo permitimos.



Instalación de Apache Tomcat

Utilizaremos Apache Tomcat como contenedor de aplicaciones para nuestra aplicación Web. Primeramente descargamos la versión 6.0 de Tomcat desde http://tomcat.apache.org/download-60.cgi. Desde la sección *Binary Distributions* descargamos la versión comprimida del *Core*.

Descomprimimos el archivo descargado en una ubicación a elección, suponiéndola "C:\Java\apache-tomcat-6.0.32". Luego debemos configurar nuestro eclipse para saber donde ubicar la instalación de Tomcat. Para esto nos dirigimos a *Window -> Preferences -> Server -> Runtime Environment* y agregamos un servidor con el botón *Add*. Seleccionamos *Apache Tomcat v6.0* y completamos los parámetros como se muestra en el cuadro a continuación:



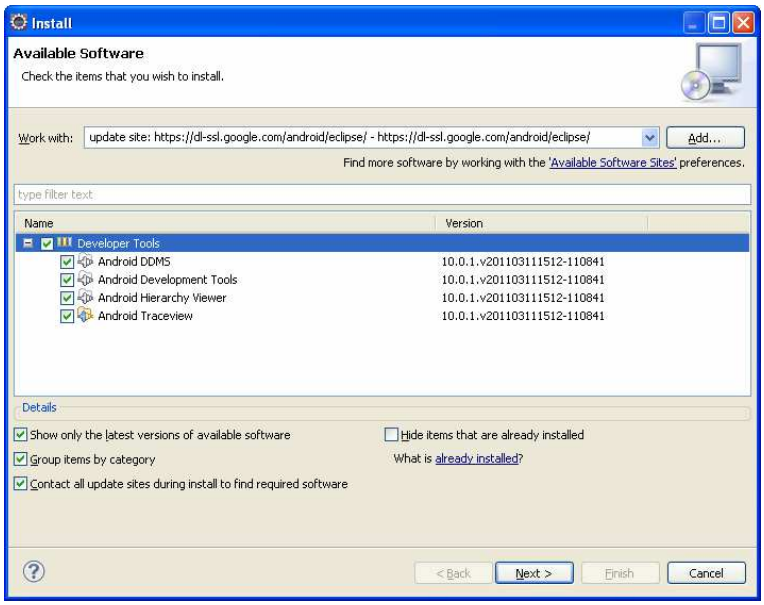
Instalación de Android SDK

Para instalar y crear proyectos Android con Eclipse, necesitamos Android SDK. Descargaremos desde http://developer.android.com/sdk, el archivo android-sdk_r10-windows.zip. Descomprimimos el archivo descargado en una ubicación a elección, suponiéndola "C:\Java\android-sdk-windows".

Seguidamente instalaremos el plugin ADT Plugin[2] para Eclipse, con el cual podremos trabajar con Android desde Eclipse. Para esto, agregaremos un sitio a nuestro *Available Software Sites*:

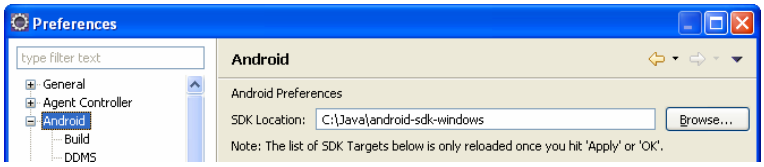
Name: Android Development Toolkit Plugin

Location: https://dl-ssl.google.com/android/eclipse/

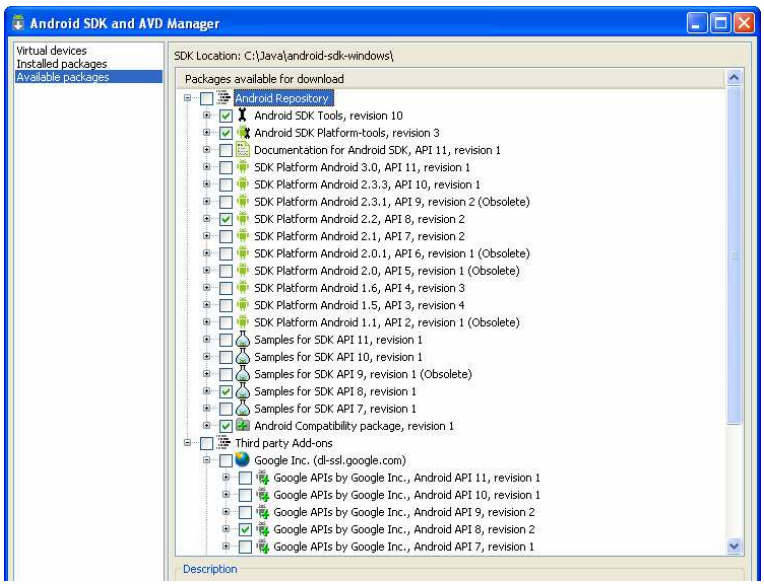


Seleccionamos los componentes tal como muestra la imagen anterior y proseguimos con la instalación.

Una vez instalado y reiniciado el Eclipse, es momento de configurar el plugin ADT. Para hacer esto nos dirigimos a *Window -> Preferences -> Android* y completamos los parámetros conforme al siguiente cuadro:



Hecho esto, debemos descargar las API de Android con las que trabajaremos. Para esto nos dirigimos a *Window -> Android SDK and ADV Manager -> Available packages* y seleccionamos los componentes listados en la imagen a continuación:



Esto nos descargara la SDK para la plataforma 2.2 que es la que usaremos para el prototipo, ejemplos de la SDK y el API de Google para Android 2.2 que contiene funcionalidades para mapas, que necesitaremos.

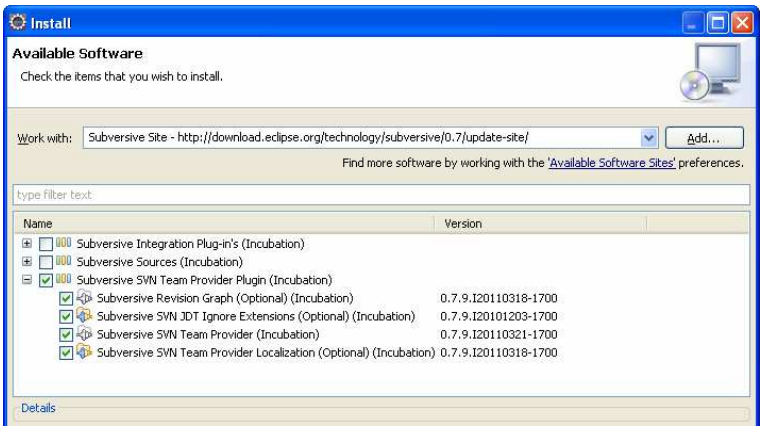
Instalación de Plugin SVN

Necesitaremos la funcionalidad para conectar a un servidor para mantención del código fuente de los prototipos. Utilizaremos un plugin SVN para Eclipse para cubrir esta función, llamado *Subversive*[3]. Debemos agregar nuevos sitios a nuestros *Available Software Sites*:

Name: Subversive plug-in
Location: http://download.eclipse.org/technology/subversive/0.7/update-site/

Name: Subversive SVN Connectors
Location: http://community.polarion.com/projects/subversive/download/eclipse/2.0/update-site/

Seleccionaremos los componentes que se listan en la imagen posterior. Una vez instalador el plugin y reiniciado el Eclipse, el plugin solicitará los conectores a instalar. Estos conectores son los requeridos para utilizar el plugin, y son los encargados de conectarse a los servidores.



Instalación de Maven

Utilizaremos Apache Maven para mantener los ambientes de desarrollo compatibles para todos los miembros del equipo. Lo descargamos desde http://maven.apache.org/download.html.

Descomprimimos el archivo descargado en una ubicación a elección, suponiéndola "C:\Java\apache-maven". Esto es todo lo necesario para Maven en este momento.

Instalación de MySQL

La instalación de MySQL solo comprende la instalación del servidor de bases de datos [4]. Conseguimos el instalador en http://dev.mysql.com/downloads/mysql/. Sugerimos el uso de "MySQL Community Server 5.5.10".

Más información acerca de la instalación de cada componente:
1 - Más info. en http://download.springsource.com/release/STS/doc/STS-installation_instructions.pdf
2 - Más info. en http://developer.android.com/sdk/installing.html#InstallingADT
3 - Más info. en http://www.eclipse.org/subversive/documentation/gettingStarted/aboutSubversive/install.php
4 - Más info. en http://dev.mysql.com/doc/refman/5.1/en/installing.html

Apéndice B - ESTADÍSTICAS DE IMPLEMENTACIÓN

Se muestran a continuación de manera resumida algunas estadísticas de implementación de prototipo, tanto la parte servidor como la del cliente. La finalidad de estas estadísticas es mostrar el esfuerzo invertido en el desarrollo del prototipo.

Estas estadísticas fueron obtenidas con distintos productos de software, al ser incompatibles las distintas plataformas utilizadas, por lo que la diferencia en el número de métricas se debe a esto.

La información del administrador de versiones SVN indica 494 revisiones de actualizaciones. Este número corresponde tanto al Administrador Web como al cliente móvil.

Si bien no tenemos el número exacto de horas invertidas, calculamos que ocupamos cerca de 300 horas de desarrollo.

Estadísticas Administrador Web

Componente	Numero	Líneas de código
Controladores	12	1040
Objetos de dominio	35	790
Servicios	8	601
Helpers Groovy	1	15
Units test	42	498
Total	**98**	**2944**

Estadísticas Cliente

Métrica	Valor
Numero de paquetes	23
Numero de clases	124
Cantidad de líneas de código	5878
Cantidad de métodos	576
Cantidad de atributos	226
Cantidad de métodos estáticos	41
Cantidad de atributos estáticos	236
Cantidad de métodos sobrescritos	47
Cantidad de interfaces	7