|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FACULTAD “SAN FRANCISCO”  UNIVERSIDAD CATÓLICA ARGENTINA   **LICENCIATURA EN SISTEMAS Y COMPUTACIÓN**  DESARROLLO DE SISTEMA DE CONTROL INTERNO PARA GABINETES DE ASESORÍA PEDAGÓGICA EN EDUCACIÓN SEMIPRESENCIAL   |  |  |  | | --- | --- | --- | | ALUMNO |  | VEGA HISSI FRANCISCO JAVIER | | DIRECTOR DE TESINA |  |  | | Prof. Asesor |  | Mg. Jorge Mariotti | | Prof. Asesor |  | Mg. Alejandro Vázquez |     Presentada ante la Secretaría Académica de la Facultad San Francisco U.C.A. como requisito parcial para optar al título de  LICENCIADO EN SISTEMAS Y COMPUTACIÓN  Septiembre de 2015 |

TÍTULO DE LA TESINA:

DESARROLLO DE SISTEMA DE CONTROL INTERNO PARA GABINETES DE ASESORÍA PEDAGÓGICA EN EDUCACIÓN SEMIPRESENCIAL

TESINA PRESENTADA POR:

Alumno: Vega Hissi Francisco Javier.

|  |
| --- |
| Aprobada en contenido por:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Aprobada en estilo por:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Nombre del profesor, miembro del tribunal  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Nombre del profesor, miembro del tribunal  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Nombre del Director de Carrera  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
|  |
|  |

AGRADECIMIENTOS

Agradezco enormemente a mi señora, Fabiana Micaela Cipolla, por no darse por vencida durante todo este tiempo brindandome el apoyo necesario para realizar esta taréa.

Agradezco a mi familia que nunca cuestionó

GRACIAS.

RESUMEN

TÍTULO DE LA TESINA

DESARROLLO DE SISTEMA DE CONTROL INTERNO PARA GABINETES DE ASESORÍA PEDAGÓGICA EN EDUCACIÓN SEMIPRESENCIAL

FECHA DE LA DEFENSA:

NOMBRE DEL ESTUDIANTE:

FRANCISCO JAVIER VEGA HISSI

LICENCIATURA EN SISTEMAS Y COMPUTACIÓN

FACULTAD SAN FRANCISCO U.C.A.

Dirigida por DRA. Paola Guadalupe Caymes Scutari

Este proyecto final de carrera consiste en la creación de una aplicación para el S.O Android la cual hace uso de los servicios web que ofrece el API de Google Maps que le permiten mostrar rutas e indicaciones para llegar a un determinado lugar, distancias y tiempos.

Además implementa distintas soluciones tanto en funcionalidad como en la interfaz gráfica que le dan un aspecto profesional a la misma.

Esta aplicación brinda a los turistas información sobre:

* Recorridos de micros en el Gran Mendoza.
* Hoteles.
* Bodegas que forman parte de los Caminos del Vino.
* Restaurantes, bares, casinos, centros de salud y hospitales cercanos a su ubicación.

ÍNDICE DE CONTENIDO

Contenido

|  |  |
| --- | --- |
| Agradecimientos | iii |
| Resumen | iv |
| Índice de Tablas | vii |
| Índice de Ilustraciones o figuras | viii |

|  |  |
| --- | --- |
| Introducción | 1 |

|  |  |  |
| --- | --- | --- |
| Capítulo 1. Desarrollo de una aplicación de posicionamiento y de localización para la plataforma Android sobre aspectos turísticos de la provincia | |  |
|  | Problema de investigación | 2 |
|  | Objetivo | 2 |
|  | Metas | 2 |
|  | Alcance | 2 |

|  |  |  |
| --- | --- | --- |
| Capítulo 2. La Plataforma Android | |  |
|  | Introducción | 3 |
|  | Características | 3 |
|  | Arquitectura | 5 |
|  | Ciclo de vida de las aplicaciones | 11 |
|  | Política de eliminación de procesos | 13 |
|  | Seguridad | 14 |
|  | Componentes de una Aplicación | 15 |

|  |  |  |
| --- | --- | --- |
| Capítulo 3.Conceptos de desarrollo, entorno y soluciones | |  |
|  | Introducción | 17 |
|  | Plataformas | 17 |
|  | Instalación del entorno | 18 |
|  | Elementos de un proyecto | 19 |
|  | Interfaz de usuario | 21 |
|  | Activity. El problema del hilo principal | 24 |
|  | Librería ActionBarSherlock | 26 |
|  | Activity. El uso de fragmentos | 29 |
|  | Recursos | 31 |
|  | Servicios Web | 34 |
|  | Tratamiento de XML | 40 |
|  | Almacenamiento de datos. Base de datos SQLITE | 44 |

|  |  |  |
| --- | --- | --- |
| Capítulo 4.Localización y Posicionamiento | |  |
|  | Introducción | 47 |
|  | Tecnologías | 47 |
|  | La API Android de Google Maps | 47 |
|  | Overlays (capas de información propias sobre mapas), ItemizedOverlay y OverlayItem | 50 |
|  | Servicios web del API | 51 |
|  | Geocodificación y Geocodificación inversa | 53 |
|  | Obtención de coordenadas | 54 |

|  |  |  |
| --- | --- | --- |
| Capítulo 5.Aplicación de conceptos y soluciones de desarrollo | |  |
|  | Introducción | 56 |
|  | Elección de la plataforma | 56 |
|  | Aplicación de AsyncTask | 56 |
|  | Utilización de recursos alternativos | 60 |
|  | Utilización de fragmentos con la librería ActionBarSherlock | 64 |
|  | Consumición de servicios web del API de Google Maps y de Twitter | 68 |
|  | Realización del parseo de la información obtenida de los servicios de la API Google Maps | 72 |
|  | Implementación y consumición de un servicio web propio | 78 |
|  | Aplicación de Capas | 84 |
|  | Utilización de Geocoder | 89 |

|  |  |
| --- | --- |
| Conclusión | 91 |

|  |  |
| --- | --- |
| Glosario | 92 |

|  |  |
| --- | --- |
| Anexo: Curriculum Vitae | 94 |

|  |  |
| --- | --- |
| Bibliografía | 96 |

ÍNDICE DE TABLAS

|  |  |
| --- | --- |
| Tabla 1: Problemas – Soluciones | 1 |
| Tabla 2: Comparativas con otras plataformas móviles | 4 |
| Tabla 3: Subdirectorios del directorio /res para trabajar con recursos | 32 |

ÍNDICE DE ILUSTRACIONES O FIGURAS

|  |  |  |
| --- | --- | --- |
| Figura 1: | Stack de Android | 5 |
| Figura 2: | Código Java en Android | 8 |
| Figura 3: | Proceso en el cual se juntan varios archivos .class en un solo fichero .dex (1) | 9 |
| Figura 4: | Proceso en el cual se juntan varios archivos .class en un solo fichero .dex (2) | 10 |
| Figura 5: | Ciclo de vida de una activity | 12 |
| Figura 6: | Ilustración de una jerarquía de vistas, que define un diseño de interfaz de usuario | 22 |
| Figura 7: | Application Not Responding | 24 |
| Figura 8: | Configuración del proyecto para usar la librería ActionBarSherlock | 27 |
| Figura 9: | Biblioteca de compatibilidad | 28 |
| Figura 10: | Incluyendo la librería ActionBarSherlock | 29 |
| Figura 11: | Un ejemplo de dos módulos de GUI que típicamente son separados en dos actividades pueden ser combinados en una sola actividad, utilizando fragmentos | 31 |
| Figura 12: | Tecnologías utilizadas en un servicio web avanzado | 38 |
| Figura 13: | Árbol resultante del procesamiento del código XML | 44 |
| Figura 14: | Sitio Web de GeoLocator. Utilizado para obtener las coordenas de las distintas entidades | 54 |
| Figura 15: | Sitio Web de GeoLocator. Resultados. A la izquierda de la imagen se visualizan las coordenadas de la entidad ingresada: -32.902074 y -68.79776. A la derecha los datos relacionados a la entidad como: nombre, dirección y teléfono | 55 |
| Figura 16: | Presencia de las distintas plataformas Android en dispositivos móviles | 56 |
| Figura 17: | Distintas activities que llevan acabo una tarea asíncrona | 60 |
| Figura 18: | Estructura del proyecto. En esta imagen se pude visualizar tres directorios values con su correspondiente sufijo | 61 |
| Figura 19: | Mostrando detalles de la parada más cercana en distintos idiomas. La primera imagen muestra la aplicación en el idioma español, la segunda en inglés y la tercera en portugués | 64 |
| Figura 20: | Estructura del proyecto. Clases necesarias para la utilización de Fragmentos | 67 |
| Figura 21: | Utilización de Fragmentos | 67 |
| Figura 22: | Obtención de datos del servicio web del API de Google Maps | 69 |
| Figura 23: | Datos obtenidos del servicio web del API de Google Maps. Caminando | 69 |
| Figura 24: | Datos obtenidos del servicio web del API de Google Maps. Manejando | 70 |
| Figura 25: | Sitio ID FROM USER. Este sitio devolverá un ID como el siguiente: 1355287615 | 70 |
| Figura 26: | Obtención de datos del servicio web de Twitter | 71 |
| Figura 27: | Creando un servicio web. Paso 1 | 79 |
| Figura 28: | Creando un servicio web. Paso 2 | 79 |
| Figura 29: | Creando un servicio web. Paso 3 | 80 |
| Figura 30: | Servicios web disponibles en el servidor web Tomcat con soporte para servicios web Axis 2 | 81 |
| Figura 31: | Contenido XML del archivo WDSL creado por Eclipse | 82 |
| Figura 32: | Distintas implementaciones de un servicio web. Implementado en SOAP 1.1, SOAP 1.2 y REST (HTTP POST) | 82 |
| Figura 33: | Contenido de un mensaje de solicitud SOAP y su correspondiente mensaje de repuesta SOAP | 83 |
| Figura 34: | Respuesta obtenida del servicio web implementado en REST | 84 |
| Figura 35: | Dos capas sobre un mismo objeto mapview | 85 |

INTRODUCCIÓN

A la hora de iniciar mi tesina, desde un primer momento mi elección del tema fue el desarrollo de una aplicación para dispositivos móviles. Luego de investigar entendí que no solo era importante comprender lo centrado al desarrollo, sino también el funcionamiento básico del sistema operativo que eligiera para el cual desarrollaría la aplicación, de manera de entender como realmente funciona una aplicación en él. Al investigar sobre las distintas opciones encontré un sistema operativo para dispositivos móviles, que a nivel mundial se encuentra en el 76% de los Smartphone que se venden, soporta muchas tecnologías de conectividad y está basado con el sistema Linux, ya probado y aceptado.

Luego de decidir para qué sistema operativo móvil desarrollar; de investigar y aprender lo básico necesario para programar en Android, me encontré con ciertas limitaciones o problemas (por ejemplo, la versión GINGERBREAD de Android no soporta un patrón de diseño llamado ACTION BAR, el problema a la hora de ejecutar tareas que conlleven un cierto tiempo de demora, utilizar recursos, entre otros) que me obligaron a investigar más y encontrar soluciones como en toda nueva tecnología que se aprende. Estas soluciones adoptadas para determinados problemas han sido mencionadas en esta tesina en el capítulo 3, además de los temas básicos para el desarrollo de una aplicación en Android, de manera que quien la lea, puede evaluar las distintas soluciones mencionadas aquí y adoptarlas para sus desarrollos. A continuación detallo los puntos a los que hago referencia anteriormente:

|  |  |
| --- | --- |
| Problema | Solución |
| Crear una Interfaz profesional y dinámica. | Hacer uso de la Librería ActionBarSherlock. Hacer Uso de Fragmentos. |
| Ejecutar tareas en segundo plano. | Hacer uso de la clase AsyncTask. |
| Ofrecer un servicio web propio. | Hacer uso de la Librería Axis2. |
| Mostrar datos en un mapa e interactuar con ellos. | Hacer uso de las Clases Overlays (capaz), ItemizedOverlay y OverlayItem. |
| Ofrecer distintos idiomas de la misma aplicación. | Utilizar recursos según las distintas configuraciones de los dispositivos móviles. |
| Mostrar imágenes para ofrecer mayor información. | Hacer uso del mecanismo de manejo de recursos ofrecido por Android. |

**Tabla N°1. Problemas y soluciones.**

CÁPITULO I. DESARROLLO DE UNA APLICACIÓN DE POSICIONAMIENTO Y DE LOCALIZACIÓN PARA LA PLATAFORMA ANDROID SOBRE ASPECTOS TURÍSTICOS DE LA PROVINCIA

* 1. **Problema de investigación**

Dada la gran oferta turística, es necesario contar con información sobre:

* Recorridos de micros en el Gran Mendoza.
* Hoteles.
* Bodegas que forman parte de los Caminos del Vino.
* Restaurantes, bares, casinos.
* Rutas.

Ésta se encuentra muy dispersa y es difícil acceder a ella dado que existe la necesidad de consultar diferentes sitios web para obtenerla.

* 1. **Objetivo**

Brindarles la información necesaria, a través de un medio portable y de rápido acceso.

* 1. **Metas**
* Utilizar mapas que permitan observar los distintos puntos de interés para las personas.
* Ofrecer información de rutas para llegar desde un punto hacia un destino.
* Indicar tiempos de duración y distancia entre distintos puntos.
* Ofrecer información sobre algún medio de transporte público que acerque al destino y su estado de circulación, como así también sus horarios.
  1. **Alcance**

Si bien existen distintos ámbitos de interés para turistas, se pretende brindar información únicamente sobre:

* Caminos del vino.
* Plaza hotelera.
* Recorridos de micros en el Gran Mendoza.
* Lugares necesarios cercanos a su ubicación.

Se propone como idea en esta tesina que la información sobre otros ámbitos de Mendoza pueda ser de algún modo subida a internet y luego descargada para completar todos los ámbitos de interés de la provincia.

CAPÍTULO II. LA PLATAFORMA ANDROID

* 1. **Introducción**

En este capítulo se explica todo lo relacionado al funcionamiento del S.O Android. Sus características, arquitectura y todo aquello que permita entender cómo funcionan las aplicaciones sobre el mismo.

* 1. **Características**

Existen muchas plataformas para dispositivos móviles como: iPhone, Symbian, Windows Phone, Black Berry, Palm, Java Mobile Edition, Linux Mobile, sin embargo la plataforma Android presenta una serie de características que la hacen diferente:

* Plataforma realmente abierta: Es una plataforma de desarrollo libre basada en Linux y de código abierto.
* Portabilidad asegurada: Las aplicaciones finales son desarrolladas en Java lo que asegura que podrán ser ejecutadas en gran variedad de dispositivos, tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.

* Arquitectura basada en componentes inspirados en internet: Por ejemplo el diseño de la interfaz gráfica de usuario de una aplicación se hace en lenguaje XML, lo que permite que la misma pueda ser ejecutada en un móvil de pantalla reducida o en una netbook (siempre que posea Android). Esto otorga reutilización y portabilidad entre diferentes dispositivos.
* Gran cantidad de servicios incorporados:
* Localización basada tanto en GPS como en torres de telefonía móvil.
* Aplicación de una potente base de datos con SQL mediante librerías de SQLite.
* Reconocimiento de voz.
* Mapas y navegador.
* Alto nivel de seguridad: Los programas se encuentran aislados unos de los otros. En esta plataforma cada aplicación se ejecuta en su propio proceso. La mayoría de las medidas de seguridad entre el sistema y las aplicaciones derivan de los estándares de Linux 2.6, cuyo kernel, constituye el núcleo principal de Android. Cada proceso forma lo que se llama una caja, que proporciona un entorno seguro de ejecución.

Cada aplicación dispone de una serie de permisos que limitan su rango de actuación. Por defecto, éstas no pueden realizar operaciones que impacten negativamente sobre otras o en el sistema mismo. La única forma de poder saltar estas restricciones impuestas por la plataforma Android, es mediante la declaración explícita de un permiso que autoriza a llevar acabo una determinada acción habitualmente prohibida. Además, es obligatorio que cada una esté firmada digitalmente mediante un certificado, cuya clave privada sea la de su desarrollador.

No es necesario vincular a una autoridad de certificado, el único cometido del certificado es crear una relación de confianza entre las aplicaciones. Mediante la firma, la aplicación lleva adjunta su autoría.

La plataforma Android combina todas estas características en una misma solución permitiendo enriquecer la experiencia del usuario y mejorar el proceso de desarrollo.

* Optimización para baja potencia y poca memoria.
* Alta calidad de gráficos y sonido: Gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en 3D basados en Open GL. Incorpora los códecs estándar más comunes de audio y video, como por ejemplo: MP3, ACC, H.264 (AVC).

A continuación se muestra una tabla donde se realiza una comparación entre las distintas plataformas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Características | Apple IOS 4.2 | | Android 2.3 | Symbian OS | Windows Phone 7 | BlackBerry  OS 6 |
| Compañía | Apple | Open Handset Aliance | | Symbian Foundation | Windows | RIM |
| Núcleo del Sistema Operativo | Mac OS X | Linux | | Mobile OS | Windows CE | Mobile OS |
| Familia de CPU Soportada | ARM | ARM, MIPS, Power, x86 | | ARM | ARM | ARM |
| Lenguaje de Programación | C++ | Java, C++ | | C++ | C++ | Java |
| Licencia de Software | Propietaria | Software libre | | Software libre | Propietaria | Propietaria |
| Soporte Flash | No | Sí | | Sí | No | No |
| HTML5 | Sí | Sí | | No | No | Sí |
| Tienda de Aplicaciones | App Store | Android Market | | Ovi Store | Windows Marketplace | BlackBerry App World |
| Número de Aplicaciones | 300.000 | 700.000 | | 13.000 | 8.000 | 30.000 |
| Soporte memoria externa | No | Sí | | Sí | No | No |
| Variedad de dispositivos | Modelo único | Muy alta | | Muy alta | Baja | Baja |
| Tienda de aplicaciones | App Store | Android Market | | Ovi Store | Windows Marketplace | BlackBerry App World |
| Actualizaciones del S. O. | Sí | Depende del fabricante | | Sí | No | Sí |
| Tipo de Pantalla | Capacitiva | Capacitiva  /Resistiva | | Capacitiva  /Resistiva | Capacitiva | Capacitiva  /Resistiva |

**Tabla N°2. Comparativa con otras plataformas móviles.**

* 1. **Arquitectura**

La arquitectura de la plataforma Android se encuentra formada por cuatro capas. Cada una de ellas está basada en software libre.



**Figura 1: Stack de Android. [[1]](#footnote-1)**

**Capa 1. El núcleo Linux**

En la parte inferior de la arquitectura Android se encuentra el kernel de Linux versión 2.6, esta capa es la encargada de manejar funciones básicas e importantes que ya están implementadas, como por ejemplo la lectura de un archivo, la administración de memoria, entre otras. Dado que todo esto ya está implementado en Linux, se reutiliza como núcleo, en Android.

Esta capa actúa como capa de abstracción entre el hardware de los dispositivos móviles y el resto de la pila. Por lo tanto, es la única dependiente del hardware (es decir que se comunica directamente con él). Por esto mismo, esta capa debe contener los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes.

Siempre que un fabricante incluya un nuevo elemento de hardware, lo primero que debe realizar para que pueda ser utilizado desde Android es crear los drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

La elección de Linux 2.6 se ha debido principalmente a dos razones: la primera, su naturaleza de código abierto y libre se ajusta al tipo de distribución que se buscaba para Android, y la segunda es que este kernel de Linux incluye de por sí numerosos drivers, además de contemplar la gestión de memoria, la gestión de procesos, los módulos de seguridad, la comunicación en red y otras muchas responsabilidades propias de un sistemas operativo, que como ya se mencionó, se encuentran bien implementadas.

**Capa 2.Librerías y el Runtine de Android**

**Librerías Nativas**

Estas librerías representan un soporte para los distintos componentes del sistema.

Se trata de un conjunto de librerías en C/C++ usadas en varios componentes de Android, compiladas en código nativo del procesador.

Alguna de estas librerías incluidas son las siguientes:

* SQLite: Potente y ligero motor de bases de datos relacionales disponibles para todas las aplicaciones.
* SSL: Proporciona servicios de encriptación.
* Media Framework: Librería que da soporte para la reproducción y grabación de multitud de formatos de audio, vídeo e imágenes MPEG4, MP3, ACC, JPG, PNG.
* WEBKIT: Librería que da soporte a un moderno navegador web, utilizada en el navegador Web de Android y en la vista Webview (una vista es un componente de una interfaz gráfica. Este tema se menciona en el capítulo 3). Se trata de la misma librería que utiliza Google Crome.
* SGL: Motor de gráficos 2D.

**Entorno de ejecución (el Runtine)**

Al mismo nivel que las librerías nativas de Android se sitúa el **entorno de ejecución (el Runtine)**. Éste, está basado en el concepto de máquina virtual utilizado en Java. Por ende, a éste, lo forman las librerías que constituyen la máquina virtual Dalvik y el “core libraries” con multitud de clases disponibles en el lenguaje Java.

Debido a la variedad y a las limitaciones (de memoria y de procesador) de los dispositivos donde ha de correr Android, no fue posible utilizar la máquina virtual Java estándar. Por esto, Google tomó la decisión de crear una nueva máquina virtual, DALVIK que responda mejor a estas limitaciones.

Alguna de las características de DALVIK que facilitan la optimización de los recursos son:

* Ejecuta ficheros Dalvix ejecutables (.dex), los cuales poseen un formato optimizado para el ahorro de memoria. A diferencia de la máquina virtual Java estándar que ejecuta archivos .jar.
* Está basada en registros.
* Cada aplicación corre en su propio proceso Linux y delega funciones como el manejo de memoria a bajo nivel al núcleo de Linux.

**Dalvik != Java Virtual Machine**

Dalvik es una process virtual machine, también llamada application virtual machine, es gestionada y ejecutada como cualquier otra aplicación, pero su función es la de proporcionar un entorno de programación independiente de la plataforma.

La máquina virtual de Java, que se puede encontrar en casi todas las PC’s actuales, se basa en el uso de las pilas. De modo contrario, Dalvik utiliza los registros, ya que los teléfonos móviles están optimizados para la ejecución basada en los mismos.

Aunque se utiliza el lenguaje Java para programar las aplicaciones Android, el bytecode de Java no es ejecutable en un sistema Android. De igual forma, las librerías Java que utiliza Android son ligeramente distintas a las utilizadas en Java Standard Edition (Java SE) o en Java Mobile Edition (Java ME), guardando también características en común.

**¿Cómo funciona?**

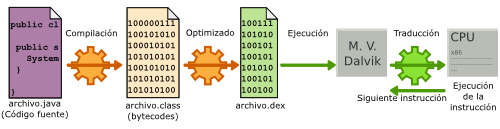
Al ser una máquina virtual intérprete, ejecuta archivos en el formato Dalvik Executable (\*.dex), un formato optimizado para el almacenamiento eficiente y ejecución mapeable en memoria. Su objetivo fundamental es el mismo que la de cualquier máquina virtual, permite que el código sea compilado a un bytecode independiente de la máquina en la que se ejecuta, y la máquina virtual interpreta este bytecode a la hora de ejecutar el programa.

El hecho de no utilizar la máquina virtual de Java, es la necesidad de optimizar al máximo los recursos y enfocar el funcionamiento de los programas hacia un entorno con escasa memoria, procesador y almacenamiento como es el que se tiene en los dispositivos móviles.

Dalvik utiliza un bytecode muy similar pero no igual que el Java Byte Code por lo que en la práctica no son compatibles, por lo que es necesario que las clases .class sean transformadas por una utilidad llamada dx (incluida en el Android SDK) que las transforma en el Bytecode de Dalvik en clases .dex, las cuales finalmente son comprimidas en ficheros .apk.

El intérprete toma los archivos generados por las clases Java y los combina en uno o más archivos Dalvik (.dex), los cuales a su vez son comprimidos en un solo fichero .apk (Android Package) en el dispositivo. De esta forma, reutiliza la información duplicada por múltiples archivos .class, reduciendo así la necesidad de espacio a la mitad de lo que ocuparía un archivo .jar.

A continuación se muestra el proceso que sufre el código fuente para ser ejecutado en un dispositivo Android.



**Figura 2: Código Java en Android. [[2]](#footnote-2)**

## ¿Para qué es necesaria la máquina virtual?

## La máquina virtual es un intermediario entre las aplicaciones y el sistema. Cuando aparece una nueva plataforma hardware, simplemente hay que adaptar DALVIK para que funcione en ella y automáticamente las aplicaciones que ya existían empiezan a funcionar en ella también. Es decir, esta arquitectura es la diferencia entre tener que preparar aplicaciones para todas las plataformas de hardware que soporta Android y simplemente desarrollar para Android. Por eso, su función es la de proporcionar un entorno de programación independiente que permita obviar el hardware instalado por debajo, y posibilite ejecutar los programas realizados para ella en cualquier tipo de plataforma hardware sin tener que realizar modificación alguna.

**Ventajas de la máquina virtual**

El código intermedio (el bytecode) es muy cercano al código máquina, por lo que la penalización en rendimiento es bastante baja al utilizar recursos del sistema al convertir el bytecode en instrucciones máquina. De esta forma se pueden desarrollar aplicaciones portables sin que los desarrolladores se preocupen por el rendimiento.

La máquina virtual provee cosas como un recolector de basura, cachés de acceso I/O y gestión de threads, que en caso de código nativo (no realiza el proceso de bytecode a instrucciones maquinas directas), los desarrolladores tienen que hacerlo ellos mismos, o bien usar librerías externas que, nuevamente, consumen memoria.

**Formato del archivo Dex**

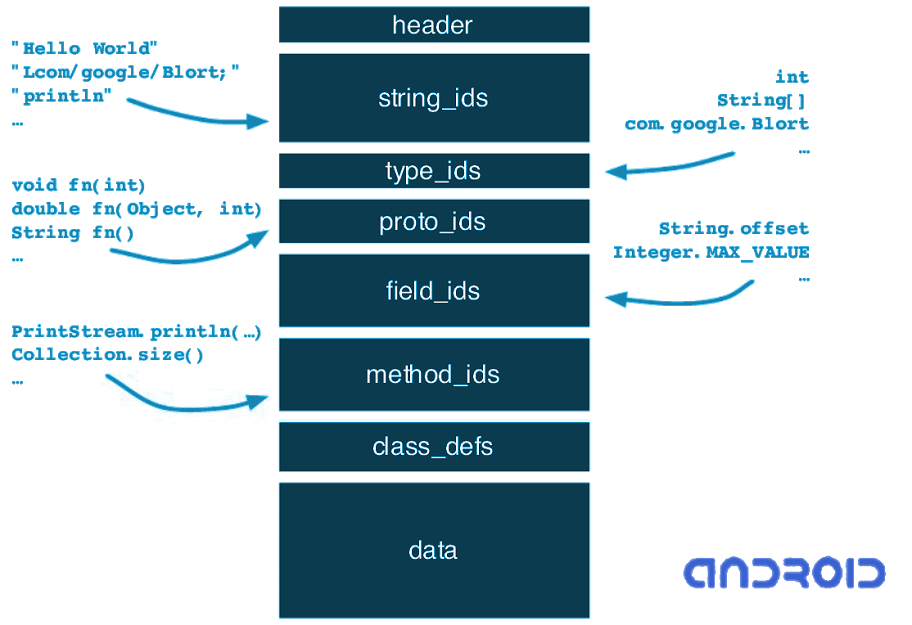
En los entornos [**Java**](http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) estándar, el código fuente de Java es compilado en bytecode de Java, y almacenado en archivos [.class](http://en.wikipedia.org/wiki/Java_class_file). Estos archivos son leídos por la [máquina virtual Java](http://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java) en tiempo de ejecución. Cada clase con código Java se traduce en un archivo .class. Esto significa que si se tiene, por ejemplo, un archivo .java (código fuente) que contiene una clase pública, una clase interna estática, y tres clases anónimas, el proceso de compilación (javac) debe generar 5 archivos .class.

En esta plataforma, el código fuente de Java también se compila en archivos .class. Pero luego de ser generados éstos, mediante la herramienta [**“dx“**](http://developer.android.com/guide/developing/tools/othertools.html) son convertidos a un único fichero .dex ([Dalvik Executable](http://es.wikipedia.org/wiki/Dalvik)). Mientras que un .class contiene una sola clase, un archivo .dex contiene múltiples. Es éste, el que se ejecuta en la máquina virtual Dalvik.

El archivo .dex está optimizado para el mínimo consumo de memoria y el diseño de éste está condicionado por la reutilización de datos. El siguiente diagrama compara el formato de los archivos .class utilizado por la JVM con el formato .dex utilizado por la máquina virtual Dalvik.

A grandes rasgos las estructura de un Archivo .dex consta de las siguientes partes:

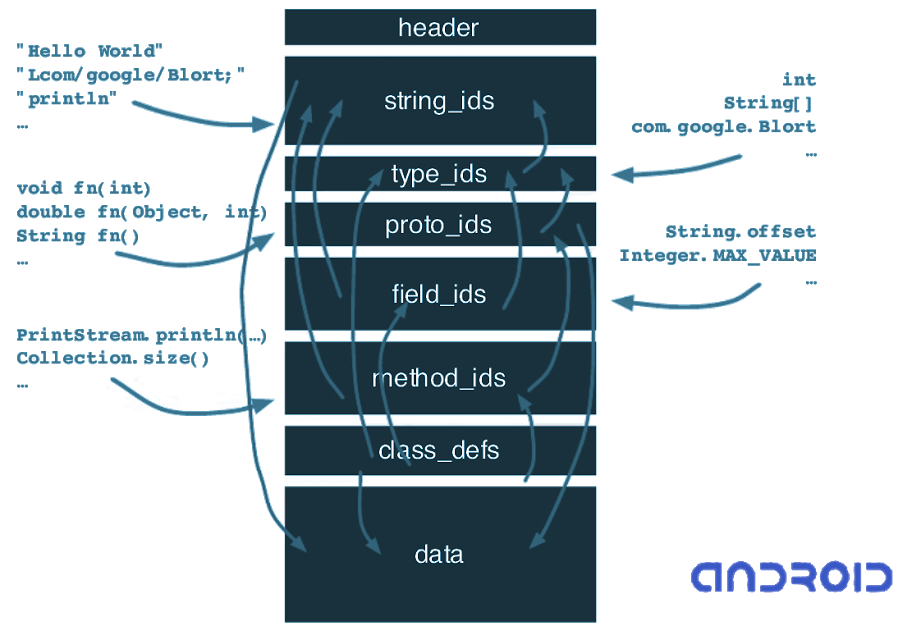
* **Cabecera.**
* Tabla con las posiciones de los **Strings.**
* Tabla con las posiciones de los **Tipos.**
* Tabla con las posiciones de las estructuras/**Prototipo**s de los métodos.
* Tabla con las posiciones de las Propiedades de las clases o **Campos** de los métodos.
* Tabla con las posiciones de los **Métodos.**
* Tabla con las posiciones de las **Clases.**
* **Datos.**



**Figura 3: Proceso en el cual se juntan varios archivos .class en un solo fichero .dex (1).[[3]](#footnote-3)**

Salvo la tabla de Strings (que es a la que hacen referencia todas las tablas ya que es en esta parte donde se almacenan todos los nombre de clases, procedimientos, funciones, variable y tipos de datos), el resto sigue un orden jerárquico inverso, es decir, si se quisiera desensamblar el archivo dex, tras obtener el listado de Strings, se obtendría el listado de Clases, propiedades, campos y estructura de sus métodos. Por último, los tipos de campos y de lo retornado por los métodos.

Es decir, es una estructura relacional, que tiene como objetivo el reaprovechamiento máximo de la información, evita redundancias y así consigue un formato óptimo para terminales móviles. Son tablas donde se indica la posición en la que se se encuentra la información que compone dicha tabla, normalmente mediante un Offset y opcionalmente una longitud. Estos datos junto con el código máquina están en la sección de datos.



**Figura 4: Proceso en el cual se juntan varios archivos .class en un solo fichero .dex (2).[[4]](#footnote-4)**

Gracias a la herramienta “dx”, esta transformación es posible: los ficheros .class de Java se compilan en ficheros .dex, de forma que cada fichero .dex puede contener varias clases. Después, este resultado se comprime en un único archivo de extensión .apk (Android Package), que es el que se instala en el dispositivo móvil.

**Capa 3. Entorno de Aplicación**

Los desarrolladores tienen acceso completo a las mismas APIs del Framework usadas por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede hacer públicas sus capacidades y cualquier otra aplicación puede luego hacer uso de ellas, siempre que no comprometan las reglas de seguridad del Framework. Este modo de funcionamiento permite la reutilización de código.

Entonces esta capa ofrece un conjunto de capacidades o servicios a las aplicaciones como mostrar alertas personalizadas en la barra de estado, acceder a datos de otras aplicaciones (como los contactos), poder ingresar a recursos que no son en código, entre otros.

**Capa 4. Aplicaciones**

Esta capa está formada por el conjunto de aplicaciones instaladas. Aquí es donde trabajan los desarrolladores. Normalmente las aplicaciones Android están escritas en Java. Para desarrollar en Java se utiliza el Android SDK. Existe otra opción menos utilizadas que es la de construir aplicaciones empleando el lenguaje C/C++, pero para esto se utiliza el NDK (Native Development Kit).

Ésta incluye un cliente de correo electrónico, un proveedor de SMS, un calendario, mapas, un navegador web, contactos, más todas aquellas que el usuario desee descargar del Android Market.

* 1. **Ciclo de vida de las aplicaciones**

El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros S.O como Windows. La mayor diferencia es que en Android el ciclo de vida es controlado completamente por el sistema, en lugar de ser controlado directamente por el usuario.

Una aplicación en Android se ejecuta en su propio proceso y está formada por un conjunto de elementos básicos de visualización conocidos como actividades. Es decir, cada proceso está formado por una o varias actividades independientes (componentes Activity). El sistema mantiene una pila con todas las actividades previamente visualizadas, de forma que el usuario puede regresar a la actividad anterior pulsando la tecla “atrás”. Gracias a que cada aplicación se ejecute en su propio proceso se obtienen beneficios en cuestiones básicas como: seguridad, gestión de memoria, o la ocupación de la CPU del dispositivo móvil. Este proceso de la aplicación continúa vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

En Android la destrucción de un proceso no es controlado directamente por la aplicación, es el sistema quien determina cuándo destruir el proceso basándose en el conocimiento que tiene el sistema sobre las partes de la aplicación que están corriendo en ese momento (actividades y servicios), qué tan importantes son para el usuario y cuánta memoria disponible hay en un determinado momento. Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esta aplicación. En estos casos, es responsabilidad del programador almacenar el estado de la aplicación, si se quiere que cuando sea reiniciada conserve su estado. Android lanza tantos procesos como permitan los recursos del dispositivo, si el S.O determina que existen recursos disponibles a la hora de abrir una nueva aplicación el sistema duerme el proceso de la aplicación anterior, éste y las actividades siguen existiendo, pero están dormidos. A la hora de ejecutar una aplicación es entonces cuando Android crea o despierta si ya existe el proceso para la aplicación que debe ser lanzada, asumiendo que existan recursos.

Las actividades dentro de un proceso también tienen su ciclo de vida bien definido. Una actividad en Android puede estar en uno de estos cuatro estados:

* Activa (Running): La actividad está encima de la pila, es decir, que es visible y tiene el foco.
* Visible (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando la actividad está tapada por completo, pasa a estar parada.
* Parada (Stopped): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, entre otras.
* Destruida (Destroyed): Cuando la actividad termina, o es matada por el sistema Android, sale de la pila de actividades.

Cada vez que una actividad cambia de estado se producen eventos que pueden ser capturados por métodos de la actividad en el código en la que se está programando. 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. Por ejemplo, cuando el usuario ha estado utilizando una actividad y tras cambiar a otras regresa a la primera, lo habitual es que la actividad permanezca en memoria y continúe su ejecución sin alteraciones. Pero como se ha dicho anteriormente, en situaciones de poca memoria es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En dicho caso, éste es ejecutado de nuevo pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables y el puntero del programa.

Como consecuencia si el usuario estaba llenando un cuadro de texto perderá esta información. Dependiendo de la sensibilidad de la información que se puede perder, una opción es la utilización de los métodos onPause() para guardar los datos y onResume() para recuperarlos (métodos explicados a continuación). Estos datos deben ser guardados en un medio pertinente como un fichero o base de datos. Posteriormente se muestra un esquema que ilustra los métodos que capturan estos eventos.

**Figura 5: Ciclo de vida de una activity. [[5]](#footnote-5)**

A continuación se mencionan y detallan los métodos que pueden atrapar los eventos generados al cambiar de estado la activity.

onCreate(): Es llamado en la creación de la actividad. Se utiliza para realizar todo tipo de iniciaciones, como la creación de la interfaz de usuario.

onStart(): Indica que la actividad está a punto de ser mostrada al usuario.

onResume(): Se llama cuando la actividad comienza a interactuar con el usuario. Es un buen lugar para lanzar animaciones y música.

onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, es un lugar indicado para guardar los datos que estaban en edición.

onStop(): Cuando este método es llamado, la actividad ya no es visible para el usuario. Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

onRestar(): Indica que la actividad vuelve a ser representada después de haber pasado por onStop().

onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

* 1. **Política de eliminación de procesos**

Como ya se mencionó, cada aplicación Android corre en su propio proceso. Éste se crea cada vez que una aplicación necesita ejecutar parte de su código, y sigue existiendo hasta que el sistema necesite utilizar parte de sus recursos para otra aplicación, considerada prioritaria.

Android mantiene en memoria todos los procesos que quepan aunque éstos no se estén ejecutando (procesos pausados). Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, el sistema determina qué proceso de los que están en ejecución ha de ser eliminado.

Para esto Android ordena los procesos en una lista de jerarquía, asignándole a cada uno una determinada “importancia”. Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (Activity, Broadcast Intent Receiver, Service y Content Provider) y el estado de éstos. No usarlos correctamente a la hora de construir una aplicación puede significar que el sistema operativo la termine cuando en realidad está haciendo una tarea importante para el usuario.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

* **Proceso de primer plano:** Aquellos que hospedan una actividad en la superficie de la pantalla y con la cual el usuario está interactuando. Debería haber solo uno o pocos procesos de este tipo. Sólo son eliminados como último recurso, si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo.
* **Proceso visible:** Aquellos que hospedan una actividad que está visible en la pantalla, pero no en primer plano (su método onPouse() ha sido llamado). Considerado importante, no es eliminado a menos que sea necesario para mantener los procesos de primer plano.
* **Proceso de servicio:** Hospeda un servicio que ha sido inicializado con el método startService(). Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son importantes (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos). El sistema siempre trata de mantener esos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.
* **Proceso de fondo:** Hospeda una actividad que no es actualmente visible al usuario (su método onStop() ha sido llamado). Si estos procesos son eliminados no tendrán un directo impacto en la experiencia del usuario. Generalmente hay mucho de estos procesos, por lo tanto el sistema trata de asegurar que el último visto por el usuario sea el último en ser eliminado.
* **Proceso vacío:** No hospeda a ningún componente de aplicación activo. La única razón para mantener ese proceso es tener un “caché” que permita mejorar el tiempo de activación en la próxima vez que un componente de su aplicación sea ejecutado.

En resumen, Android, según esta jerarquía, 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.

* 1. **Seguridad**

Si se descarga una aplicación maliciosa, de internet o del market, ésta puede leer la lista de contactos, las contraseñas almacenadas, las fotos y videos, averiguar la posición con el GPS incorporado al teléfono y mandar toda esta información por internet o a través de SMSs.

Por estos motivos y por muchos más la seguridad en una aplicación Android como la de todo sistema es un aspecto clave.

Ésta se fundamenta en tres pilares elementales:

* **Android está basado en Linux.** La mayoría de las medidas de seguridad entre el sistema y las aplicaciones derivan de los estándares de Linux 2.6, cuyo kernel, recuérdese, constituye el núcleo principal de Android. Un ejemplo de medida de seguridad de Linux espara proteger el acceso a recursos utilizados por otras aplicaciones, Android crea una cuenta de usuario Linux (user ID) nueva por cada paquete (.apk) instalado en el sistema. Este usuario es creado cuando se instala la aplicación y permanece constante durante toda su vida en el dispositivo. Cualquier dato almacenado por la aplicación es asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones.
* **Toda aplicación está firmada con un certificado digital que identifique a su autor.**
* **Si se desea que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad de dicho sistema se debe utilizar un modelo de permisos; de manera que el usuario puede conocer los riesgos antes de instalar una aplicación.** Para proteger ciertos recursos y características especiales del hardware, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intensión de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se genera una excepción de permiso y la aplicación se interrumpe inmediatamente. Cuando el usuario instala una aplicación, éste puede examinar una lista de permisos que solicita la misma y decidir si considera oportuno instalarla o no. El esquema para definir un permiso es:

<uses-permission>

**Nombre\_Del\_Permiso. Grupo al que pertenece – descripción del permiso que se muestra al usuario antes de instalar la aplicación. Descripción y comentarios.**

Para establecer un permiso para una aplicación, es necesario declarar en el manifiesto uno o más elementos <uses-permission> donde se especifica el tipo de permiso que se desea habilitar.

En la clase Android.Manifest.permission se especifican todos los posibles permisos que se pueden conceder a una aplicación: utilización de Wi-Fi, Bluetooth, llamadas telefónicas, cámara, Internet, mensajes SMS y MMS, vibrador, entre otros.

El elemento <uses-permission> contempla una serie de atributos que definen y matizan el alcance del permiso dado:

**Android: name:** Especificación del permiso que se pretende conceder. Debe ser un nombre de alguno de los listados en la clase Android.Manifest.permission.

**Android: label:** Una etiqueta o nombre convencional fácilmente legible para el usuario.

**Android: permissionGroup:** Permite especificar un grupo asociado al permiso. Los posibles grupos se encuentran listados en la clase Android.Manifest.permission y pueden tener valores como ACCOUNTS (cuentas válidas de Google), COST\_MONEY (acciones que llevan vinculadas un pago) o PHONE\_CALLS (acciones relacionadas con llamadas), entre otros.

**Android: protectionLevel:** Determina el nivel de riesgo del permiso, y en función del mismo influye en cómo el sistema otorga o no el permiso a la aplicación. Oscila entre valores desde el 0 hasta el 3.

**Android: description:** Descripción textual del permiso.

**Android: icon:** Icono gráfico que puede ser asociado al permiso.

* 1. **Componentes de una aplicación**

Todas las aplicaciones Android están desarrolladas a partir de cuatro tipos de bloques o componentes principales. Cada una de ellas es una combinación de uno o más de éstos según las necesidades. Estos, componentes o bloques principales deben estar declarados de forma explícita en un fichero con formato XML denominado “AndroidManifest.xml”.

Los componentes principales son los siguientes:

**Activity.**

Este componente refleja una determinada labor realizada por la aplicación, donde se necesita la interacción del usuario para poder llevarla acabo. En ciertas circunstancias una tarea necesita de varias subtareas para poder cumplir su prometido, de esta manera en ocasiones son necesarias más de una activity para completar su trabajo. Una activity lleva asociada una ventana o una interfaz de usuario.

Éste se implementa mediante la clase de mismo nombre Activity.

**Intento (Intent).**

Este componente refleja la intención o voluntad de hacer algo como realizar una llamada de teléfono o visualizar una página web.

Éste se implementa mediante la clase de mismo nombre Intent.

**Servicio (Service).**

Este componente representa una tarea o proceso ejecutado sin interfaz de usuario, y que generalmente tiene lugar en segundo plano mientras otras tareas (éstas con interfaz) son las que están activas en la pantalla del dispositivo. Es parecido a un demonio en Linux o a un servicio en Windows. Un ejemplo de uso de un service puede ser cuando se desea actualizar la aplicación. Mientras el usuario realiza alguna tarea propia de la aplicación (es decir está interactuando con una activity), en segundo plano la aplicación (el service) puede estar descargando los archivos para la actualización.

Éste se implementa mediante la clase de mismo nombre Service.

En Android se disponen de dos tipos de servicios: servicios locales, que pueden ser utilizados por aplicaciones locales, es decir, del mismo terminal y servicios remotos, que pueden ser utilizados desde otras terminales.

**Proveedores de contenido.**

Este componente brinda un mecanismo estándar para compartir información entre aplicaciones y a su vez permite distribuir datos sin necesidad de comprometer la seguridad del sistema de ficheros.

Con éste, se puede acceder a información de otras aplicaciones, como la lista de contactos, o proporcionar datos desde una aplicación a otras.

Éste se implementa mediante la clase de mismo nombre Service.

Una clase que implemente a Content Provider contiene una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación.

CAPÍTULO III: CONCEPTOS DE DESARROLLO, ENTORNO Y SOLUCIONES

* 1. **Introducción**

En este capítulo se muestra y explica todo lo relacionado con el desarrollo de una aplicación Android. Desde los conceptos básicos para poder programar, APIs, el entorno de desarrollo y los plugin necesarios, como así también temas complementarios para el desarrollo de aplicaciones más robustas y profesionales.

* 1. **Plataformas**

Antes de empezar a programar en Android se debe evaluar la versión del sistema para la que se desea realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una determinada versión, si se van a utilizar ha de conocerse la versión mínima necesaria.

Cuando se ha lanzado una nueva plataforma siempre ha sido compatible con las versiones anteriores. Lo habitual es añadir nuevas funcionalidades. En el caso de modificar alguna, ésta no se elimina, se etiqueta como obsoleta pero se puede continuar utilizando.

Las plataformas lanzadas hasta la fecha son:

* **Android 1.0 Apple Pie:** La primera versión comercial, fue lanzada el 23 septiembre de 2008. El primer dispositivo Android, el [HTC Dream](http://es.wikipedia.org/wiki/HTC_Dream), incorporó las características de Android 1.0.
* **Android 1.1 Banana Bread:** El 9 de febrero de 2009, La actualización Android 1.1 fue lanzada. La actualización resolvió fallos, cambió la API y agregó una serie de características.
* **Android 1.5 Cupcake:** El 30 de abril de 2009, La actualización de Android 1.5 fue lanzada, basada en [núcleo Linux](http://es.wikipedia.org/wiki/N%C3%BAcleo_Linux) 2.6.27. Ésta fue la primera versión en usar un nombre basado en un postre ("Cupcake",[Magdalena glaseada](http://es.wikipedia.org/wiki/Cupcake)), un tema que se utilizaría para todas las versiones de aquí en adelante. La actualización incluye varias nuevas características y correcciones de interfaz de usuario.
* **Android 1.6 Donut Pantalla de inicio de Android 1.6:** El 15 de septiembre de 2009, el SDK de Android 1.6 SDK – dubbed Donut – fue lanzado, basado en el núcleo Linux 2.6.29. Son numerosas características nuevas.
* **Android 2.0/2.1 Eclair:** El 26 de octubre de 2009, el SDK de Android 2.0 – con nombre en clave Eclair – fue lanzado, basado en el núcleo de linux 2.6.29.
* **Android 2.2.x Froyo:** El 20 de mayo de 2010, El SDK de Android 2.2 (Froyo, yogur helado) fue lanzado, basado en el núcleo Linux 2.6.32.
* **Android 2.3.x Gingerbread:** El 6 de diciembre de 2010, el SDK de Android 2.3 (Gingerbread) fue lanzado, basado en el núcleo Linux 2.6.35
* **Android 3.x Honeycomb:** El 22 de febrero de 2011, el SDK de Android 3.0 (Honeycomb) – fue la primera actualización exclusiva para [tablet](http://es.wikipedia.org/wiki/Tabletas) Android – fue lanzada, basada en el núcleo de linux 2.6.36. El primer dispositivo con esta versión, fue la tableta [Motorola Xoom](http://es.wikipedia.org/wiki/Motorola_Xoom), fue lanzado el 24 de febrero de 2011.

### Android 4.0.x Ice Cream Sandwich: El SDK para Android 4.0.1 (Ice Cream Sandwich), basado en el núcleo de linux 3.0.1,fue lanzado públicamente el 19 de octubre del 2011.Gabe Cohen de Google declaró que Android 4.0 era "teóricamente compatible" con cualquier dispositivo Android 2.3.x en producción en ese momento. El [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente) para Android 4.0 se puso a disposición el 14 de noviembre del 2011.

### Android 4.1 Jelly Bean: Google anunció Android 4.1 (Jelly Bean) en conferencia [Google I/O](http://es.wikipedia.org/wiki/Google_I/O) el 27 de junio de 2012. Basado en el núcleo de linux 3.0.31, Jelly Bean fue una actualización incremental con el enfoque primario de mejorar la funcionalidad y el rendimiento de la interfaz de usuario.

### Android 4.2 Jelly Bean: Se esperaba que Google anunciara Jelly Bean 4.2 en un evento en la ciudad de [Nueva York](http://es.wikipedia.org/wiki/Nueva_York) el 29 de octubre de 2012, pero el evento fue cancelado debido al [Huracán Sandy](http://es.wikipedia.org/wiki/Hurac%C3%A1n_Sandy). En lugar de reprogramar el evento en vivo, Google anunció la nueva versión con un comunicado de prensa, bajo el slogan "A new flavour (US: flavor) of Jelly Bean". El primer dispositivo en correr Android 4.2 fue el [Nexus 4](http://es.wikipedia.org/wiki/Nexus_4) de [LG](http://es.wikipedia.org/wiki/LG) y el [Nexus 10](http://es.wikipedia.org/wiki/Nexus_10) de [Samsung](http://es.wikipedia.org/wiki/Samsung), los cuales fueron lanzados el 13 de noviembre de 2012.

* 1. **Instalación del entorno de desarrollo**

Para el desarrollo de aplicaciones Android existen varias alternativas para usar como entorno de desarrollo, pero la elección es comúnmente utilizar uno que al igual que Android esté basado en software libre:

* Java JRE 7.0. (Java Runtine Environment)
* Eclipse (Eclipse IDE for Java Developers).
* Android SDK (Google).
* Eclipse Plug-in (Android Development Toolkit -ADT).

**Instalación de la máquina virtual Java (Java JRE)**

Este software permite ejecutar código Java en el equipo. A la máquina virtual Java también se la conoce como entorno de ejecución Java – Java Runtine Environment (JRE) o Java Virtual Machine (JVM).

Para instalar la Máquina Virtual Java hay que descargarla de http:java.com/es/download y seguir los pasos ofrecidos por el Wizard.

**Instalación de Eclipse**

Eclipse resulta el entorno de desarrollo más recomendable para Android, es libre y además es soportado por Google (ha sido utilizado por los desarrolladores de Google para crear Android).

La instalación de Eclipse incluye la instalación del SUN JDK, para instalarlo hay que descargar el instalador de <http://www.eclipse.org/downloads> y descomprimir los ficheros en la carpeta que uno prefiera.

**Instalación del Android SDK de Google**

**El Android SDK** es el kit de desarrollo necesario para crear aplicaciones para [Android](http://www.gizmos.es/5996/moviles/android-moviles-open-source-de-la-mano-de-google/). En este **kit** se puede encontrar todas las herramientas que los desarrolladores necesitan: entornos de desarrollo y depuración, librerías, un emulador de móviles, documentación, tutoriales, código de ejemplo.

Para la instalación del kit de desarrollo de Android hay que realizar los siguientes pasos:

Descargar el instalador de <http://developer.android.com/sdk/> y descomprimir los ficheros en la carpeta que uno prefiera. Luego ejecutar el archivo SDK Setup.exe. A continuación seleccionar los paquetes a instalar. Opcionalmente crear un dispositivo virtual Android (AVD) para testear la aplicación.

**Instalación del plug-in Android para Eclipse (ADT)**

Este software desarrollado por Google, instala una serie de complementos en Eclipse, de forma que éste queda adaptado para permitir la construcción de aplicaciones. Se crean nuevos botones, tipos de aplicación, vistas, entre otros.

Para la instalación del plug-in Android para Eclipse (ADT)hay que realizar los siguientes pasos:

* Iniciar Eclipse y seleccionar “Help” luego “Install New Software”. En el díalogo que se abre “Available Software” hacer click en Add. En el cuadro Add Site que sale, hay que introducir un nombre para el sitio remoto y en el campo Location hay que introducir la siguiente URL <http://dl-sal.google.com/android/eclipse/>.
* Ahora en el cuadro “Available Software” debe aparecer “Developer Tools” en la lista. Se debe pulsar en el recuadro y se seleccionarán automáticamente todas las herramientas que hay anidadas en Android DDMS y Android Development Tools.
* Se debe seleccionar los dos paquetes a instalar y luego pulsar Next. A continuación se debe reiniciar Eclipse.
  1. **Elementos de un proyecto Android**

Cuando se crea un proyecto Android en Eclipse, éste crea por defecto una nueva carpeta con el nombre de dicho proyecto en el WORKSPACE. En esta carpeta se crea una estructura especial de directorios, subdirectorios y archivos que constituyen la anatomía de un proyecto Android.

Un proyecto Android está formado básicamente por un descriptor de la aplicación (AndroidManifest.xml), el código fuente y una serie de ficheros con recursos. Como se dijo anteriormente cada uno de los elementos del proyecto se almacena en una carpeta específica.

Las carpetas y ficheros han sido creadas por el plug-in ADT, un proyecto Android posee la siguiente estructura:

* **Carpeta src:** Carpeta que contiene los archivos fuente .java del proyecto. Utiliza la misma jerarquía de carpetas que la indicada por el nombre del paquete que se haya asignado.
* **Carpeta gen:** Carpeta que contiene el código generado de forma automática por el SDK. Nunca hay que modificar de forma manual estos ficheros. Dentro de esta carpeta se encuentran los siguientes ficheros:
  + - **R.java:** “R.java”, es un archivo que siempre se adjunta por defecto a cualquier proyecto Android y que declara una serie de índices o referencias a los recursos externos que se utilizan en el proyecto actual. De esta forma los recursos pueden ser accedidos desde Java.
    - **Android x.x:** Código JAR, el API de Android según la versión seleccionada.
* **Carpeta assets:** Carpeta que puede contener una serie arbitraria de ficheros o carpetas que pueden ser utilizadas por la aplicación (ficheros de datos, ficheros JAR externos, fuentes). A diferencia de la carpeta res, nunca se modifica el contenido de los ficheros de esta carpeta.
* **Carpeta res:** La carpeta “\res” alberga los recursos utilizados en el proyecto. Por recurso se entiende cualquier fichero externo que contenga datos o descripciones referentes a la aplicación, y que debe ser compilado junto a los ficheros fuente. Esta compilación permite al recurso ser accedido de forma más rápida y eficiente. Android contempla muchos tipos de recursos, como XML, JPEG o PNG, entre otros. En la mayoría de los casos, un recurso constituye (o viene acompañado de) un fichero XML que lo describe y/o configura. Por ejemplo, un caso típico de recurso en Android son las traducciones de una aplicación a distintos idiomas. A pesar de ser cadenas de texto, se representan como recurso a través de un documento XML. Cada uno de los recursos utilizados en una aplicación para 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:** Esta carpeta contiene recursos que pueden ser dibujados en la pantalla. Por ejemplo, imágenes con formato JPG, PING o GIF.
* **Layout:** Contiene diseños que pueden ser usados para construir interfaces. Un diseño, que representa la pantalla completa o simplemente una parte, se describe a través de un fichero XML.
* **Menú:** Contiene archivos XML con los menús de la aplicación.
* **Values:** Esta carpeta contiene ficheros XML que declaran valores de diferentes tipos. Se utiliza para indicar valores del tipo string, color o estilo. De esta manera se pueden cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, permite traducir una aplicación a otro idioma.
* **Anim:** Aquí se ubican aquellos ficheros XML que describen una animación para un determinado elemento. Las animaciones son movimientos y efectos gráficos básicos.
* **Xml:** En esta carpeta se ubican aquellos ficheros XML genéricos que pueden ser procesados como tales desde el código (es decir, utilizando un parseador de XML).

* **Raw:** En esta carpeta se incluyen aquellos recursos que son añadidos a la compilación directamente sin procesar, de forma que la aplicación puede leerlos como flujos de bytes. Un recurso típico de esta carpeta son los archivos de audio. Generalmente cualquier fichero que no se encuentre en formato XML.
* **AndroidManifest.xml:** Este fichero describe la aplicación Android. En él se indican las actividades, intentos, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que requiere la aplicación. Se indica la versión mínima de Android para poder ejecutarla.
* **Default.properties:** Fichero generado automáticamente por el SDK. Nunca hay que modificarlo. Se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.
* **Doc:** En esta carpeta se encuentra la documentación asociada al proyecto.
  1. **Interfaz de usuario**

Como se dijo anteriormente, la interfaz de usuario se puede diseñar de dos formas distintas mediante el propio código Java en una activity o mediante layouts en XML. Éste da la posibilidad de separar el código propio del diseño de la interfaz del código que define el comportamiento de la misma.

Todos los elementos de la interfaz de usuario de una aplicación Android son construidos usando objetos “View” and “ViewGroup”. Un View (o una vista) es un objeto que dibuja un elemento en la pantalla con el que el usuario puede interactuar.

Un ViewGroup es un objeto que contiene otros objetos view y viewgroup en el orden que define el layout de la interfaz.

Android ofrece una colección de ambos, de View y de subclases de ViewGroup que le ofrecen los controles comunes de entrada (por ejemplo, botones y campos de texto) y diversos modelos de diseño (como un diseño lineal o relativo) respectivamente.

Entonces cada componente de interfaz de usuario (cada interfaz de usuario en si misma) de una aplicación es definido usando una herencia de objetos View y ViewGroup como se muestra en la figura a continuación.

Cada viewgroup es un contenedor invisible que organiza objetos view hijos, mientras los objetos view hijos pueden ser controles de entrada u otros widgets que dibujan alguna parte de la pantalla. Este árbol de herencia puede ser tan simple como complejo según lo amerite la aplicación.



**Figura 6: Ilustración de una jerarquía de vistas, que define un diseño de interfaz de usuario.[[6]](#footnote-6)**

Para declarar el diseño de la interfaz de usuario se puede instanciar objetos View en el código y comenzar construyendo un árbol, pero la forma más eficiente y más efectiva es definir el diseño con un archivo XML. Éste ofrece una estructura legible para el diseño, similar a HTML.

El nombre de un elemento XML para una vista es la clase respectiva a Android que la representa. Así un elemento <TextView> crea un TextView en la interfaz gráfica, y un elemento <LinearLayout> crea un viewgroup LinearLayout.

Cuando se carga un recurso de diseño en una aplicación, Android inicializa cada nodo del diseño dentro del objeto en tiempo de ejecución, que se puede usar para definir comportamientos adicionales, consultar el estado del objeto, o modificar el diseño.

**Componentes de interfaces de usuario predeterminados**

No se tiene que construir todos los elementos de la interfaz de usuario usando objetos View y ViewGroup. Android provee muchos componentes de aplicación que ofrecen un diseño de interfaz gráfica estándar para la que sólo hay que definir el contenido. Cada uno de estos componentes de interfaz de usuario tiene un conjunto único de APIs que se describen en sus respectivos documentos, tales como barra de acciones, diálogos y notificaciones de estado.

**Creación de la interfaz de usuario usando XML**

Un principio importante en el diseño de software es que conviene separar toda la lógica de diseño lo más posible de los datos y de la lógica de negocios es decir la lógica de la aplicación. Para esto Android proporciona una alternativa para el diseño de las interfaces de usuario: los diseños basados en XML.

**Layouts en XML**

Un layout es un recurso en el que se puede describir lo que se desea mostrar por pantalla y cómo se quiere mostrar. La manera más común de crearlo es a través de un archivo XML (en el directorio res/layout del proyecto), con un formato muy similar a HTML, que sigue este patrón:

**<Nombre\_del\_layout atributo1=”valor1”… atributoN=”valorN”> elementos/componentes**

**</Nombre\_del\_layout>.**

Una vez creado el archivo XML que define el diseño de la pantalla, hay que cargarlo desde el código de la aplicación, en el método onCreate() de la actividad. Debe ser similar a esto:

**public void onCreate(Bundle savedInstance) {**

super.onCreate(savedInstance);

setContentView(**R.layout.nombre\_del\_layout**);

**}**

Todos los layouts tienen parámetros específicos. Los únicos que son comunes a todos son layout\_height y layout\_width, que sirven para especificar el valor de la altura y el ancho del entorno, respectivamente. Aunque se pueden emplear valores numéricos, con “wrap\_content”, el

tamaño se ajusta a las dimensiones del contenido, es decir ocupa lo que tiene que ocupar. Con “fill\_parent”, será tan grande como lo permita su padre o contenedor. Estas dos opciones son más recomendables que el ajuste manual.

También se pueden establecer y consultar los márgenes, bordes y la posición del layout, entre otras opciones, mediante métodos.

Cada componente tiene su propia variedad de atributos en XML. El atributo “ID” se encarga de distinguirlos del resto, otorgándole un nombre único. Una vez establecido (mediante, por ejemplo, android:id="@+id/nombre"), para referenciarlo desde el código de la aplicación es preciso hacerlo de esta manera:

Tipo myTipo = (Tipo) findViewById(R.id.nombre);

Existe una gran cantidad de atributos, que varían dependiendo del componente con el que se está tratando. Con éstos se puede establecer el color de fondo, tamaño, gravedad y un sin fin de opciones más.

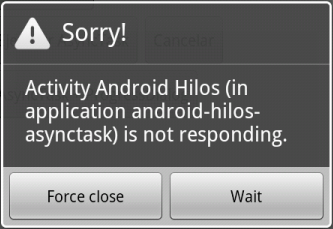
A continuación se muestra como ejemplo un layout con una caja de texto y un botón:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
              android:layout\_width="fill\_parent"   
              android:layout\_height="fill\_parent"  
              android:orientation="vertical" >  
    <TextView android:id="@+id/text"  
              android:layout\_width="wrap\_content"  
              android:layout\_height="wrap\_content"  
              android:text="I am a TextView" />  
    <Button android:id="@+id/button"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="I am a Button" />  
</LinearLayout>

* 1. **Activity. El problema del hilo principal**

Todos los componentes de una aplicación Android, tanto las actividades, los servicios, o los broadcast receivers se ejecutan en el mismo hilo de ejecución, el llamado hilo principal, main thread o GUI thread, que como este último nombre indica también es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación.

Es por ello, que cualquier operación larga o costosa que se realice en este hilo puede bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo, o mal funcionamiento en general, algo que se debe evitar a toda costa. Incluso puede ser peor, dado que Android monitoriza las operaciones realizadas en el hilo principal y detecta aquellas que superen los 5 segundos, en cuyo caso se muestra el famoso mensaje de “Application Not Responding” (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine.



**Figura 7: Application Not Responding.[[7]](#footnote-7)**

Para evitar inconvenientes a la hora de ejecutar una operación de larga duración (por ejemplo acceder a una base de datos con una consulta muy compleja, ejecutar un servicio), existe una clase en el SDK de Android llamada AsyncTask, la cual es de gran ayuda para desarrollar una aplicación profesional. AsyncTask permite el uso apropiado y fácil del hilo de la interfaz de usuario.

Esta clase permite realizar operaciones en segundo plano y publicar los resultados en el hilo de la interfaz de usuario sin tener que manipular los hilos directamente el desarrollador.

AsyncTask está diseñada para ser una clase de ayuda en hilos, y no constituye un marco genérico para el trabajo con hilos. AsyncTasks idealmente se debe utilizar para las operaciones de corta duración (unos segundos como máximo).

Una tarea asíncrona se define mediante un cálculo que se ejecuta en un subproceso de fondo y cuyo resultado se publica en el hilo de la interfaz de usuario. Una tarea asíncrona se define por tres datos, denominados parámetros, progreso y resultados, y por 4 pasos, llamados onPreExecute, doInBackground, onProgressUpdate y onPostExecute.

AsyncTask debe ser heredada, es decir poseer una subclase para ser utilizada. La subclase debe sobrescribir al menos un método (doInBackground(Params…)) y comúnmente debe sobrescribir un segundo método (onPostExecute(Result)) .

Ejemplo de una tarea asíncrona:

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
     protected Long doInBackground(URL... urls) {  
         int count = urls.length;  
         long totalSize = 0;  
         for (int i = 0; i < count; i++) {  
             totalSize += Downloader.downloadFile(urls[i]);  
             publishProgress((int) ((i / (float) count) \* 100));  
             // Escape early if cancel() is called  
             if (isCancelled()) break;  
         }  
         return totalSize;  
     }  
     protected void onProgressUpdate(Integer... progress) {  
         setProgressPercent(progress[0]);  
     }  
     protected void onPostExecute(Long result) {

         showDialog("Downloaded " + result + " bytes");  
     }  
 }

Una vez creada, una tarea se ejecuta simplemente llamando a su constructor para instanciarla de la siguiente manera: new DownloadFilesTask().execute(url1,url2,url3);

**Datos de AsyncTask**

Los tres tipos de datos utilizados por una AsyncTask son los siguientes:

* Parámetros, el tipo de los parámetros enviados a la tarea en ejecución.
* Progreso, el tipo de las unidades de progreso publicados durante el cálculo del fondo.
* Resultado, el tipo del resultado de la computación de fondo.

No todos los tipos se utilizan siempre por una AsyncTask. Para marcar un tipo como sin usar, sólo tiene que utilizar el tipo Void:

Private class MyTask extends AsyncTask<Void,Void,Void>{ … }

Estos datos son la base para que se pueda realizar la tarea, es la información que se necesita pasarle para que se lleve acabo correctamente.

**Pasos de AsyncTask**

Cuando una tarea asíncrona se ejecuta, el trabajo se hace a través de 4 pasos:

* OnPreExecute (): Se invoca en el hilo de la interfaz de usuario antes de que se ejecute la tarea. Este paso se utiliza normalmente para configurar la tarea, por ejemplo, mostrando una barra de progreso en la interfaz de usuario.
* doInBackground (Params. ..): Se invoca en el hilo de fondo inmediatamente después de que OnPreExecute () termina de ejecutarse. Este paso se utiliza para realizar cálculos de fondo que pueden llevar mucho tiempo. Los parámetros de la tarea asíncrona se pasan a éste. El resultado del cálculo debe ser devuelto por éste y se pasa al último método. En este paso también se puede utilizar el procedimiento publishProgress (Progress. ..) para publicar una o más unidades de progreso. Estos valores se publican en el hilo de interfaz de usuario, en el procedimiento onProgressUpdate (Progress. ..).
* onProgressUpdate (Progress. ..): Se invoca en el hilo de interfaz de usuario después de una llamada a publishProgress (Progress. ..). El momento de la ejecución no está definido. Este método se utiliza para mostrar cualquier forma de progreso en la interfaz de usuario, mientras que el cálculo de fondo todavía se está ejecutando. Por ejemplo, puede ser utilizado para animar una barra de progreso.

* OnPostExecute (resultado): Se invoca en el hilo de interfaz de usuario una vez finalizado el cálculo de fondo. El resultado del cálculo de fondo se pasa a este método como un parámetro.

**Reglas del uso de hilos con AsyncTask**

* La clase AsyncTask debe ser cargada en el hilo de la interfaz de usuario. Esto se realiza automáticamente en versiones como JELLY\_BEAN.
* La instancia de la tarea debe ser creada en el hilo de la interfaz de usuario, execute (Params. ..) debe ser invocado en el hilo de la interfaz de usuario.
* No se puede invocar OnPreExecute (), OnPostExecute (resultado), doInBackground (Params. ..), onProgressUpdate (Progress. ..) manualmente.
* La tarea puede ser ejecutada una sola vez (una excepción es lanzada si se intenta una segunda ejecución).
  1. **Librería ActionBarSherlock**

En Android existe un patrón de diseño llamado [ActionBar](https://developer.android.com/design/patterns/actionbar.html), el cual consiste en una barra situada en la parte superior de la aplicación. Esta barra identifica la aplicación y permite al usuario saber en qué parte de la aplicación se encuentra. También permite situar acciones y modos de navegación fácilmente usables para el usuario.

Este patrón se introdujo con la versión Android 3.0 (Honeycomb - API 11) por lo que no está disponible para versiones inferiores. Al no estar soportado de forma oficial en las versiones inferiores a Android 3.0, para poder usarlo es necesario recurrir a bibliotecas no oficiales. 

Para poder construir una interfaz gráfica profesional en una aplicación desarrollada para una versión anterior a la 3.0 se debe hacer uso de una librería llamada ActionBarSherlock. Para saber en qué consiste esta librería se puede observar la definición de su autor:

“ActionBarSherlock es una extensión de la librería de compatibilidad diseñada para facilitar el uso del patrón de diseño action bar en una única API a través de todas las versiones de Android. La librería utiliza el action bar nativo cuando esté disponible o utiliza una implementación propia sobre los layouts de una forma automática. Esto permite desarrollar fácilmente una aplicación con un action bar para todas las versiones de Android desde la 2.x en adelante”.

**Instalación de ActionBarSherlock**

Para poder usar la biblioteca en un proyecto, primero debe compilar, ya que no se suministra un archivo \*.jar, sino sólo el código fuente. Los pasos a seguir son los siguientes:

1. Descargar el código de la [página web](http://actionbarsherlock.com/) del autor. La última versión disponible.
2. Extraer el archivo comprimido en un directorio temporal. Se crearán tres directorios separados: library, samples y website.
3. Importar el proyecto a Eclipse mediante File -> New -> Android Project from Existing Code. El directorio de origen es el creado en el punto anterior. Si sólo interesa el código de la librería, con seleccionar el directorio “library” es suficiente. Para copiar la librería al workspace de Eclipse se debe seleccionar el checkbox "Copy projects into workspace". Una vez importado el proyecto se puede cambiar el nombre de éste, por ejemplo a “ActionBarSherlock-4.1.0”.
4. Asegurar que el nuevo proyecto está correctamente configurado haciendo click con el botón derecho del ratón y seleccionando "Propiedades". En la nueva ventana, seleccionar "Android" y han de estar marcadas la versión Android 4.0 y el checkBox "Is Library".

|  |
| --- |
| [http://2.bp.blogspot.com/-CrM9b_Sr5KM/UFisxLb3QjI/AAAAAAAABx0/wqpwPDdX26E/s320/ActionBarSherlock-configuracion.png](http://2.bp.blogspot.com/-CrM9b_Sr5KM/UFisxLb3QjI/AAAAAAAABx0/wqpwPDdX26E/s1600/ActionBarSherlock-configuracion.png) |
| **Figura 8: Configuración del proyecto para usar la librería ActionBarSherlock.** |
|  |

1. Tener la biblioteca de compatibilidad instalada. Lo más sencillo es buscar dentro de la carpeta “libs”. A continuación pulsar con el botón derecho del ratón sobre el archivo “android-support-v4.jar” y seleccionar “Build -> Add to build Path”.

|  |
| --- |
| [http://1.bp.blogspot.com/-4fI7uIT_NJk/UFitcwLmiII/AAAAAAAABx8/RQT4O9SvC8Q/s320/ActionBarSherlock-libs.png](http://1.bp.blogspot.com/-4fI7uIT_NJk/UFitcwLmiII/AAAAAAAABx8/RQT4O9SvC8Q/s1600/ActionBarSherlock-libs.png) |
| **Figura 9: Biblioteca de compatibilidad.** |
|  |

De esta forma ya se tiene instalada la biblioteca en Eclipse disponible para futuros proyectos.

**Uso de ActionBarSherlock**

Se debe crear un nuevo proyecto en Eclipse cuya versión de destino mínima sea Android 4.0 (API 14) ya que esta versión es la requerida por ActionBarSherlock.

Más tarde se puede indicar cuál es la versión mínima con la que funcionará una aplicación modificando las siguientes líneas del manifiesto:

Ejemplo Android Manifest:

**<uses-sdk**

**android:minSdkVersion="4"**

**android:targetSdkVersion="15" />**

Una vez creado el proyecto, se selecciona el proyecto y se pulsa el botón derecho del ratón y en Propiedades -> Android, no se marca la opción “Is library” y en cambio, se añade el proyecto que se ha importado anteriormente. En este caso, “ActionBarSherlock-4.1.0”.

|  |
| --- |
| [http://1.bp.blogspot.com/-8vyUhrvNnGI/UFiwpGb2iyI/AAAAAAAAByM/tmaDfY4akCc/s320/ActionBarSherlock-Nuevo.png](http://1.bp.blogspot.com/-8vyUhrvNnGI/UFiwpGb2iyI/AAAAAAAAByM/tmaDfY4akCc/s1600/ActionBarSherlock-Nuevo.png) |
| **Figura 10: Incluyendo la librería ActionBarSherlock.** |
|  |

Modificando el Android Manifest se puede cambiar el tema de la aplicación para que use el de Sherlock :

<application

   android:icon="@drawable/ic\_launcher"

   android:label="@string/app\_name"

   android:theme="@style/Theme.Sherlock"

Ahora se puede ejecutar la aplicación y ver la biblioteca en funcionamiento.

A partir de aquí, se pueden usar otros aspectos de la biblioteca, como pueden ser los fragmentos, listas, entre otros. Si se quiere recuperar la barra desde el código se puede llamar al método getSupportActionbar() en lugar de getActionBar() que es el usado por la API oficial. El resto de métodos son iguales a los de la API oficial.

* 1. **Activity. El uso de fragmentos**

Como se mencionó anteriormente en Android existe un patrón de diseño llamado [ActionBar](https://developer.android.com/design/patterns/actionbar.html) consistente en una barra situada en la parte superior de la aplicación. Ésta la identifica aplicación y permite al usuario saber en qué lugar se encuentra dentro de la misma. También permite situar acciones y modos de navegación fácilmente usables para el usuario. Este patrón de diseño se introdujo con la versión Android 3.0 (Honeycomb - API 11) por lo que no está disponible para versiones inferiores.

También desde la aparición de Android 3.0, se introdujo un nuevo aspecto para las interfaces, el uso de fragmentos (pestañas integradas al actionbar) en esa barra superior que incluyen las App diseñadas a partir de 3.0. Sin embargo, no es necesario excluir a todos los dispositivos que no tengan esa versión de Android. Los que tengan 2.1 en adelante pueden aprovechar de esta característica si se desarrolla usando la librería externa mencionada anteriormente. [La librería Action Bar Sherlock](http://actionbarsherlock.com/) permite la navegación a través de tabs o pestañas (fragmentos).

Un [fragmento](http://www.edu4android.com/es/reference/android/app/Fragment.html) representa un comportamiento o una parte de la interfaz de usuario en una [Actividad](http://www.edu4android.com/es/reference/android/app/Activity.html). Se pueden combinar múltiples fragmentos en una sola actividad para construir una GUI multi-panel y reutilizar un fragmento en múltiples actividades.

Se puede pensar que un fragmento es como una sección modular de una actividad, que tiene su propio ciclo de vida, recibe sus propios eventos inputs, y se puede añadir o eliminar mientras la actividad se está ejecutando.

Un fragmento siempre debe estar dentro de una actividad y el ciclo de vida del fragmento está directamente afectado por el ciclo de vida de la actividad principal. Por ejemplo, cuando la actividad está en pausa, también lo están sus fragmentos y cuando se destruye, también son destruidos sus fragmentos.

Sin embargo, cuando una actividad se está ejecutando, se puede manipular cada fragmento independientemente, por lo que se pueden añadir o eliminar. Cuando se añade un fragmento como parte del layout de la actividad, vive en un [ViewGroup](http://www.edu4android.com/es/reference/android/view/ViewGroup.html) dentro de la jerarquía de una vista de la actividad y define su propio layout de vistas.

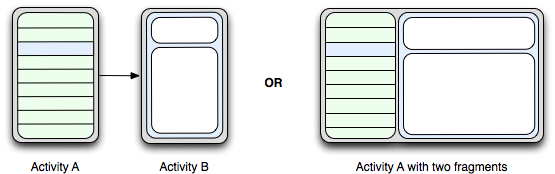
Se puede insertar un fragmento en el layout de la actividad mediante la declaración del fragmento en el layout de la actividad, como un elemento<fragment>, o desde el código de la aplicación, añadiéndolo a un [ViewGroup](http://www.edu4android.com/es/reference/android/view/ViewGroup.html) existente.

Android introdujo fragmentos principalmente para soportar diseños de GUI más dinámicos y flexibles en pantallas grandes, como las tablets. Como una pantalla de un tablet es mucho más grande que la de un teléfono móvil, existe más espacio para combinar e intercambiar componentes de GUI. Los fragmentos permiten estos diseños sin que haya que gestionar cambios muy complejos en la jerarquía de la vista.

Mediante la división del layout de una actividad en fragmentos, se puede modificar la apariencia de la actividad en tiempo de ejecución y preservar esos cambios en un back stack gestionado por una actividad. Por ejemplo, una aplicación sobre noticias puede utilizar un fragmento para mostrar una lista de artículos a la izquierda y otro fragmento para mostrar un artículo a la derecha—los dos fragmentos aparecen en la actividad, juntos, y cada uno de los fragmentos tiene su propio conjunto de métodos callback del ciclo de vida y gestiona sus propios eventos input de usuario.

Así, en vez de utilizar una actividad para seleccionar un artículo y otra actividad para leer el artículo, el usuario puede seleccionar un artículo y leerlo en la misma actividad.

A continuación se muestra un esquema que representa las distintas opciones a la hora de utilizar fragmentos.



**Figura 11:** **Un ejemplo de dos módulos de GUI que típicamente son separados en dos actividades pueden ser combinados en una sola actividad, utilizando fragmentos.[[8]](#footnote-8)**

Un fragmento debería ser un componente modular y reutilizable en la aplicación. Como un fragmento define su propio layout y su propio comportamiento utilizando sus propios callbacks de ciclo de vida, se puede incluir en múltiples actividades. Esto es especialmente importante ya que permite adaptar el uso del usuario a diferentes tamaños de pantalla. Por ejemplo, se puede incluir múltiples fragmentos en una actividad sólo cuando el tamaño de la pantalla es suficientemente grande, y cuando no lo es, se lanzan diferentes actividades que utilizan diferentes fragmentos.

Para seguir con el ejemplo de la aplicación de noticias; la aplicación puede incluir dos fragmentos en la Actividad A, cuando se ejecuta en una pantalla extra grande (una tablet, por ejemplo).

Sin embargo, en una pantalla de tamaño normal (como por ejemplo un teléfono), no hay sitio suficiente para los dos fragmentos, por lo que la Actividad A incluye solo el de la lista de artículos, y cuando el usuario elige un artículo, arranca la Actividad B, que incluye el fragmento que lee el artículo.

* 1. **Recursos**

A la hora de trabajar con recursos, siempre se deben separar los mismos tales como imágenes y cadenas de textos (string) del código de la aplicación, de modo que se puedan mantener de forma independiente. Separarlos permite también proporcionar recursos alternativos que apoyan determinadas configuraciones de dispositivos, que se vuelven cada vez más importante a medida que más dispositivos con Android están disponibles con diferentes configuraciones.

Con el fin de proporcionar compatibilidad con configuraciones diferentes, se debe organizar los recursos en el directorio del proyecto “res /”, utilizando diversos subdirectorios que convoquen recursos por grupo y por tipo de configuración. En tiempo de ejecución, Android utiliza el recurso adecuado, como se dijo anteriormente en función de la configuración actual, entonces es posible, por ejemplo, proporcionar un diseño de interfaz de usuario diferente en función del tamaño de la pantalla o cadenas diferentes en función de la configuración de idioma.

**Agrupando Tipos de Recursos**

Se debería colocar cada tipo de recurso en un subdirectorio especifico del directorio “res” del proyecto.

Como se mencionó anteriormente en la estructura de un proyecto Android, también al trabajar con recursos se debe respetar la estructura del proyecto. Los nombres de los directorios de recursos son importantes y se describen en la siguiente tabla.

|  |  |
| --- | --- |
| Directory | Resources Type |
| animator/ | Archivos XML que definen las animaciones. |
| color/ | Los archivos XML que definen una lista estática de colores. |
| drawable/ | Archivos de mapa de bits (png, jpg, gif....). |
| layout/ | Archivos XML que definen un diseño de interfaz de usuario. |
| menu/ | Archivos XML que definen los menús de aplicaciones, como un menú de opciones, menú contextual, o submenú. |
| raw/ | Archivos arbitrarios para guardarlos en su forma cruda. Para abrir estos recursos hay que llamar a Resources.openRawResource () con el identificador de recurso, que es R.raw.filename. |
| values/ | Archivos XML que contienen valores simples, como cadenas, números enteros, y colores.  Mientras que los archivos de recursos XML en el otro subdirectorio res / definen un recurso único basado en el nombre del archivo XML, los archivos en el directorio values / describen múltiples recursos. Para un archivo en este directorio, cada elemento hijo del elemento <Resources> define un solo recurso. Por ejemplo, un elemento <string> crea un recurso R.string y un elemento <color> crea un recurso R.color.  • strings.xml para valores de cadena.  • styles.xml de estilos. |
| xml/ | Arbitrarias archivos XML que se pueden leer en tiempo de ejecución llamando Resources.getXML () |

**Tabla N°3: Subdirectorios del directorio /res para trabajar con recursos.**

**Proveyendo alternativa de recursos**

Casi todas las aplicaciones deben proporcionar recursos alternativos para soportar las configuraciones de dispositivos específicos. Por ejemplo, debe incluir los recursos disponibles de forma alternativa para diferentes densidades de pantalla y recursos alternativos de cadena para diferentes idiomas. En tiempo de ejecución, Android detecta la configuración actual del dispositivo y carga los recursos adecuados para su aplicación.

Para especificar alternativas de configuración específicas para un conjunto de recursos se debe:

1. Crear un nuevo directorio en /res nombrado de la siguiente manera:

<resources\_name>-<config\_qualifier>.

<resources\_name> es el nombre del directorio de los recursos correspondientes.

<qualifier> es un nombre que especifica una configuración individual para la que estos recursos serán utilizados. Se puede añadir más de un <qualifier>. Se debe separar cada uno con un guión.

1. Guardar los respectivos recursos alternativos en este nuevo directorio:

Por ejemplo, éstos son algunos recursos alternativos:

res/

drawable/

icon.png

background.png

drawable-hdpi/

icon.png

background.png

El calificador hdpi indica que los recursos de ese directorio son para los dispositivos con una pantalla de alta densidad. Las imágenes en cada uno de estos directorios drawable están dimensionadas para una densidad de pantalla específica, pero los nombres de los archivos son exactamente los mismos. De esta manera, el ID de recurso que se utiliza para hacer referencia a la imagen icon.png o background.png es siempre la misma, pero Android selecciona la versión de cada uno de los recursos que mejor coincide con el dispositivo actual, mediante la comparación de la información de configuración del dispositivo con el calificador de clasificación en el nombre del directorio de recursos.

**Accediendo a los recursos**

Una vez que se proporciona un recurso en la aplicación, se puede utilizar haciendo referencia a su identificador de recurso (ID). Todos los identificadores de recursos se definen en la clase R de un proyecto, que la herramienta AAPT los genera automáticamente.

Cuando la aplicación es compilada, AAPT genera la clase R, que contiene identificadores de recursos para todos los ubicados en el directorio /res. Para cada tipo, hay una subclase R (por ejemplo, R.drawable) y para cada recurso de ese tipo, hay un número entero estático (por ejemplo, R.drawable.icon). Éste es el identificador que se puede utilizar para recuperarlo. Hay dos formas de acceder a uno de éstos:

1. En el código: Con el uso de un número entero estático de una subclase de la clase R, tales como R.string.hello, string es el tipo y hola es el nombre del recurso. Hay algunas APIs Android que pueden acceder a los recursos cuando se proporciona un identificador en este formato.
2. En XML: Es necesario usar una sintaxis especial XML que también se corresponde con el ID del recurso definido en la clase R, tal como: @ string / hello, string es el tipo y hola es el nombre del recurso. Puede utilizar esta sintaxis en un archivo XML en cualquier lugar donde se espere un valor que se proporcione en un recurso.

Detalles de las formas de acceder a un recurso:

1. Accediendo desde el código: Se puede utilizar un recurso en código pasando el ID del mismo como un parámetro de método. Por ejemplo, se puede establecer un ImageView que utiliza un recurso ubicado en res / drawable / myimage.png utilizando setImageResource ():

ImageView imageView = (ImageView) findViewById (R.id.myimageview);

imageView.setImageResource (R.drawable.myimage);

También se pueden recuperar los recursos individuales utilizando los métodos de la clase Resources. Se puede obtener una instancia de la misma con Context.getResources ().

1. Accediendo recursos desde XML:

Puede definir los valores para algunos atributos y elementos de XML utilizando una referencia a un recurso existente. A menudo se hace esto al crear archivos de diseño, para abastecer cadenas e imágenes para sus widgets.

Por ejemplo, si agrega un botón , usted debe utilizar un recurso de cadena para el texto del botón:

<Button

android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content"

android:text="**@string/submit**" />

* 1. **Servicios Web**

**¿Qué es un servicio web ?**

La W3C define un Servicio web como un sistema de software diseñado para permitir la interoperatibilidad máquina a máquina en una red.

Para describirlo de una manera más comprensible se puede decir que es una funcionalidad o conjunto de funcionalidades que son publicadas e invocadas a través de la web. Una vez desarrolladas, son instaladas en un servidor y otras aplicaciones pueden descubrirlas desde otros ordenadores de Internet e invocar uno de sus servicios.

Como norma general, el transporte de los datos se realiza a través del protocolo HTTP y la representación de los datos mediante XML. Sin embargo, no hay reglas fijas en los servicios web y en la práctica no tiene por qué ser así.

La ventaja es que se puede utilizar un servicio web sin importar el sistema operativo o el lenguaje en el que fue programado. Además al apoyarse sobre el protocolo HTTP, se puede utilizar los sistemas de seguridad (HTTPS).

**Arquitecturas en los servicios web**

Existen tres enfoques a la hora de definir un servicio web, estos enfoques son lo que se conoce como la arquitectura de un servicio web.

* Llamadas a Procedimientos Remotos (RPC): Se enfoca el servicio web como una colección de operaciones o procedimientos que pueden ser invocados desde una máquina diferente a donde se ejecutan.
* Arquitectura Orientada a Servicios (SOA): En el planteamiento anterior, RPC, la unidad básica de interacción es la operación, en este nuevo planteamiento, la unidad de interacción pasa a ser el mensaje. Por lo tanto, en muchos casos se conoce como servicios orientados a mensajes. Cada uno de los que se utiliza ha de ser definido siguiendo una estricta sintaxis expresada en XML.
* Transferencia de Estado Representacional (REST): Este planteamiento se caracteriza principalmente por su simplicidad. En REST se utiliza directamente el protocolo HTTP, por medio de sus operaciones GET, POST, PUT y DELETE. En consecuencia, esta arquitectura se centra en la solicitud de recursos, en lugar de las operaciones o mensajes de las alternativas anteriores.

**Alternativas para implementar un servicio web**

Un servicio web (web service) es básicamente una función que puede ser accedida vía web por cualquier programa o aplicación sin importar en qué plataforma reside el servicio o en qué lenguaje ha sido desarrollado, el término “web” implica que el acceso se hace mediante una conexión a internet habitualmente vía HTTP aunque otros protocolos de transporte pueden ser utilizados. De esta forma, una aplicación desarrollada en Java puede, por ejemplo, usar una rutina de conversión de euros a dólares que fue realizada en Python y reside en un servidor en cualquier lugar del mundo.

Para que semejante grado de interoperabilidad sea posible es necesario que el “cliente - llamador” y el “servidor – quien responde” que implementa el servicio acuerden el mecanismo utilizado para comunicarse (un protocolo) mediante el cual se define cómo se invoca el servicio, cómo se pasan parámetros, cómo se recibe un resultado, cómo se manejan errores, entre otros. Los dos mecanismos o alternativas dentro del mundo de los servicios web más utilizadas en la actualidad son SOAP (un protocolo en sí mismo) y REST (que hace uso directo de HTTP).

**Servicios web basados en SOAP (Simple Object Access Protocol)**

SOAP es el protocolo más utilizado en la actualidad para implementar servicios Web. Fue creado por Microsoft, IBM, y otros, aunque en la actualidad está bajo el auspicio de la W3C. Es independiente de la plataforma, y del lenguaje.

Este protocolo permite la comunicación entre aplicaciones a través de mensajes enviados por medio de Internet, utiliza HTTP para llevar acabo el envío de los mismos aunque es posible utilizar otros métodos de transporte. Estos mensajes del protocolo se definen utilizando un estricto formato XML, es decir son documentos XML propiamente dichos.

A continuación se muestra la estructura básica de los mensajes del protocolo: 

<?xml version="1.0"?>   
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"   
   Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">   
      <soap:Header>   
         ...   
      </soap:Header>   
      <soap:Body>   
         ...   
         <soap:Fault>   
            ...   
         </soap:Fault>   
      </soap:Body>   
</soap:Envelope>

Explicación del código anterior:

<?xml version="1.0"?>

Como se puede observar en esta línea SOAP es un documento XML, y como tal, debe comenzar con el tag <?xml….?> y la versión correspondiente.

<soap:Envelope

Aquí se indica que comienza el envelope (sobre) del mensaje.

xmlns:soap = "http://www.w3.org/2001/12/soap-envelope"

Un mensaje SOAP debe contener siempre un elemento envelope asociado con el namespace (espacio de nombres) <http://www.w3.org/2001/12/soap-envelope>.

Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

En esta línea lo que se hace es indicar donde se encuentran definidos los tipos de datos utilizados en el documento.

<soap:Header>

Esta línea indica el comienzo del Header (encabezado). En esta sección se incluye información específica del mensaje, como puede ser la autenticación.

</soap:Header>

Como todo documento XML los tags que son abiertos deben ser cerrados, esta línea indica la finalización del Header (encabezado).

<soap:Body>

Aquí comienza el cuerpo del mensaje, en esta sección se incorpora toda la información necesaria para el nodo final. Por ejemplo, los parámetros para la ejecución, o la respuesta a una petición.

<soap:Fault>

Cualquier tipo de fallo que se produzca es notificado en esta sección. La cual está contenida dentro del cuerpo del mensaje.

</soap:Fault>

Cierre de la sección Fault.

</soap:Body>

Indica el final del cuerpo del mensaje.

</soap:Envelope>

Fin del mensaje SOAP.

Como se mencionó anteriormente, SOAP es el protocolo más utilizado en la actualidad para implementar servicios Web. Esto ha llegado a ser así por las siguientes ventajas ofrecidas por el protocolo:

* **No está asociado con ningún lenguaje**: SOAP no especifica ninguna API, por lo que la implementación de la API se deja al lenguaje de programación.
* **No se encuentra fuertemente asociado a ningún protocolo de transporte**: La especificación de SOAP no describe cómo se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
* **No está atado a ninguna infraestructura de objeto distribuido.**

* **Aprovecha los estándares existentes en la industria**: En SOAP se evitaron, intencionadamente, reinventar las cosas. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de codificación de los mismos. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.
* **Permite la interoperabilidad entre múltiples entornos**: Ya que SOAP se desarrolló sobre los estándares existentes de la industria XML y utiliza HTTP.

**Servicios web basados en REST (Representational State Transfer)**

Conviene destacar que el término REST se refiere a una arquitectura en lugar de un protocolo en concreto como es el caso de SOAP. A diferencia del caso de usar SOAP, con REST no se añade una capa adicional a la pila de protocolos, sino que se utiliza directamente el protocolo HTTP.

Siendo estrictos, la arquitectura REST no impone el uso de HTTP, no obstante en la práctica se entiende que un servicio web basado en REST es aquel que se implementa directamente sobre la web.

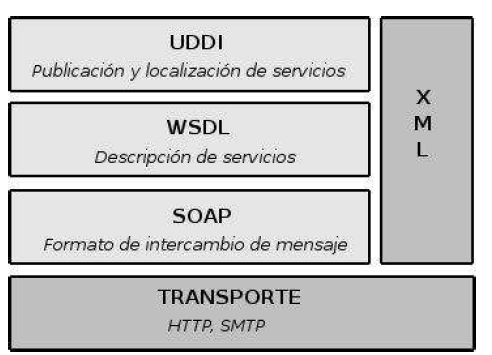
Esta alternativa o mecanismo supone seguir los principios de la WWW, pero en lugar de solicitar páginas web se solicitan servicios web. Los principios de REST son los siguientes:

* El transporte de datos es mediante HTTP, utilizando las operaciones de este protocolo: GET, POST, PUT, DELETE.
* Los servicios son invocados mediante el espacio de URIs.
* El tipo de codificación preferido es XML.

Las ventajas de REST derivan de su simplicidad, mejores tiempos de repuesta y disminución de sobrecarga tanto en el cliente como en el servidor. La gran ventaja es la gran sencillez en el desarrollo de clientes, éstos únicamente han de ser capaces de realizar interacciones HTTP y codificar información en XML.

**Tecnologías utilizadas en el desarrollo de un servicio web avanzado**

A la hora de desarrollar un servicio web es necesario utilizar distintas tecnologías para que éste pueda ser consumido correctamente por un cliente del mismo.



**Figura 12: Tecnologías utilizadas en un servicio web avanzado. [[9]](#footnote-9)**

**Nota: En la imagen se puede observar que la alternativa elegida para implementar el servicio web es SOAP. Esto no significa que la aplicación desarrollada en esta tesina implemente esta alternativa. En ella se lleva acabo la implementación de un servicio web basado en REST ya que esta alternativa cuenta con la gran ventaja de requerir menos recursos. Esto resulta muy adecuado en Android ya que se lograr disminuir la complejidad y sobrecarga, que implican un mayor rendimiento frente a otras alternativas como SOAP. Además Android no incorpora las librerías necesarias para trabajar con SOAP.**

La alternativa Soap ya ha sido explicada anteriormente, por este motivo sólo se explican a continuación las tecnologías restantes en la anterior (figura N° 12)

**UDDI - Universal Discovery Description and Integration**

Es un modelo de directorios para servicios web. Se trata de una especificación para mantener directorios estandarizados de información acerca de los mismos, sus capacidades, ubicación, y requerimientos en un formato reconocido universalmente.

UDDI utiliza WSDL (lenguaje de descripción de servicios web) para describir las interfaces de los servicios web. 

Podemos decir que es un lugar en el cual se puede buscar cuales son los servicios web disponibles, una especie de directorio en el cual se puede encontrar los servicios web publicados y publicar los que se han desarrollados.

**WSDL - Web Services Description Language**

Es un protocolo basado en XML que describe los accesos a los servicios web. Se puede decir que es el manual de operación, porque indica cuáles son las interfaces que provee y los tipos de datos necesarios para la utilización del mismo. El WSDL describe los servicios Web a través de los mensajes que se intercambian entre el proveedor del servicio y el cliente.

A continuación se muestra un ejemplo de la estructura un documento WSDL:

<?xml version="1.0">   
<definitions>   
 <types>   
 ...  
 </types>   
 <message>   
 ...  
 </message>   
 <portType>

...  
 </portType>   
 <binding>   
 ...  
 </binding>   
</definitions>

Explicación del código anterior:

<?xml version="1.0">

Éste es un documento XML, es por esto que debe comenzar con el tag <?xml .. ?>

<definitions>

Comienzo del documento, este tag agrupa a todos los demás.

<types>

Se definen los tipos de datos utilizados en el Web Service.

</types>

Fin de la definición de tipos.

<message>

Se definen los métodos y parámetros para realizar la operación. Cada message puede consistir en una o más partes (parámetros).

</message>

Fin de la definición de los parámetros.

<portType>

Esta sección es la más importante, ya que se definen las operaciones que pueden ser realizadas, y los mensajes que involucran (por ejemplo el mensaje de petición y el de respuesta).

</portType>

Fin de la definición de las operaciones y mensajes.

<binding>

Se definen el formato del mensaje y detalles del protocolo para cada portType.

</binding>

Fin de la definición del formato del mensaje y detalles del protocolo para cada PortType.

</definitions>

Fin del documento WSDL.

**Protocolo HTTP**

Éste se compone de dos mensajes, el primer mensaje es originado por el cliente y es el requisito inicial de la comunicación. Este requerimiento llamado Resquest está divido en una cabecera y un mensaje de texto. En la cabecera el cliente envía información al servidor del recurso solicitado e información de la aplicación cliente que hace la solicitud en el User- Agent, entre otras cosas, como así también los datos de la plataforma en la cual se encuentra corriendo la aplicación cliente.

Una vez recibido este requerimiento el servidor devuelve una repuesta llamada Response la cual también está compuesta por una cabecera y un mensaje. Lo importante de la cabecera de respuesta está en donde se informa el estado del mensaje, donde 200 significa estado exitoso.

**Pasos para consumir el servicio web**

Los pasos que se realizan al consumir un servicio web son:

1. Lo primero que realiza el cliente al hacer una solicitud al servicio es encontrar algún servidor mediante UDDI, una vez realizado esto luego es posible tomar la definición del archivo WSDL.
2. El servidor entrega el fichero WSDL. Este archivo indica a la petición los métodos y propiedades de ese servicio que están disponibles.
3. El cliente hace la petición en el formato que espera el servidor según las especificaciones del fichero WSDL en el que se dice qué parámetros acepta y de qué tipo.
4. El servidor entrega el resultado de la consulta.
   1. **Tratamiento de XML**

A la hora de desarrollar una aplicación para un dispositivo móvil es muy común que ésta demande el intercambio de información con aplicaciones externas al dispositivo. Dada la gran interacción que se da en estos días entre aplicaciones y redes sociales, aplicaciones y servicios web; es necesario un mecanismo para la codificación de la información intercambiada por éstas.

XML es uno de los estándares más utilizados en la actualidad para codificar información. Es ampliamente utilizado en Internet para el intercambio de información y además es utilizado para múltiples usos en el SDK de Android. Por ejemplo para definir layouts, animaciones, y para confeccionar el archivo principal de una aplicación Android: el AndroidManifest.

Como ya se ha visto la plataforma Android aprovecha el lenguaje de programación Java y sus librerías. El SDK de Android no acaba de ofrecer todo lo disponible en el estándar del entorno de ejecución de Java (JRE), pero es compatible con una fracción muy significativa del mismo. Esto ocurre en lo referente a trabajar con XML, Java dispone de gran cantidad de APIs con este propósito pero no todas están disponibles en Android.

**Estructura**

Antes de analizar cuáles son los modelos existentes para poder leer y escribir un documento XML, es necesario ver y entender su estructura:

**Encabezado, en él se podrá especificar:**

versión del XML que se está utilizando.

tipo de codificación que la aplicación utiliza para interpretar el código XML.

**Cuerpo, el cuerpo está compuesto por un nodo raíz y uno o más nodos elementos los cuales pueden poseer atributos.**

Ejemplo de código XML:

**<?**xml **versión=** “1.0” **enconding=** “UTF-8”**?>**

<lista\_alumnosUCA> **// Nodo raíz**

<alumno condicion=”regular”, estado=”promociono”> **// Nodo elemento con dos atributos condición y estado. Y sus respectivos valores regular y promociono**

<nombre> Javier Cocco </nombre>

</alumno>

<alumno condicion=”no regular”, estado=”recuperatorio”>

<nombre> Luis Cocco </nombre>

</alumno>

</lista\_alumnosUCA> **// Fin del nodo raíz**

En este simple código XML se puede observar que éste contiene información sobre alumnos de la UCA. También se puede observar un nodo raíz (lista\_alumnosUCA) con dos nodos elementos, los cuales poseen dos atributos condición y estado. Cada nodo elemento posee un sub nodo elemento nombre.

**Modelos**

Al día de hoy, en casi todas las grandes plataformas de desarrollo existen varias formas de leer y escribir datos en formato XML. Los dos modelos más extendidos son SAX (Simple API for XML) y DOM (Document Object Model). Posteriormente, han ido apareciendo otros tantos, con más o menos éxito, entre los que destaca StAX (Streaming API for XML). Pues bien, Android no se queda atrás en este sentido e incluye estos tres modelos principales para el tratamiento de XML, vale aclarar, los dos primeros como tal y una versión análoga del tercero (XmlPull). Por supuesto con cualquiera de los tres modelos se puede hacer las mismas tareas, dependiendo de la naturaleza de la tarea que se quiera realizar resulta más eficiente utilizar un modelo u otro.

Teniendo en cuenta estos modelos, las librerías disponibles en Android son:

* (SAX) Java’s Simple API for XML (en Android ofrecido por el paquete org.xml.sax).
* (DOM) Document Object Model (en Android ofrecido por el paquete org.w3c.dom).
* XmlPull equivalente a StAX (Streaming API for XML). (en Android ofrecido por el paquete org.xmlpull.v1.XmlPullParser).

A continuación se propone la explicación de cómo leer y escribir información en formato XML en cada una de estas librerías y en el capítulo cinco de esta tesina, Aplicando Conceptos y Soluciones de Desarrollo, se detalla la aplicación de cada una de estas librerías.

**Procesando XML con SAX**

El uso de esta API se recomienda cuando se desea un programa de análisis rápido y se quiere reducir al mínimo el consumo de memoria de la aplicación. Esto se hace muy apropiado para un dispositivo móvil con Android. También es recomendable para procesar archivos de gran tamaño.

En el modelo SAX, el tratamiento de XML se basa en un analizador (parser) que a medida que lee secuencialmente el documento XML va generando diferentes eventos con la información de cada elemento leído. Así, por ejemplo, a medida que lee el XML, si encuentra el comienzo de una etiqueta <title> ésta genera un evento de comienzo de etiqueta, startElement(), con su información asociada, si después de esa etiqueta se encuentra un fragmento de texto se genera un evento characters() con toda la información necesaria, y así sucesivamente hasta el final del documento.

Entonces para analizar un documento XML con SAX, se debe escribir una serie de métodos asociados a cada tipo de evento que se puede generar. A medida que se va leyendo el código XML se producen los eventos y se llaman a estos métodos asociados. Como se dijo anteriormente a medida que se van encontrando distintas etiquetas se van generando los distintos eventos.

En estos métodos se puede leer la información asociada al evento y almacenarla en algún lugar de interés ya que SAX no almacena los datos en una estructura propia por lo tanto es necesario contar con una estructura propia de datos donde guardar los datos leídos. Además en estos métodos asociados a los eventos no sólo se puede leer los datos asociados al mismo si no también realizar acciones (es decir llamar métodos propios desarrollados por el programador).

Para poder definir el comportamiento de estos métodos es necesario extender la clase DefaultHandler (es decir Un Manejador de eventos, que capturara los eventos generados al analizar el código XML el analizador o parse)que permite sobrescribir 5 métodos.

* **startDocument():** Método llamado cuando se produce el evento de Comienzo del Documento XML.
* **endDocumment():** Método llamado cuando se produce el evento de finalización de un documento XML.
* **startElement(String uri, String nombreLocal, String nombreCualif, Attributes atributos):** Método llamado cuando se produce el evento de comienzo de una nueva etiqueta.

**Lista de parámetros:**

* + - * **uri:** La uri del espacio de nombres o vacío, si no se ha definido**.**
      * **nombreLocal:** Nombre local de la etiqueta sin prefijo.
      * **NombreCualif:** Nombre local de la etiqueta con prefijo.
      * **Atributos:** Lista de atributos de la etiqueta.
* **endElement(String uri, String nombreLocal, String nombreCualif):** Método llamado cuando se produce el evento de fin de una etiqueta.
* **Characters(char ch[], int comienzo, int longitud):** Método llamado cuando se produce el evento de encontrar caracteres dentro de una etiqueta. Es decir <etiqueta> caracteres <etiqueta>. Este método devolvería caracteres.

**Procesando XML con DOM**

Anteriormente en SAX se vio que con dicho modelo el tratamiento del código XML se realizaba de forma secuencial, es decir, se iban realizando las acciones necesarias durante la propia lectura del documento. Sin embargo, con DOM la estrategia cambia radicalmente. Con DOM, el documento XML se lee completamente antes de poder realizar alguna acción en función de su contenido. Esto es posible gracias a que, como resultado de la lectura del documento, el parser DOM devuelve todo su contenido en forma de una estructura de tipo árbol, donde los distintos elementos del XML se representa en forma de nodos y su jerarquía padre-hijo se establece mediante relaciones entre dichos nodos.

Ejemplo de código XML:

**<?**xml **versión=** “1.0” **enconding=** “UTF-8”**?>**

<lista\_alumnosUCA> **// Nodo raíz**

<alumno condicion=”regular”, estado=”promociono”> **// Nodo elemento con dos atributos condición y estado. Y sus respectivos valores regular y promociono**

<nombre> Javier Cocco </nombre>

<carrera> Sistemas</carrera>

</alumno>

<alumno condicion=”no regular”, estado=”recuperatorio”>

<nombre> Luis Cocco </nombre>

<carrera> Computación</carrera>

</alumno>

</lista\_alumnosUCA> **// Fin del nodo raíz**

Este XML se traduciría en un árbol parecido al siguiente:

Lista\_alumnosUCA

alumno

alumno

nombre

carrera

nombre

carrera

Javier Cocco

Sistemas

Luis Cocco

Computacion

**Figura 13: Árbol resultante del procesamiento del código XML.**

Se puede observar que este árbol conserva la misma información contenida en el fichero XML pero en forma de nodos y transiciones entre nodos, de forma que se puede navegar fácilmente por la estructura. Además, este árbol se conserva persistente en memoria una vez leído el documento completo, lo que permite procesarlo en cualquier orden y tantas veces como sea necesario (a diferencia de SAX, donde el tratamiento era secuencial y siempre de principio a fin del documento, no pudiendo volver atrás una vez finalizada la lectura del XML).

**Tratamiento de XML con XmlPull**

Los modelos SAX y DOM son los dos métodos más comunes de lectura de XML soportados en la plataforma Android. Existe un método menos conocido aunque igual de válido según el contexto de la aplicación, llamado XmlPull. Este método es una versión similar al modelo StAX (Streaming API for XML), que en esencia es muy parecido al modelo SAX porque también se basa en definir las acciones a realizar para cada uno de los eventos generados durante la lectura secuencial del documento XML. La diferencia radica principalmente en que, mientras que en SAX no se tenía el control sobre la lectura del XML una vez iniciada (el parser lee automáticamente el XML de principio a fin generando todos los eventos necesarios), en el modelo XmlPull se puede guiar o intervenir en la lectura del documento, siendo el desarrollador quien vaya pidiendo de forma explícita la lectura del siguiente elemento del XML y respondiendo al resultado ejecutando las acciones oportunas.

* 1. **Almacenamiento de datos.** **Base de datos SQLITE**

En la mayoría de los casos una aplicación necesita almacenar información de manera permanente. Las alternativas más habituales para conservar esta información son los ficheros, las base de datos o servicios a través de la red. En Android cualquier repositorio de datos es privado y únicamente accesible por la aplicación a la que pertenece. Sin embargo, esto no implica necesariamente que no pueda ser compartido o accedido por otras aplicaciones; Android facilita una serie de mecanismos que posibilitan en ciertas circunstancias el acceso a dicha información.

Ante estas alternativas de almacenamiento propuestas por Android, en esta tesina se ha hecho la elección de uso de SQLite. SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre. Esta elección es dado a que Android incorpora la librería SQLite en forma nativa, que permite crear y manipular bases de datos de forma muy sencilla.

**Base de datos SQLite**

Android incluye la librería de SQLite que permite crear bases de datos relacionales, navegar entre las tablas, ejecutar sentencias en SQL y otras funcionalidades propias del sistema SQLite. La base de datos resultante puede ser accedida desde el código de la aplicación como si se tratara de un objeto más, gracias a las clases contenidas en el paquete Android.database.sqlite.

Cualquier base de datos es accesible desde los demás componentes de la misma aplicación, pero no fuera de ésta.

**¿Cómo trabajar con una base de datos SQLite?**

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite es a través de una clase auxiliar llamada SQLiteOpenHelper, o para ser más exacto, de una clase propia que derive de ella y que se debe personalizar para adaptar a las necesidades concretas de la aplicación. Esta clase posee dos métodos abstractos, onCreate() y onUpgrade(), que se debe personalizar con el código necesario para crear la base de datos y para actualizar su estructura respectivamente.

El método onCreate() es ejecutado automáticamente por la clase que hereda de SQLiteOpenHelper cuando sea necesaria la creación de la base de datos, es decir, cuando aún no exista. Las tareas típicas que deben hacerse en este método son la creación de todas las tablas necesarias y la inserción de los datos iniciales si son necesarios.

Por su parte, el método onUpgrade() es lanzado automáticamente cuando sea necesaria una actualización de la estructura de la base de datos o una conversión de los datos.

Una vez que se tiene una referencia a la clase que herede de SQLiteOpenHelper, la apertura de la base de datos desde la aplicación resulta ser algo de lo más sencillo. Lo primero es crear un objeto de la clase que herede de SQLiteOpenHelper al que se le pasa el nombre de la base de datos, y por último la versión de la base de datos que se necesita. La simple creación de este objeto puede tener varios efectos:

* Si la base de datos ya existe y su versión actual coincide con la solicitada simplemente se realiza la conexión con ella.
* Si la base de datos existe pero su versión actual es anterior a la solicitada, se llama automáticamente al método onUpgrade() para convertir la base de datos a la nueva versión y se conecta con la base de datos convertida.
* Si la base de datos no existe, se llama automáticamente al método onCreate() para crearla y se conecta con la base de datos creada.

Una vez que se tiene una referencia al objeto de tipo de la clase que extienda a SQLiteOpenHelper, se llama a su método getReadableDatabase() o getWritableDatabase() para obtener una referencia a la base de datos, dependiendo de si sólo se necesita consultar los datos o también realizar modificaciones, respectivamente.

Ahora que ya se ha conseguido una referencia a la base de datos (objeto de tipo SQLiteDatabase) ya se puede realizar todas las acciones que sean necesarias sobre ella. Para llevar acabo estas acciones se puede utilizar el método execSQL() con las sentencias correspondientes.

Por último se debe cerrar la conexión con la base de datos llamando al método close().

CAPÍTULO IV: LOCALIZACIÓN Y POSICIONAMIENTO EN ANDROID

* 1. **Introducción**

En este capítulo se explica todo lo relacionado a la localización de un dispositivo, a la ubicación en mapa de distintos puntos de interés, al uso de mapas y a los servicios web del API de Google Maps.

* 1. **Tecnologías**

Las tecnologías de localización en los móviles permiten conocer la ubicación donde se encuentra un dispositivo en un momento a través de la triangulación de la posición de los satélites captados (en el caso de utilizar tecnología GPS) y proporcionarla a través de los datos de longitud, latitud, y altitud. Estos datos son calculados a través de la distancia entre el receptor y cada uno de los satélites.

Éstas son una combinación de hardware (por ejemplo antenas de GPS) y software (cálculo de triangulaciones en función de la señal recibida de distintas antenas GSM). Cada tipo posee ventajas e inconvenientes, en términos de exactitud, velocidad de localización y consumo de batería, entre otros.

Los receptores GPS pueden recibir, y habitualmente lo hacen, la señal de más de tres satélites (entre ocho y doce) para calcular su posición. En principio, cuantas más señales reciben, más exacto es el cálculo de su posición.

La plataforma Android cuenta con tecnología de localización y de posicionamiento mediante el uso de:

* Información recibida de las torres de telefonía celular y de puntos de acceso WIFI.
* Información recibida del sistema global basado en GPS. Éste sólo funciona si se dispone de visibilidad directa de los satélites.

Un complemento ideal para las tecnologías de localización y posicionamiento es la utilización de mapas, los cuales son utilizados en la aplicación desarrollada en esta tesina mediante el uso de la API Android de Google Maps.

* 1. **La API Android de Google Maps**

Esta API incluye todas las clases relacionadas con la carga y manejo de mapas directamente desde Google Maps. Dentro de ella se pueden encontrar las siguientes clases:

* 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 de un mapa, según su latitud y longitud.
* Overlay: Permite dibujar y representar elementos sobre un mapa.
* MapActivity: Clase muy relevante que extiende la clase base Activity, y permite crear una Activity específica para gestionar mapas.

Un paso previo a la utilización de esta API para mostrar y manejar mapas es el registro en Google Maps y la aceptación de sus términos y condiciones de uso. Este registro se realiza a través de una clave denominada API key.

Una vez que se ha completado el registro, la utilización de mapas en la aplicación no requiere más que unas cuantas llamadas a las clases pertinentes.

**Obtención de la API Key**

Como ya se mencionó, para utilizar los servicios de Google Maps en una aplicación Android es imprescindible realizar una serie de pasos previos, que implican el registro del desarrollador y la aceptación de las condiciones de uso. De esta forma, Google quiere asegurar que se hace en todo momento un uso adecuado de los servicios y datos que proporciona con Google Maps.

Toda aplicación Android está acompañada de un certificado que asegura su autoría y la vincula con su desarrollador. Si se utiliza el plug-in de Android para Eclipse, todas las aplicaciones están firmadas por un certificado al que se puede calificar de prueba y que permite a los desarrolladores crear y probar sus aplicaciones sin más esperas. Este certificado se obtiene a través del fichero de claves de prueba “debug.keystore”, presente por defecto en el SDK de Android. Se debe recordar que si un desarrollador desea distribuir su aplicación de forma pública o hacerla accesible desde los servicios de descarga, como por ejemplo Android Market, es necesario que se cree su propio fichero individual de claves “.keystore” con el que se puede crear un certificado y firmar su aplicación.

El registro para tener acceso a Google Maps se hace a través de dicho certificado. En concreto el proceso es el siguiente, utilizando el fichero “debug.keystore”. En caso de disponer un fichero de claves propio, el proceso sería el mismo cambiando los parámetros necesarios.

1. En primer lugar, se ha de obtener un resumen MD5 del certificado que se debe usar para firmar la aplicación donde se hará uso de mapas. El resumen puede generarse, por ejemplo, con la herramienta keytool presente en el SDK de Java de la siguiente manera:

>keytool -list -alias androiddebugkey –keystore

C:\debug.keystore -storepass android -keypass android

Huella digital de certificado (MD5):

0B:41:31:DC:4B:89:22:8E:A5:45:79:C6:13:DA:7E:D4

2. A continuación, se debe registrar el resumen MD5 obtenido en Google Maps. Para ello, se visita la web de registro, se aceptan los términos de uso y se envía el resumen. Es necesario disponer de una cuenta Google activa.

3. Google Maps proporciona entonces una clave alfanumérica, llamada API key. Ésta es la que se debe utilizar para obtener y manejar mapas, siendo además única y exclusiva para la aplicación firmada con el certificado utilizado.

Tu clave es: 0sO8gCzgaeJ3QpNmuPTX3pwzQ7m6vD3\_tV2CY4w

Esta clave es válida para todas las aplicaciones firmadas con el certificado cuya huella dactilar sea: 0B:41:31:DC:4B:89:22:8E:A5:45:79:C6:13:DA:7E:D4

**Mostrar un mapa**

Una vez que se cuenta con la clave de uso de la API, la inclusión de mapas en una aplicación es una tarea relativamente sencilla y directa.

Para incluir un mapa de Google Maps en una ventana de una aplicación se debe utilizar el control MapView. Este control se puede añadir al layout de la ventana como cualquier otro control, tan solo se debe tener en cuenta que es necesario indicar la clave de uso de Google Maps en su propiedad android: apiKey.

A continuación se muestra un fragmento de código en el cual se ha especificado esta clave de uso.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<com.google.android.maps.MapView

android:id="@+id/mapa"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

**android:apiKey="0ss-5q6s3FKYkkp3atMUH..."**

android:clickable="true" />

</LinearLayout>

Se puede observar, que además de la propiedad apiKey, también se ha establecido la propiedad clickable con valor true, de forma que se pueda interactuar con el control mediante gestos con el dedo (por ejemplo, para desplazar el mapa).

Este tipo de control tiene la particularidad de que sólo puede ser añadido en una actividad de tipo MapActivity. A continuación se muestra un fragmento de código que hace uso de un mapa:

package net.sgoliver.android;

import com.google.android.maps.MapActivity;

import com.google.android.maps.MapView;

import android.os.Bundle;

public class AndroidMapas extends MapActivity {

private MapView mapa = null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

mapa = (MapView)findViewById(R.id.mapa);

mapa.setBuiltInZoomControls(true);

}

@Override

protected boolean isRouteDisplayed() {

return false;

Se puede observar, que si una clase hereda de MapActivity se debe implementar obligatoriamente el método isRouteDisplayed(), cuyo valor de retorno debe ser true sólo en caso de que se vaya a representar algún tipo de información de ruta sobre él.

Además de ésto, en el método onCreate() se llama al método setBuiltInZoomControls() sobre la variable de referencia al control MapView para mostrar los controles de zoom estándar sobre el mapa, de forma que se pueda acercar y alejar la vista del mapa. Con esto, ya se dispone de un mapa completamente funcional en una aplicación.

Para ejecutar la aplicación es necesario realizar un paso más, modificar el fichero AndroidManifest.xml. Por un lado, se especifica que se hace uso de la API de Google Maps mediante la cláusula <uses-library> y en segundo lugar que se habilitan los permisos necesarios para acceder a Internet mediante la cláusula <uses-permission>.

En el siguiente fragmento de código se puede observar el fichero AndroidManifest resultante tras añadir estas dos líneas:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="net.sgoliver.android"

android:versionCode="1"

android:versionName="1.0">

<uses-sdk android:minSdkVersion="7" />

<application

android:icon="@drawable/icon"

android:label="@string/app\_name">

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

<activity android:name=".AndroidMapas"

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" />

</manifest>

* 1. **Overlays (capas de información propias sobre mapas), itemizedoverlay y overlayitem**

Para añadir información personalizada sobre un mapa se necesita incorporar sobre éste una nueva capa, donde se dibuja el elemento de información adicional que se desee mostrar. Se puede incluir cualquier tipo de información en estas nuevas capas como rutas y notas de texto, entre otras.

Se puede definir una nueva capa de información creando una clase Java que derive de la clase Overlay. En ésta se tiene que sobrescribir el método draw(), en el cual se debe dibujar toda la información a incluir sobre el mapa. Se pueden añadir varios Overlays.

El método draw() recibe un objeto Canvas de parámetro, en el que se puede dibujar directamente utilizando los métodos de dibujo drawLine(), drawCircle() y drawText().

Sin embargo, a todos estos métodos de dibujo hay que indicarles las coordenadas en píxeles relativos al mapa en el que se está dibujando. En cambio, lo que se posee desde un principio al obtener una ubicación son coordenadas en forma de latitud y longitud expresadas en grados.

Para traducir entre unas unidades y otras con el fin de saber en qué posición del mapa dibujar los elementos a mostrar, Android proporciona la clase Projection con la cual se puede realizar conversiones precisas entre ambos sistemas de referencia.

**ItemizedOverlay y OverlayItem**

**ItemizedOverlay**

Es recomendable utilizarla clase Overlay a la hora de dibujar líneas, círculos y mostrar un cuadro con texto sobre un mapa. A la hora de mostrar distintas ubicaciones sobre un mapa mediante algún marcado es recomendable utilizar una clase denominada ItemizedOverlay que consiste en una lista de elementos.

Ésta se encarga de ordenar, de dibujar un marcador para cada elemento, de mantener el foco sobre alguno de éstos y de realizar alguna acción cuando se presiona sobre alguno de éstos.

ItemizedOverlay es otra subclase de Overlay lo que permite que se pueda dibujar sus elementos en un mapa y que se puedan añadir a éste, está formada por un conjunto de elementos u objetos del tipo OverlayItems que representan los elementos a dibujar en el mapa.

# Servicios web del API

El sitio de desarrolladores de Google describe los servicios web de la API de la siguiente manera:

El API de Google Maps proporciona estos servicios web como una interfaz para solicitar a servicios externos datos del API de Google Maps y utilizarlos en las aplicaciones. Estos servicios se han diseñado para utilizarse en combinación con un mapa. Se consumen mediante solicitudes HTTP a URL específicas, transmitiendo a los servicios parámetros de URL como argumentos.

Normalmente, estos servicios devuelven los datos de la solicitud HTTP en formato JSON o XML para que la aplicación los analice o procese.

Las solicitudes de servicios web suelen tener el siguiente formato:

http://maps.googleapis.com/maps/api/service/output?parameters

El servicio concreto a consumir se debe indicar mediante el argumento service y el formato de respuesta mediante el argumento output que será normalmente JSON o XML.

**Servicios ofrecidos por el API de Google Maps**

**API de rutas (utilizada en la aplicación)**

El API de rutas es un servicio que utiliza una solicitud HTTP para calcular rutas para llegar de una ubicación a otra. Para determinar las rutas se deben especificar los orígenes y los destinos expresados como coordenadas de latitud/longitud.

Una solicitud del API de rutas tiene el siguiente formato:

http://maps.googleapis.com/maps/api/directions/output?parameters

[**API de matriz**](https://developers.google.com/maps/documentation/distancematrix?hl=es) **(utilizada en la aplicación)**

El API de matriz de distancia es un servicio que proporciona el tiempo y la distancia de viaje para una matriz de orígenes y destinos.

La información devuelta se basa en la ruta recomendada entre los puntos de partida y llegada, según los cálculos del API de Google Maps, y se compone de dos filas que incluyen los valores de duración y de distancia para cada par.

Una solicitud del API de matriz de distancia tiene el siguiente formato:

http://maps.googleapis.com/maps/api/distancematrix/output?parameters

[**API de elevación**](https://developers.google.com/maps/documentation/elevation?hl=es) **(No utilizada en la aplicación)**

Esta API no es comentada debido a que no es utilizada.

[**API de codificación geográfica**](https://developers.google.com/maps/documentation/geocoding?hl=es) **(Utilizado en la aplicación)**

El API de codificación geográfica es la encargada de llevar adelante el proceso de codificación. Este proceso es el encargado de transformar direcciones ("1600 Amphitheatre Parkway, Mountain View, CA") en coordenadas geográficas (37.423021 de latitud y -122.083739 de longitud), que se pueden utilizar como dato para colocar marcadores en un determinado punto de un mapa.

Además, el servicio permite realizar la operación contraria (convertir coordenadas en direcciones), este proceso se conoce con el nombre de "codificación geográfica inversa".

**Nota: Este servicio en la aplicación desarrollada es utilizado indirectamente, es decir no se lo consume directamente mediante una solicitud web, sino mediante el uso de una clase proporcionada en el SDK de Android que se encarga de consumir el servicio ella misma y manejar todo lo relacionado con el mismo internamente.**

Las solicitudes del API de codificación geográfica deben tener el siguiente formato:

http://maps.googleapis.com/maps/api/geocode/output?parameters

[**API de Google Places**](https://developers.google.com/maps/documentation/places?hl=es) **(Utilizado en la aplicación)**

Otro servicio de alto interés que se puede consumir de Google es Google Places:

**¿Qué es Google Places?**

Google ha lanzado un directorio para que los dueños de empresas registren sus negocios de forma gratuita y puedan aparecer en los resultados de **Google Maps**.

**Google Places** ofrece crear un registro gratuito de una empresa en los mapas de Google (**Google Maps**), para que de esta forma cuando los usuarios de Google hagan búsquedas locales como por ejemplo pizzerías en el centro, florerías en Maipú, entre otras, puedan encontrar un negocio (el que se registró) visualizando su ubicación, teléfono, horario de atención y fotos.

Son miles las personas que usan **Google Maps** para decidir dónde comprar, dónde ir o en qué lugar encontrar lo que buscan. Al registrar una empresa o negocio con **Google Places,** éste puede ser uno de estos lugares.

Google también ofrece un servicio web para consumir esta información mencionada anteriormente, mediante la API de Google Places.

Una solicitud de búsqueda de sitio es una URL HTTP con el siguiente formato:

http: //maps.googleapis.com/maps/api/place/search/output?parameters

A esta URL se le puede agregar un parámetro radius, el cual define la distancia (en metros) dentro de la que se deben mostrar resultados de sitios. El radio máximo permitido es de 50.000 metros.

Este parámetro es muy utilizado en la aplicación desarrollada en esta tesina con un valor mínimo de manera de permitirle al usuario conseguir un listado de sitios próximos a su ubicación actual.

* 1. **Geocodificación y geocodificación inversa**

Debido a que la información obtenida sobre la ubicación del dispositivo está expresada en coordenadas, para poder mostrarle al usuario los datos de la dirección de esta ubicación (es decir nombre de calle, departamento, ciudad, entre otros) es necesario llevar acabo un proceso que transforme las coordenadas en direcciones. Para esto existe una clase en Android que permite realizar geocodificación inversa.

La geocodificación inversa es el proceso de transformar coordenadas (latitud, longitud) en una dirección. La cantidad de detalles obtenidos de este proceso de geocodificación inversa puede variar, por ejemplo, el resultado podría contener la dirección postal completa del edificio más cercano, mientras que otro resultado podría contener solamente un nombre de ciudad y el código postal.

Para llevar acabo estos procesos se debe utilizar la clase Geocoder ofrecida por Android, cabe destacar que la clase Geocoder realiza la conexión con un servicio de back-end que no está incluido en el framework central de Android, la conexión es con un servicio web de la API de Android Maps. Los métodos de consulta de la clase Geocoder devuelven una lista vacía si no se logra la conexión con el servicio. Se debe destacar que la cantidad de detalles obtenidos puede variar según el lugar y que también es posible realizar el proceso de geocodificación con esta clase, que es el proceso de transformar una dirección de un lugar en coordenadas (latitud y longitud).

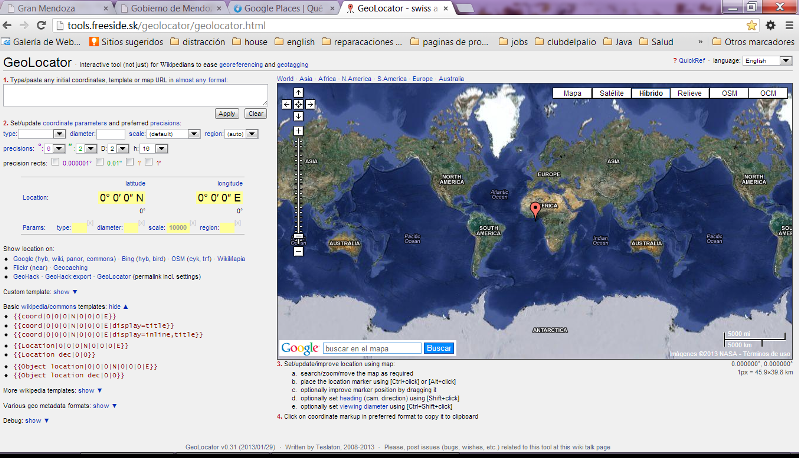
El uso de esta clase es explicado en el capítulo 5.

Como final de este capítulo se menciona la manera en que se han conseguido las coordenadas necesarias, expresadas en latitud y longitud, de los distintos sitios y lugares mostrados en cada uno de los mapas de la aplicación.

* 1. **Obtención de coordenadas**

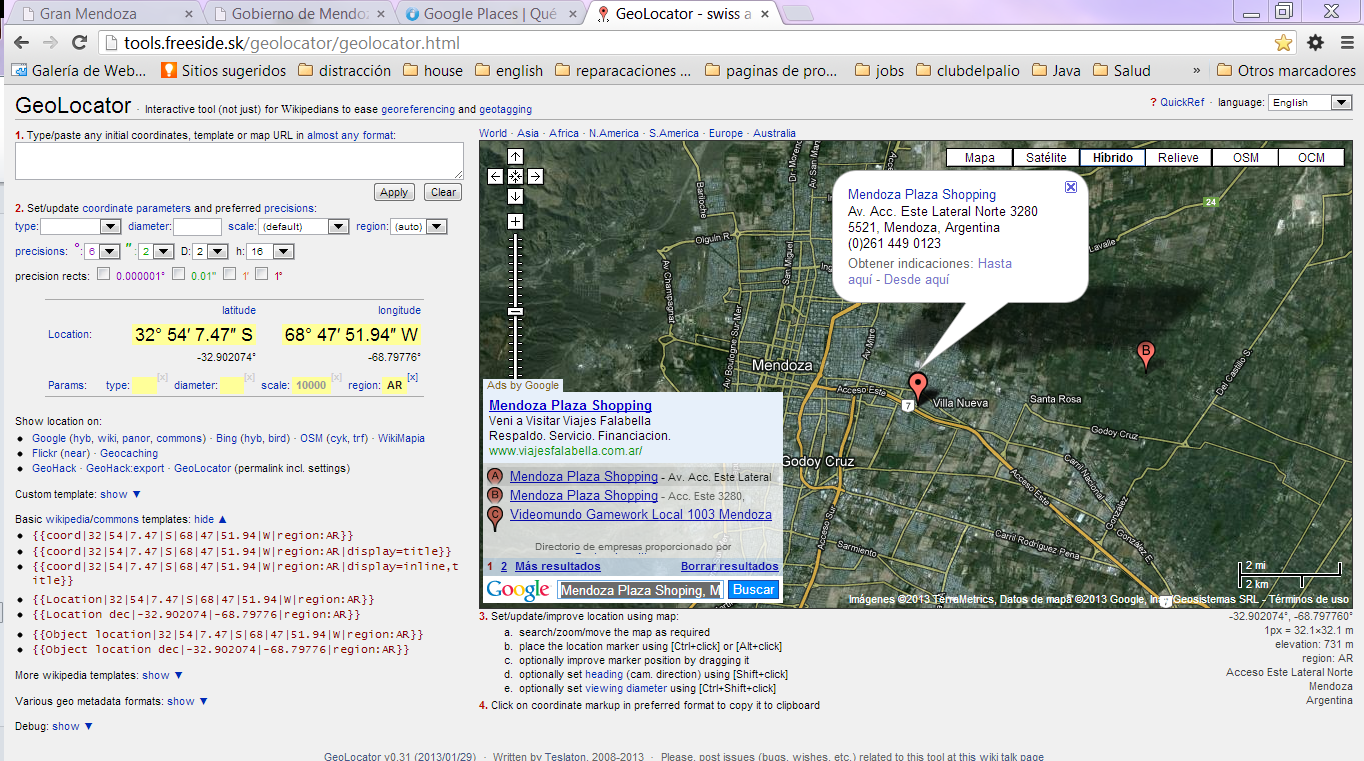
Ya se ha visto anteriormente que a la hora de crear un punto que represente un lugar o una entidad en un mapa es necesario instanciar un objeto GeoPoint, el cual requiere para su creación un conjunto de coordenadas expresadas en latitud y longitud. En base a estas coordenadas el GeoPoint puede ser ubicado. Estas coordenadas han sido necesarias para representar hoteles, bodegas, paradas y recorridos del micro (un conjunto aproximado de 100 a 150 de ellas) como así también los lugares ofrecidos por el servicio de Google Places.

Para obtener cada una de ellas se ha consultado el sitio <http://tools.freeside.sk/geolocator/geolocator.html>.



**Figura 14: Sitio Web GeoLocator. Sitio Web de GeoLocator. Utilizado para obtener las coordenas de las distintas entidades.**

En él se puede ingresar la dirección de un lugar, por ejemplo el Km0 de la ciudad de Mendoza de la siguiente manera “San Martín y Vicente Zapata, Mendoza” y dará las coordenadas para poder ubicar el GeoPoint en un mapa. También se puede ingresar el nombre de alguna entidad de lasiguiente manera, por ejemplo “Mendoza Plaza Shoping, Mendoza” y luego se obtendrán las coordenadas exactas del lugar.



**Figura 15: Sitio Web de GeoLocator. Resultados .A la izquierda de la imagen se visualizan las coordenadas de la entidad ingresada: -32.902074 y -68.79776. A la derecha los datos relacionados a la entidad como nombre, dirección y teléfono.**

CAPÍTULO V: APLICANDO CONCEPTOS Y SOLUCIONES EN EL DESARROLLO

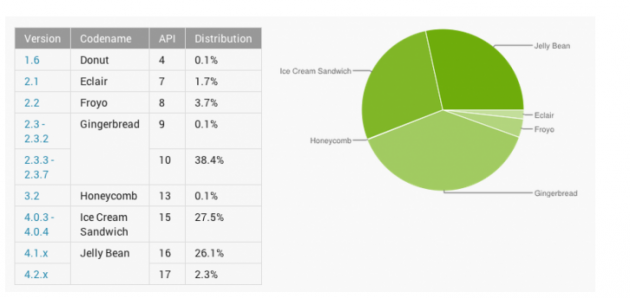
* 1. **Introducción**

En este último capítulo se muestra de qué modo se han implementado los conceptos y las soluciones relacionadas al desarrollo ya explicadas en los capítulos anteriores de esta tesina, con el objetivo de mostrar en qué partes de la aplicación se ha llevado acabo esta tarea y sus resultados.

* 1. **Elección de la plataforma**

En el capítulo anterior se mencionaron las distintas plataformas de desarrollo actuales de Android, para el desarrollo de la aplicación detallada en este capítulo se ha utilizado la plataforma Gingerbread de Android 2.3.

La elección de esta plataforma se ha sustentado en el hecho de que ésta es la que se encuentra en la mayoría de los dispositivos móviles actuales.

****

**Figura 16: Presencia de las distintas plataformas Android en dispositivos móviles.[[10]](#footnote-10)**

Se puede observar que la plataforma Gingerbread 2.3.x está presente casi en un 39 % de los dispositivos móviles con Android. Es la versión de Android más utilizada en la actualidad.

* 1. **Aplicación de Asynctask**

Todos los componentes de una aplicación Android, tanto las actividades, los servicios, o los broadcast receivers se ejecutan en el hilo principal que es donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Es por ello, que cualquier operación larga o costosa que se realice en este hilo puede bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz. Para resolver este problema se utiliza la clase AsyncTask que permite realizar operaciones en segundo plano y luego publicar los resultados en el hilo de la interfaz de usuario sin tener que manipular los hilos directamente el programador.

Esta clase consta de 3 tipos (parámetros) y 4 métodos que hay que desarrollar para lograr el propósito ofrecido.

A continuación se muestra el código de una activity que define una tarea asíncrona.

public class SitiosLocalesActivity extends SherlockActivity {

private ListView lv;

private TextView miUbicacion;

private int idTipoSitioLocal;

private ArrayList<ItemSitioLocal> itemsSitiosLocal;

private ProgressDialog progressDialog;

private boolean excepcionEstado = false;

private String excepcionDescripcion;

@Override

protected void onCreate(Bundle savedInstanceState) {

try{

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_sitioslocales);

lv = (ListView) findViewById(R.id.listView);

miUbicacion = (TextView) findViewById(R.id.miposicionSitiosLocales);

itemsSitiosLocal = obtenerItems();

AdaptadorListaSitiosLocales adapter = new AdaptadorListaSitiosLocales(this, itemsSitiosLocal);

lv.setAdapter(adapter);

progressDialog = new ProgressDialog(this);

progressDialog.setProgressStyle(ProgressDialog.STYLE\_SPINNER);

progressDialog.setMessage("Obteniendo su ubicación");

progressDialog.setCancelable(true);

progressDialog.setMax(100);

lv.setOnItemClickListener(new AdapterView.OnItemClickListener(){

@Override

public void onItemClick(AdapterView<?> padre, View itemPulsado, int posicion,

long id) {

idTipoSitioLocal = itemsSitiosLocal.get(posicion).getId();

lanzarActividadDetalleSitioLocal();

}

});

new ConsultarServicioWebDeGeolocalizacion(this).execute();

}catch(NumberFormatException ex){

Toast.makeText(this, "Error interno de la aplicación al mostrar los datos en el mapa. Por favor inténtelo nuevamente ", Toast.LENGTH\_LONG).show();

}catch(Exception ex){

Toast.makeText(this, "Error grave de la aplicación al mostrar los datos en el mapa. Por favor inténtelo nuevamente ", Toast.LENGTH\_LONG).show();

}

}

**private class ConsultarServicioWebDeGeolocalizacion extends AsyncTask<Void, Integer, String>{**

private Context context;

private boolean tareaCorrecta = false;

public ConsultarServicioWebDeGeolocalizacion(Context context){

this.context = context;

}

@Override

**protected String doInBackground(Void...params) {**

String resultado = null;

try {

resultado = MiUbicacion.obtenerNombreUbicacion();

tareaCorrecta = true;

}catch(Exception ex){

progressDialog.dismiss();

excepcionEstado = true;

excepcionDescripcion = ex.getMessage();

}

return resultado;

**}**

@Override

**protected void onPostExecute(String string) {**

progressDialog.dismiss();

if(tareaCorrecta){

Toast.makeText(context, "Tarea finalizada!",Toast.LENGTH\_SHORT).show();

llenarDatosConServicioWeb(string);

tareaCorrecta = false;

}else{

if(excepcionEstado){

Toast.makeText(context, "No se pudo completar la tarea correctamente! " + excepcionDescripcion, Toast.LENGTH\_LONG).show();

excepcionEstado = false;

}else{

Toast.makeText(context, "No se pudo completar la tarea correctamente!" , Toast.LENGTH\_LONG).show();

}

}

**}**

@Override

**protected void onProgressUpdate(Integer... values) {**

int progreso = values[0].intValue();

progressDialog.setProgress(progreso);

**}**

@Override

**protected void onPreExecute() {**

progressDialog.setProgress(0);

progressDialog.show();

**}**

**} // Fin clase interna**

public void llenarDatosConServicioWeb(String string){

try{

miUbicacion.setText(string);

}catch(NullPointerException exception){

miUbicacion.setText("No se han obtenido resultados");

Toast.makeText(this, "Error al mostrar los datos en pantalla. Intentelo nuevamente", Toast.LENGTH\_LONG).show();

}catch(Exception ex){

miUbicacion.setText("No se han obtenido resultados");

Toast.makeText(this, "Error inesperado no se pueden cargar los datos", Toast.LENGTH\_LONG).show();

}

}

private ArrayList<ItemSitioLocal> obtenerItems() {

ArrayList<ItemSitioLocal> items = new ArrayList<ItemSitioLocal>();

items.add(new ItemSitioLocal(1, "Restaurant", "Restaurant", "drawable/list\_restaurant"));

items.add(new ItemSitioLocal(2, "Bar", "Bar", "drawable/list\_drink" ));

items.add(new ItemSitioLocal(3, "Casino", "Casino", "drawable/list\_casino" ));

items.add(new ItemSitioLocal(4, "Doctor", "Doctor", "drawable/list\_medico" ));

items.add(new ItemSitioLocal(5, "Hospital", "Hospital", "drawable/list\_hospital" ));

return items;

}

}

Se puede observar en el código anterior que dentro de la clase SitiosLocalesActivity se ha declarado una clase interna llamada ConsultarServicioWebDeGeolocalización que define una tarea asíncrona al extender de AsyncsTask.

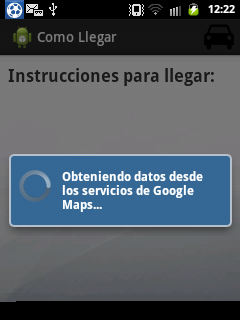
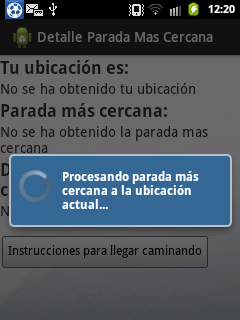
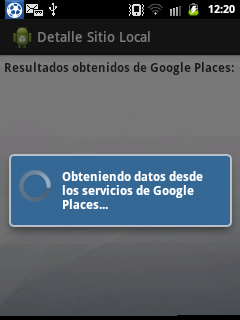
El objetivo de la clase SitiosLocalesActivity es mostrarle al usuario su ubicación y ofrecer un listado de tipos de sitios locales a consultar, cercanos a la misma.

Cuando la actividad inicia tiene que, inmediatamente, obtener la ubicación actual del usuario mediante el dispositivo GPS. Éste retorna la ubicación como un conjunto de coordenadas expresadas en forma de latitud y longitud. Una vez obtenida estas coordenadas, el siguiente paso es transformarlas en una dirección real. Para realizar esto es necesario pasarle estas coordenadas a la clase Geocoder. Como se puede observar todo este proceso se realiza dentro del método doInBackground(Void...params) mediante la sentencia MiUbicación.obtenerNombreUbicación().

Este método, doInBackground(Void...params) que no recibe ningún parámetro, es quien se encarga de realizar esta tarea de transformación de coordenadas en segundo plano mientras la interfaz de usuario se carga. Una vez que finaliza la tarea, el resultado es pasado al método onPostExecute(String string) mediante el parámetro llamado string, que como se puede observar está presente en esta clase y él es el encargado de llamar al método que presenta el resultado en la pantalla y de pasarle el resultado obtenido desde doInBackground que cuenta con una sentencia return. Dentro del método onPostExecute(String string) podemos ver que se invoca al método llenarDatosConServicioWeb(string).

Cuando la aplicación es utilizada por el usuario y él corra esta activity, ésta mostrará un círculo de progreso hasta que finalice la tarea de transformación de coordenadas en segundo plano y la interfaz gráfica de la Activity se cargará sin problemas mientras dure esta tarea. Luego de que finalice la tarea en segundo plano los resultados serán publicados en la interfaz gráfica.

A continuación se muestra un conjunto de activities que definen dentro de ella una tarea asíncrona.



**Figura 17: Distintas activities que llevan acabo una tarea asíncrona.**

* 1. **Utilización de recursos alternativos**

La tarea de separar los recursos en una aplicación permite al desarrollador proporcionarlos según las distintas configuraciones de los dispositivos, a estos recursos se los denomina alternativos. Éstos se vuelven cada vez más importantes a medida que más dispositivos con Android están disponibles con diferentes configuraciones. Con el fin de proporcionar compatibilidad o alternativas a configuraciones diferentes, se deben organizar los recursos en el directorio del proyecto “res /”, utilizando diversos subdirectorios que agrupen recursos por grupo y por tipo de configuración.

Android utiliza una lista de sufijos para expresar recursos alternativos para determinada configuración. Estos sufijos pueden hacer referencia a la orientación del dispositivo, al lenguaje, la región, entre otras.

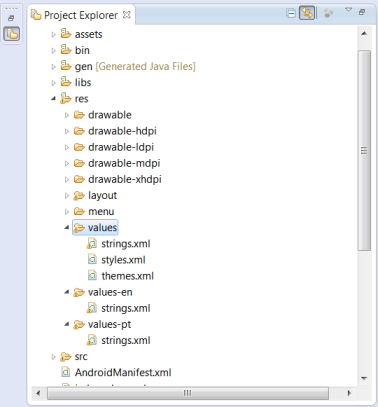
Luego en tiempo de ejecución Android evalúa cuál es la configuración actual del dispositivo, una vez determinada Android analiza los sufijos presentes en los recursos de la aplicación y en función de esto carga el recurso adecuado para la configuración. Permitiendo proporcionar cadenas diferentes (estas cadenas no son más que recursos definidos en archivos XML con distintos prefijos en sus nombres) en función de la configuración del idioma en el dispositivo.

Para traducir la aplicación desarrollada en esta tesina a los idiomas inglés y portugués, presentes en la mayoría de los dispositivos, se crearán y agregarán dos versiones distintas del fichero strings.xml, y se los deberá guardar en los siguientes directorios:

* res/values/strings.xml
* res/values -en/strings.xml
* res/values -pt/strings.xml

Como se puede observar, en el segundo punto de la lista anterior, el directorio agrega un sufijo “-en” el cual determina que dentro de éste se encuentran las cadenas de texto en idioma inglés. En el tercer punto el directorio agrega un sufijo “-pt” el cual determina que dentro de éste se encuentran cadenas de texto en idioma portugués. El directorio que no agrega sufijo es el usado por defecto para buscar las cadenas y cargarlas.

A continuación se muestra la estructura resultante del proyecto.



**Figura 18: Estructura del proyecto. En esta imagen se pude visualizar tres directorios values con su correspondiente sufijo.**

Posteriormente se muestra una sección de estos documentos con el fin de mostrar su implementación. No se muestra el total del contenido de cada uno de los archivos Strings.xml dada su gran extensión.

**res/values/strings.xml:**

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Sistema de asistencia al turista</string>

<string name="menu\_settings">Settings</string>

<string name="title\_activity\_recorridos">Recorridos Disponibles</string>

<string name="title\_activity\_entidades\_relacionadas">Entidades Relacionadas</string>

<string name="title\_activity\_lugares\_interes">Lugares de Interes</string>

<string name="title\_activity\_detalle\_parada">Detalle Parada</string>

<string name="title\_activity\_recorrido\_elegido">Recorrido Elegido</string>

<string name="title\_activity\_recorrido\_destino">Recorrido Destino</string>

<string name="title\_activity\_listado\_recorridos">Listado de Recorridos</string>

<string name="title\_activity\_entidad\_referencia">EntidadReferencia</string>

<string name="title\_activity\_detalle\_entidad\_relacionada">Detalle Entidad Relacionada</string>

<string name="title\_activity\_parada\_cercana\_xrecorrido">Parada Cercana Según Recorrido</string>

**<string name="lbl\_ubicacionActual">Tu ubicación es:</string>**

/……………………………………./

</resources>

**res/values -en/strings.xml:**

(Contiene las etiquetas en inglés)

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Tourist assistance system´s</string>

<string name="menu\_settings">Settings</string>

<string name="title\_activity\_recorridos">Tours available</string>

<string name="title\_activity\_detalle\_parada">Bus stop detail</string>

<string name="title\_activity\_recorrido\_elegido">Chosen Tour</string>

<string name="title\_activity\_recorrido\_destino">Tour destination</string>

<string name="title\_activity\_listado\_recorridos">List of tours</string>

<string name="title\_activity\_detalle\_entidad\_relacionada">Detail related entity </string> <string name="title\_activity\_recorrido\_elegido\_lista">Detail Chosen Tour</string>

<string name="title\_activity\_parada\_cercana\_xrecorrido">Near bus stop for tour</string>

**<string name="lbl\_ubicacionActual">Your location is:</string>**

/……………………………………./

</resources>

**res/values -pt/strings.xml:**

(Contiene las etiquetas en portugués)

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Assistência turística do Sistema</string>

<string name="menu\_settings">Settings</string>

<string name="title\_activity\_recorridos">Tours disponíveis</string>

<string name="title\_activity\_detalle\_parada">Detalhes de ônibus</string>

<string name="title\_activity\_recorrido\_elegido">turismo escolhido</string>

<string name="title\_activity\_recorrido\_destino">Destino turístico</string>

<string name="title\_activity\_listado\_recorridos">Lista de passeios</string>

<string name="title\_activity\_detalle\_entidad\_relacionada">Detalhe entidade relacionada</string>

<string name="title\_activity\_recorrido\_elegido\_lista">Turismo Detalhe Escolhido</string>

<string name="title\_activity\_parada\_cercana\_xrecorrido">Parada de turismo de acordo próximo</string>

**<string name="lbl\_ubicacionActual">Sua localização é:</string>**

/……………………………………./

</resources>

Se mencionó en la sección de construcción de una interfaz gráfica que en el layout de cada interfaz se hace el llamado a cada una de las etiquetas o cadenas de texto con sólo el nombre definido para la misma (por ejemplo **lbl\_ubicacionActual**), según sea necesario cuál mostrar. De esta manera en tiempo de ejecución, la aplicación según el tipo de idioma configurado, irá invocando las distintas etiquetas. Ya que el nombre de éstas es el mismo en los tres archivos, Android sabrá en cuál layout buscar la correspondiente según el idioma.

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:background="@drawable/fondo"

tools:context=".DetalleParadaMasCercanaBusquedaXDestino" >

<LinearLayout

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

>

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="22sp"

android:textStyle="bold"

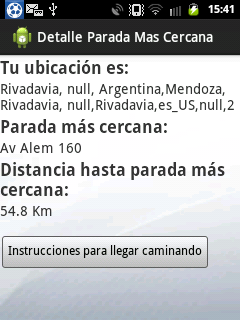
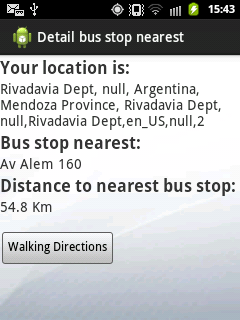
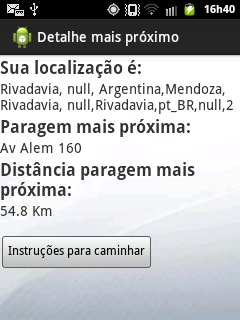
**android:text="@string/lbl\_ubicacionActual" // llamado a la etiqueta “lbl\_ubicacionActual”**

/>

/…………………………………./

</LinearLayout>

Resultado:

**Figura 19: Mostrando detalles de la parada más cercana en distintos idiomas. La primera imagen muestra la aplicación en el idioma español, en la segunda en inglés y la tercera en portugués.**

* 1. **Utilización de fragmentos con la librería Sherlock**

Desde la aparición de Android 3.0, se introdujo un nuevo aspecto para la interfaz de usuario, el uso de fragmentos (pestañas) en una barra superior que incluyen las aplicaciones diseñadas a partir de la versión 3.0. Sin embargo, no es necesario excluir a todos los dispositivos que no tengan esa versión de Android. Los que tengan 2.1 en adelante pueden aprovechar esta característica, si se programa usando la librería externa [Action Bar Sherlock](http://actionbarsherlock.com/), que permite la navegación a través de tabs o pestañas (fragmentos) y observar una barra sobre cada actividad que indica en qué parte de la aplicación se encuentra el usuario.

Para poder usar fragmentos con esta librería se necesita definir en el proyecto los siguientes elementos:

* Una clase Java, que herede de SherlockFragmentActivity. Ésta gestionará los fragmentos.
* Una clase Java que herede de SherlockFragment. Ésta representará un fragmento. En este caso se crean dos fragmentos, en uno se visualizan todas las noticias, actualidades de la ciudad de Mendoza, y en el otro se visualizan todos los tweets con las novedades sobre la aplicación.
* Un layout para la clase que gestiona los fragmentos, que estará vacío, ya que el contenido de éstos entrará de forma dinámica, y no se necesitará mostrar nada más en la pantalla.
* Dos layouts para los fragmentos.

A continuación se muestra el código de la clase que representa un fragmento encargado de mostrar los tweets relacionados con las novedades de la aplicación.

**public** **class** FragmentoNoticiasAplicacion **extends** SherlockFragment {

**private** List<String> datos;

**private** List<TweetItem> itemDatos;

**private** ProgressDialog progressDialog;

**private** ListView vistaLista;

**private** ConsultarServicioWebTwitter servicioWebTwitter = **new** ConsultarServicioWebTwitter();

**public** FragmentoNoticiasAplicacion() {

}

**public** **void** onCreate(Bundle savedInstanceState){

**try**{

**super**.onCreate(savedInstanceState);

progressDialog = **new** ProgressDialog(**this**.getActivity());

progressDialog.setProgressStyle(ProgressDialog.*STYLE\_SPINNER*);

progressDialog.setMessage("Obteniendo últimos tweets desde twiter ...");

progressDialog.setCancelable(**true**);

progressDialog.setMax(100);

**new** ConsultarServicioTwitter(**this**.getActivity()).execute();

}**catch**(NullPointerException ex){

}

}

Se puede observar que FragmentoNoticiasAplicacion extiende de SherlockFragment, convirtiéndose en un fragmento. Dentro de esta clase se realiza una conexión al servicio web de Twitter mediante una tarea asíncrona llamada ConsultarServicioTwitter.

Posteriormente se muestra el código de la clase encargada de la gestión de los distintos fragmentos.

**public** **class** NovedadesEnLaWeb **extends** SherlockFragmentActivity **implements** TabListener {

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_twitter\_main*);

ActionBar bar = getSupportActionBar();

ActionBar.Tab tab1 = bar.newTab();

ActionBar.Tab tab2 = bar.newTab();

tab1.setText("Gobierno");

tab1.setTag("Gobierno");

tab2.setText("Novedades Aplicación");

tab2.setTag("NovedadesAplicacion");

tab1.setTabListener(**this**);

tab2.setTabListener(**this**);

bar.addTab(tab1);

bar.addTab(tab2);

bar.setNavigationMode(ActionBar.*NAVIGATION\_MODE\_TABS*);

}

@Override

**public** **void** onTabSelected(Tab tab, FragmentTransaction ft) {

String tag = (String) tab.getTag();

Log.*d*("TabTutorial", "Elegido tab con tag: " + tag);

**if** (tag.equalsIgnoreCase("Gobierno")){

FragmentoGobierno fragGobierno = **new** FragmentoGobierno();

ft.replace(android.R.id.*content*, fragGobierno);

}**else** **if** (tag.equalsIgnoreCase("NovedadesAplicacion")){

FragmentoNoticiasAplicacion fragNoticiasAplicacion = **new** FragmentoNoticiasAplicacion();

ft.replace(android.R.id.*content*, fragNoticiasAplicacion);

}**else**{

Log.*w*("TabTutorial", "tag de tab no registrada: " + tag);

}

}

Se puede observar dentro de la clase NovedadesEnLaWeb, en el método onCreate(), que se crean dos pestañas (Tab) en las cuales podremos pulsar para observar el contenido de cada fragmento. Éstas son creadas mediante el método newTab(). Luego se debe asignar una etiqueta mediante setText() y además una identificación mediante setTag().

Una vez realizado ésto, el paso siguiente es asignar a cada pestaña un oyente, el cual estará a la espera de que se presione sobre alguna de éstas, cuando esto suceda, él será el encargado de realizar una serie de acciones. Esta asignación de un oyente se hace mediante el método setTabListener().

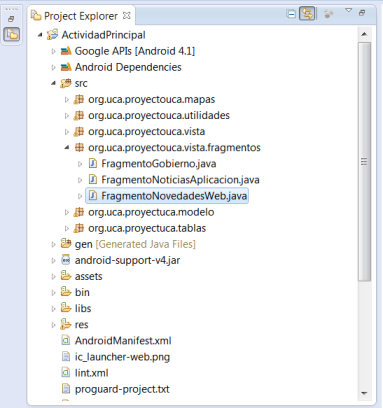
Como paso final, se asignan las pestañas a la barra y se activa el modo de navegación por éstas.

Cabe destacar que el oyente que se setea a las pestañas es la misma clase NovedadesEnLaWeb.

Al utilizar esta interfaz se debe implementar el método onTabSelected(), el cual es llamado por el oyente cada vez que se pulsa sobre una pestaña. Dentro de éste creamos un fragmento nuevo con new Fragment() del tipo adecuado según la elección de la pestaña, y luego fijamos el fragmento que será el que se visualizará en la pantalla mediante el método replace(android.R.id.*content*, fragGobierno).

En este caso, tanto las clases para gestionar como para representar los fragmentos se encuentran en el mismo paquete.

Seguidamente se muestra la estructura del proyecto resultante.



**Figura 20: Estructura del proyecto. Clases necesarias para la utilización de Fragmentos.**

Por último se muestra la clase Novedades en la Web, que es la encargada de gestionar los dos fragmentos creados anteriormente.

C:\Users\Javier\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\Nueva imagen (4).bmp

**Figura 21: Utilización de fragmentos.**

* 1. **Consumición de los servicios web del API de Google Maps y de Twitter**

**Servicios web del API de Google Maps**

Esta API proporciona un conjunto de servicios web para solicitar datos a servicios externos y poder utilizarlos en las aplicaciones. Éstos se consumen mediante solicitudes HTTP a URLs específicas, transmitiendoles parámetros de URL como argumentos. Éstos devuelven los datos solicitados, en formato JSON o XML para que la aplicación los analice o procese.

A continuación se muestra la clase que tiene la responsabilidad de consumir uno de estos servicios ofrecidos por esta API. En este caso, es el servicio de rutas.

public class ConsultarServicioWebGoogle {

**public** StringBuilder consultarServicioRutaCaminando(String latitudDestino, String longitudDestino) **throws** InformacionException{

StringBuilder resultados = **null**;

**try**{

URL url = **new** RL("https://maps.googleapis.com/maps/api/directions/xml?origin=" + MiUbicacion.getLatitud() + "," + MiUbicacion.getLongitud() + "&destination="+ latitudDestino + "," + longitudDestino + "&mode=walking&region=es&language=es&sensor=false");

HttpURLConnection conexcion = (HttpURLConnection) url.openConnection();

conexcion.setRequestMethod("GET");

conexcion.setDoOutput(**true**);

OutputStreamWriter salida = **new** OutputStreamWriter(conexcion.getOutputStream());

**if**(conexcion.getResponseCode() == HttpURLConnection.HTTP\_OK){

SAXParserFactory fabrica = SAXParserFactory.newInstance();

SAXParser parser = fabrica.newSAXParser();

XMLReader lector = parser.getXMLReader();

ManejadorSerWebRutaGoogleCaminando manejadorXML = **new** ManejadorSerWebRutaGoogleCaminando();

lector.setContentHandler(manejadorXML);

lector.parse(**new** InputSource(conexcion.getInputStream()));

resultados = manejadorXML.getResultados();

}**else**{

**throw** **new** InformacionException("No se han obtenido datos del servicio de rutas de Google maps" + conexcion.getResponseMessage());

}

}**catch**(MalformedURLException ex){

**throw** **new** InformacionException("La URL a la que intenta conectarse la aplicación no es correcta en el servicio de rutas");

}

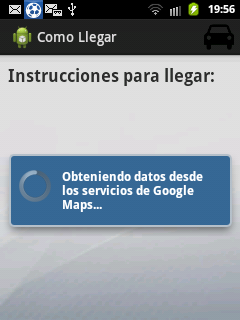
**return** resultados;

}

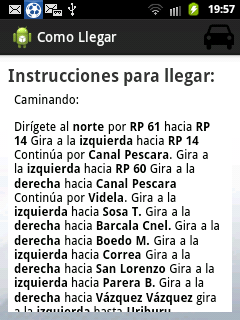
Se puede observar que la clase ConsultarServicioWebGoogle cuenta con un método llamado consultarServicioRutaCaminando, en el cual se crea un objeto URL. A éste, se le pasa como parámetro la dirección HTTP de dónde consumir el servicio, y ésta posee como argumentos, el origen, el destino, el modo (caminando o en vehículo) y el formato de los datos devueltos.

Luego, mediante el objeto HttpURLConnection se realiza la conexión. Posteriormente se le pasa al manejador (clase que hereda de DefaultHandler) el resultado obtenido de la conexión. Éste es el encargado de realizar el parseo de la información obtenida en formato XML.

Por último se muestra el momento en que se consume el servicio y el resultado del mismo.

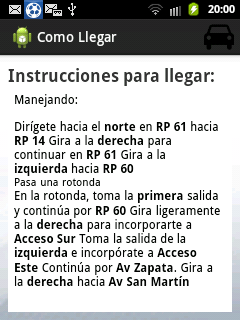


**Figura 22: Obteniendo datos del servicio web del API de Google Maps.**



**Figura 23: Datos obtenidos del servicio web del API de Google Maps. Caminando.**

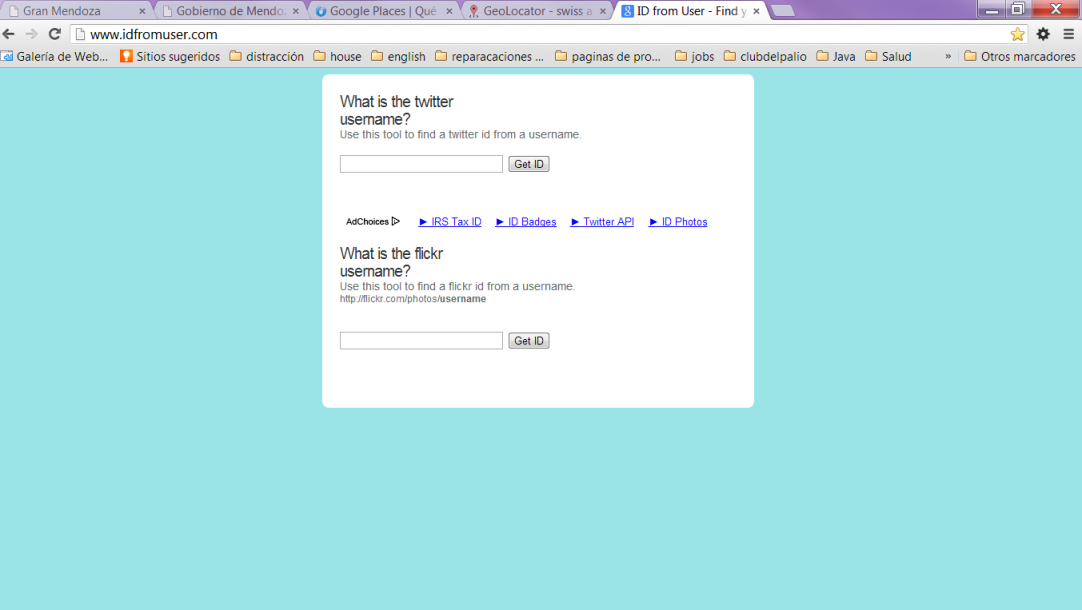
En la figura anterior se puede observar, en la barra superior de la aplicación, un icono de un auto. Al pulsar sobre éste, se obtendrá nuevamente información sobre la ruta a seguir para llegar al hotel o bodega, pero en este caso la ruta calculada tendrá en cuenta que se dirige en auto.



**Figura 24: Datos obtenidos del servicio web del API de Google Maps. Manejando**

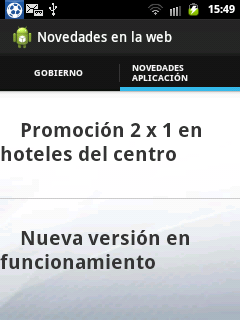
**Servicios web de Twitter**

Twitter proporciona un servicio web que al ser consumido permite obtener los distintos tweets asociados a una cuenta. Para poder consumir éste, es necesario primero conseguir el ID (identificador de usuario) asociado a la cuenta a la cual se desea ingresar. Para esto es necesario consultar el siguiente sitio: <http://www.idfromuser.com/>



**Figura 25: Sitio ID FROM USER. Este sitio devolverá un ID como el siguiente: 1355287615.**

Una vez obtenido el ID asociado a la cuenta es necesario enviar una solicitud GET con el siguiente formato: <http://api.twitter.com/1/statuses/user_timeline.xml?user_id=1355287615>.



**Figura 26: Obteniendo datos del servicio web de Twitter.**

Posteriormente se muestra la clase que lleva acabo la solicitud de esta información:

public class ConsultarServicioWebTwitter {

public ConsultarServicioWebTwitter() {

}

public List<String> consultarServicioTwitter() throws InformacionException{

List<String> resultados = null;

try{

URL url = new URL("http://api.twitter.com/1/statuses/user\_timeline.xml?user\_id=1355287615");

HttpURLConnection conexcion = (HttpURLConnection) url.openConnection();

conexcion.setRequestMethod("GET");

conexcion.setDoOutput(true);

if(conexcion.getResponseCode() == HttpURLConnection.HTTP\_OK){

SAXParserFactory fabrica = SAXParserFactory.newInstance();

SAXParser parser = fabrica.newSAXParser();

XMLReader lector = parser.getXMLReader();

ManejadorSerWebTwitter manejadorXML = new ManejadorSerWebTwitter();

lector.setContentHandler(manejadorXML);

lector.parse(new InputSource(conexcion.getInputStream()));

resultados = manejadorXML.getResultados();

}else{

throw new InformacionException("No se han obtenido datos del servicio de Twitter" + conexcion.getResponseMessage());

}

}catch(Exception ex){

throw new InformacionException("No se han obtenido datos del servicio de Twitter" + conexcion.getResponseMessage());}

return resultados;

}

}

Por último, se puede observar que la clase ConsultarServicioWebTwitter cuenta con el método consultarServicioTwitter() el cual tiene la responsabilidad de consumir el servicio web ofrecido por twitter. Para esto se crea un objeto URL al cual se le pasa la dirección HTTP dónde consumir el servicio y se le pasa como parámetro el ID de la cuenta de twitter a consultar.

Luego se realiza la conexión mediante el objeto HttpURLConnection. A continuación se crea un objeto SAXParserFactory y a partir de éste, se obtiene un objeto parse. A éste objeto parse se le pasa el resultado de la conexión realizada con el servidor de Twitter mediante la siguiente sentencia lector.parse(new InputSource(conexcion.getInputStream())) para que se realice el parseo de la información obtenida.

* 1. **Realización del parseo de la información obtenida de los servicios de la API Google Maps**

La información devuelta por los servicios web de esta API se encuentra codificada en código JSON o XML, según lo especificado en la solicitud GET que se envíe. En la aplicación desarrollada para esta tesina, todas las solicitudes especifican que la información devuelta sea en el formato XML. De modo que es necesario parsear ésta de manera que pueda ser mostrada en forma legible en las pantallas de la aplicación.

Todas las clases que realizan esta tarea adoptan el modelo SAX. Esta elección se debe a que es la alternativa que menos cantidad de recursos demanda para realizar su tarea.

**Realizando el parseo con SAX**

El tratamiento de XML con SAX se basa en un analizador (parse) que a medida que lo lee secuencialmente, va generando diferentes eventos con la información de cada elemento examinado.

Para analizar un documento XML con SAX se debe escribir una serie de métodos asociados a cada tipo de evento que se pueda generar. Para poder definir el comportamiento de estos métodos es necesario extender de la clase DefaultHandler, de esta manera la clase se convertirá en un manejador de eventos, que capture los eventos generados al leer el documento XML.

A continuación se muestra el código de la clase que define el manejador de eventos y el comportamiento de cada uno de los métodos encargados de manejar los eventos generados durante la lectura.

**private** **class** ManejadorSerWebRutaGoogleManejando **extends** DefaultHandler{

**private** StringBuilder cadena;

**private** StringBuilder resultados;

**public** StringBuilder getResultados(){

resultados.append("</body></html>");

**return** resultados;

}

@Override

**public** **void** startDocument() **throws** SAXException{

resultados = **new** StringBuilder();

resultados.append("<html><head><meta name=\"tipo\_contenido\" content=\"text/html;\" http-equiv=\"content-type\" charset=\"utf-8\"></head><body><p>Manejando:</p>");

cadena = **new** StringBuilder();

}

**public** **void** characters(**char** ch[], **int** comienzo, **int** longitud){

cadena.append(ch,comienzo,longitud);

}

@Override

**public** **void** endElement(String uri, String nombreLocal, String nombreCualif) **throws** SAXException{

**if**(nombreCualif.equals("html\_instructions")){

**try**{

resultados.append(((URLDecoder.decode(cadena.toString(), "UTF8"))));

}**catch**(UnsupportedEncodingException e){

Log.e("Error interno de la aplicación", e.getMessage(),e);

}

}

cadena.setLength(0);

}

}

Se puede observar que la clase, ManejadorSerWebRutaGoogleManejando extiende de DefaultHandler, de manera de poder definir un manejador que capture los cinco eventos generados en el proceso de parsing en SAX. En el método startDocument() se inicializa la variable resultados, la cual contendrá la información resultante de aplicar el parsing. Se comienza agregándole a la variable de nombre resultado, contenido en html para luego mostrar el contenido total de la variable en un control webView. En este método también utilizamos una variable llamada cadena la cual se utiliza para almacenar cadenas temporales del proceso de parsing.

En ésta, también se ha definido el método characters(), el cual es invocado cada vez que se encuentra texto dentro de una etiqueta. Otro método definido es endElement() que es llamado cuando finaliza una etiqueta, el contenido de ésta es agregado a la variable cadena, luego se decodifica y se agrega a la variable resultado.

Como paso final, es necesario la creación de una clase que utilice el Handler:

Public class AnalizadorXML(){

Public void leerCodigoXML(){

SAXParserFactory fabrica = new SAXParserFactory.newInstance();

SAXParser parser = fabrica.newSAXParser();

XMLReader lector = parse.getXMLReader();

ManejadorXML manejador = new ManejadorXML();

Lector.setContentHandler(manejador);

Lector.parse(new InputSource(entrada));

}

}

Se comienza creando instancias de:

* SAXParserFactory: lo que permite obtener un nuevo parser.
* XMLReader: lo que permite obtener un lector asociado a este parser.
* ManejadorSerWebRutaGoogleManejando (clase que hereda de DefaultHandler): lo que permite obtener un manejador, que capture los eventos generados por el parser, éste se asocia al lector XMLReader.

Para finalizar se le indica al lector XMLReader qué entrada tiene para que realice el proceso del parser.

A continuación se muestra la implementación de los dos modelos restantes para poder leer y escribir un documento XML. Estos dos modelos no han sido utilizados en el desarrollo de la aplicación para esta tesina, pero a modo de agregado, se muestra como podría llegar a ser su implementación dado que en todo momento se ha hecho mención de éstos al hablar sobre XML.

**Realizando parseo con DOM**

En el tratamiento de XML con DOM, se lee completamente antes de poder realizar alguna acción en función de su contenido. Esto es posible gracias a que, como resultado de la lectura del documento, el parser DOM devuelve todo su contenido en forma de una estructura de tipo árbol, donde los distintos elementos del XML se representa en forma de nodos y su jerarquía padre-hijo se establece mediante relaciones entre dichos nodos.

Para poder procesar la estructura del árbol generado, DOM ofrece una serie de clases y métodos que permiten la navegación y el tratamiento de ésta.

A continuación se muestra una clase que hace uso de un parser utilizando el modelo DOM y los métodos que ofrece:

public class AlumnosParserDom{

private URL rssUrl;

public AlumnosParserDom (String url){

try{

this.rssUrl = new URL(url);

} catch (MalformedURLException e){

throw new RuntimeException(e);

}

}

public List<Noticia> parse(){

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

List<Alumno> alumnos = new ArrayList<Alumno>();

try{

DocumentBuilder builder = factory.newDocumentBuilder();

Document dom = builder.parse(this.getInputStream());

Element root = dom.getDocumentElement();

NodeList items = root.getElementsByTagName("alumno");

for (int i=0; i<items.getLength(); i++){

Alumno alumno = new Alumno();

Node item = items.item(i);

NodeList datosAlumno = item.getChildNodes();

for (int j=0; j<datosAlumno.getLength(); j++){

Node dato = datosAlumno.item(j);

String etiqueta = dato.getNodeName();

if (etiqueta.equals("nombre")){

alumno.setNombre(dato.getFirstChild().getNodeValue());

}else if (etiqueta.equals("carrera")){

alumno.setCarrera(dato.getFirstChild().getNodeValue());

}

}

alumnos.add(alumno);

}

}catch (Exception ex){

throw new RuntimeException(ex);

}

return alumnos;

}

private InputStream getInputStream(){

try{

return rssUrl.openConnection().getInputStream();

}catch (IOException e){

throw new RuntimeException(e);

}

}

}

Al igual que se hizo en SAX, el primer paso es instanciar una nueva fábrica, esta vez de tipo DocumentBuilderFactory, y posteriormente crear un nuevo parser (analizador) a partir de ella mediante el método newDocumentBuilder(). Tras esto, ya se puede realizar la lectura del documento XML llamando al método parse() del parser DOM, pasándole como parámetro el stream de entrada del fichero. Al hacer esto, el documento XML con los alumnos de la facultad se lee completo y se genera la estructura de árbol equivalente, que se devuelve como un objeto de tipo Document. Como se ve este es el objeto que se utiliza para poder navegar, para realizar el tratamiento necesario del XML.

Para poder navegar por el documento se accede al nodo principal del árbol (la etiqueta <lista\_alumnosUCA >) utilizando el método getDocumentElement(). Una vez posicionados en dicho nodo, se buscará todos los nodos cuya etiqueta sea <alumno>. Utilizando el método getElementsByTagName(“nombre\_de\_etiqueta“), que devolverá una lista (de tipo NodeList) con todos los nodos hijos del nodo actual cuya etiqueta coincida con la pasada como parámetro.

Una vez que se han localizados todos los elementos <alumno>, se los recorrerá uno a uno para ir generando todos los objetos alumnos necesarios.

Para cada uno de ellos, se obtienen los nodos hijos del elemento, mediante getChildNodes(), se recorren y se obtienen sus textos y se los almacena en los atributos correspondientes de los objetos alumnos. Para saber a qué etiqueta corresponde cada nodo hijo se utilizará el método getNodeName().

**Parseo de la información con XmlPull**

Con XmlPull al igual que SAX se debe definir las acciones a realizar para cada uno de los eventos generados durante la lectura secuencial del documento XML. En este modelo se podrá guiar o intervenir en la lectura del documento, siendo el programador quien vaya pidiendo de forma explícita la lectura del siguiente elemento del XML y respondiendo al resultado, ejecutando las acciones oportunas.

A continuación se muestra una clase que hace uso del parser utilizando el modelo XmlPull y los métodos que ofrece:

public class RssParserPull{

private URL rssUrl;

public RssParserPull(String url){

try{

this.rssUrl = new URL(url);

}catch (MalformedURLException e){

throw new RuntimeException(e);

}

}

public List<Noticia> parse(){

List<Noticia> noticias = null;

XmlPullParser parser = Xml.newPullParser();

try

{

parser.setInput(this.getInputStream(), null);

int evento = parser.getEventType();

Noticia noticiaActual = null;

while (evento != XmlPullParser.END\_DOCUMENT){

String etiqueta = null;

switch (evento){

case XmlPullParser.START\_DOCUMENT:

noticias = new ArrayList<Noticia>();

break;

case XmlPullParser.START\_TAG:

etiqueta = parser.getName();

if (etiqueta.equals("item")){

noticiaActual = new Noticia();

}else if (noticiaActual != null){

if (etiqueta.equals("link")){

noticiaActual.setLink(parser.nextText());

}else if (etiqueta.equals("description")){

noticiaActual.setDescripcion(

parser.nextText());

}else if (etiqueta.equals("pubDate")){

noticiaActual.setFecha(parser.nextText());

}else if (etiqueta.equals("title")){

noticiaActual.setTitulo(parser.nextText());

}else if (etiqueta.equals("guid")){

noticiaActual.setGuid(parser.nextText());

}

}

break;

case XmlPullParser.END\_TAG:

etiqueta = parser.getName();

if (etiqueta.equals("item") && noticiaActual != null){

noticias.add(noticiaActual);

}

break;

}

evento = parser.next();

}

}catch (Exception ex){

throw new RuntimeException(ex);

}

return noticias;

}

private InputStream getInputStream(){

try{

return rssUrl.openConnection().getInputStream();

}catch (IOException e){

throw new RuntimeException(e);

}

}

}

Al centrarse en el método parse(), se puede observar que tras crear el nuevo parser XmlPull y establecer el fichero de entrada en forma de stream (mediante XmlPull.newPullParser() y parser.setInput(...)) se entrará en un bucle en el que se irá solicitando al parser en cada paso el siguiente evento encontrado en la lectura del XML, utilizando para ello el método parser.next().

Para cada evento devuelto como resultado, se consultará su tipo mediante el método parser.getEventType(), y se responderá con las acciones oportunas según dicho tipo, entre ellos: (START\_DOCUMENT, END\_DOCUMENT, START\_TAG, END\_TAG). Una vez identificado el tipo concreto de evento, se podrá consultar el nombre de la etiqueta del elemento XML mediante parser.getName() y su texto correspondiente mediante parser.nextText(). En cuanto a la obtención del texto, con este modelo se adquiere la ventaja de no tener que preocuparse por “recolectar” todos los fragmentos de texto contenidos en el elemento XML, ya que nextText() devolverá todo el texto que encuentre hasta el próximo evento de fin de etiqueta (ENT\_TAG).

* 1. **Implementación y consumición de un servicio web propio**

Para su implementación, se puede usar un servidor web de uso comercial, como Apache Tomcat, e instalar soportes para servicios web, como Axis2.

El software necesario es el siguiente:

* Apache Tomcat.
* Axis2.
* Eclipse con WTP.

**Pasos para su implementación**

1) Crear la funcionalidad del servicio (Java, C, Perl...).

2) Generar el archivo WSDL.

3) Alojar el servicio en un contenedor (Tomcat, entre otros).

4) Registrarlo mediante UDDI.

A partir de la siguiente clase Java, se le pedirá a Eclipse generar el servicio web. Cada uno de sus métodos corresponderá con una operación del servicio que podrá ser consumido.

**public** **class** ProyectoUcaWeb {

**private** EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("ProyectUcaWeb");

**private** EntityManager em = emf.createEntityManager();

**public** **int** consultarEstado(**int** clave){

Recorrido recorrido = **null**;

recorrido = (Recorrido) em.createQuery("SELECT r FROM Recorrido r WHERE r.clave =:clave").setParameter("clave", clave).getSingleResult();

**return** recorrido.getEstado();

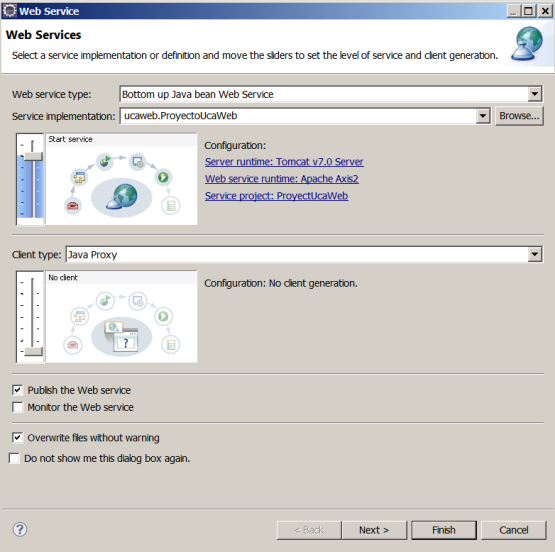
}

}

Se puede observar que esta clase cuenta con un método llamado consultarEstado, Eclipse lo convertirá en una operación del servicio.

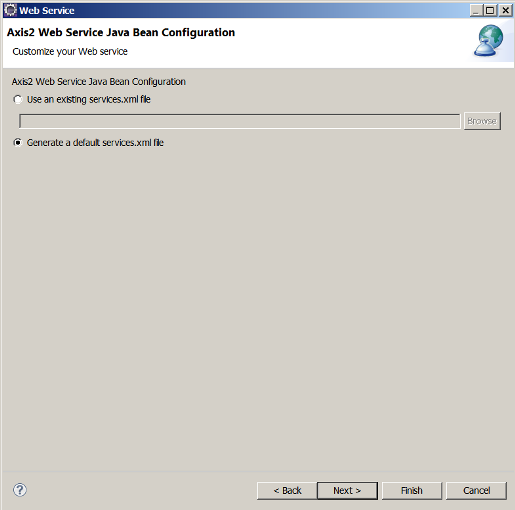
Para convertir esta clase en un servicio web, se deberá, en el explorador de archivos de Eclipse, hacer click con el botón derecho sobre la clase, se desplegará el menú contextual y se debe seleccionar “Web Service” y luego “Create Web Service”.

A continuación aparecerá la siguiente ventana:



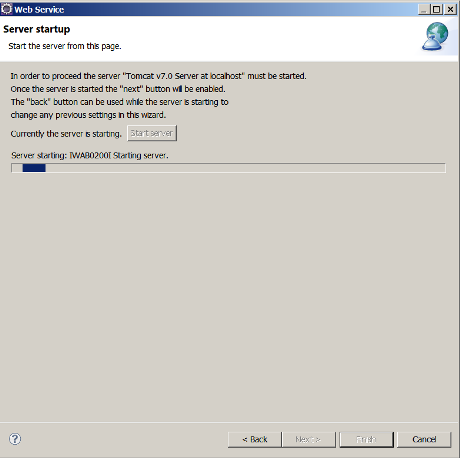
**Figura 27: Creando un servicio web. Paso 1.**

En ella se debe marcar la opción “Publish the web service“. Luego se debe pulsar el botón next y en la siguiente ventana se debe marcar la opción “Generate a default services.xml”.



**Figura 28: Creando un servicio web. Paso 2.**

Inmediatamente se pulsará el botón next y Eclipse iniciará el servidor.



**Figura 29: Creando un servicio web. Paso 3.**

Automáticamente eclipse realizará las siguientes tareas:

* Generará el archivo WSDL.
* Colocará el servicio en el contenedor Tomcat.
* Registrará el servicio web mediante UDDI.

Una vez que se han llevado estas tareas acabo con éxito, Eclipse ofrece la posibilidad de explorar el servicio web. Para ello en el explorador de archivos se deberá hacer click con el botón derecho sobre el proyecto, a continuación se desplegará el menú contextual y se seleccionará la opción “Run As” y luego “Run on Server”.

Si el servidor se encuentra iniciado aparecerá un navegador web, en el que se visualizará la siguiente dirección <http://localhost:8080://nombredelproyecto/>. Desde esta página se puede gestionar el administrador de servicios web Axis2. Seguidamente, se hará click en el link Services, y el servidor listará los servicios disponibles:



**Figura 30: Servicios web disponibles en el servidor web Tomcat con soporte para servicios web Axis 2.**

En este caso se observan dos servicios disponibles. Uno que se instala por defecto y el que se acaba de crear “ProyectoUcaWeb”.

Para cada uno de ellos se listan las operaciones disponibles.

Como se puede observar en el servicio ProyectoUcaWeb se encuentra el método consultarEstado.

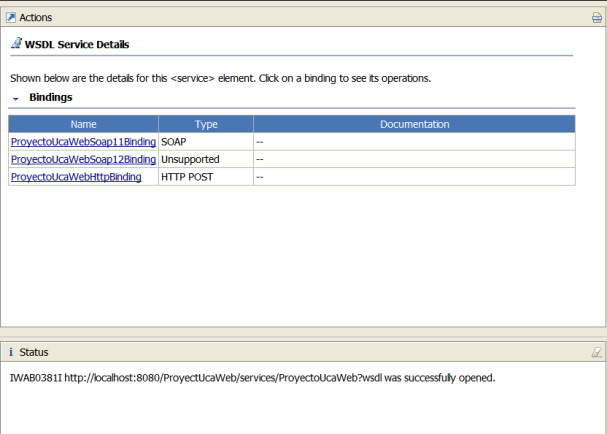
Al pulsar sobre el link que contiene el nombre del proyecto se visualizará el fichero WSDL asociado al servicio (cuya funcionalidad fue explicado anteriormente), el cual describe la interfaz pública de un servicio web.

Se mostrará el contenido del archivo WSDL que generó eclipse en la siguiente imagen:



**Figura 31: Contenido XML del archivo WDSL creado por Eclipse.**

Posteriormente se abrirá el Launch the Web Services Explorer, se pulsará sobre WDSL Page y se podrá observar los detalles del archivo WSDL en un formato mucho más entendible.



**Figura 32: Distintas implementaciones de un servicio web. Implementado en SOAP 1.1, SOAP 1.2 y REST (HTTP POST).**

Se puede observar en la figura anterior, en la tabla “Bindings”, que el servicio web se ha implementado mediante las dos alternativas mencionadas anteriormente SOAP versión 1.1 y 1.2, y REST (HTTP POST)

A continuación se muestra el contenido de los mensajes SOAP que implementan el servicio web:



**Figura 33: Contenido de un mensaje de solicitud SOAP y su correspondiente mensaje de repuesta SOAP.**

Seguidamente se define una página web con html básico para realizar una consulta POST para consumir el servicio web implementado en REST. El código de ésta es el siguiente:

<html>

<body>

<h2>Introducir ID del recorrido</h2>

<form method="POST" action="http://localhost:8080/ProyectUcaWeb/services/ProyectoUcaWeb.ProyectoUcaWebHttpEndpoint/consultarEstado">

ID: <input name="clave" type="text" value="1">

<input type="submit" value="Invocar al servicio web">

</form>

</body>

</html>

Por último, se detalla la respuesta dada por el servicio web:



**Figura 34: Respuesta obtenida del servicio web implementado en REST.**

* 1. **Aplicación de Capas**

Para poder mostrar datos sobre un mapa, Android ofrece tres clases Overlay, OverlayItem e ItemizedOverlay. Seguidamente se hace hincapié una vez más sobre la diferencia entre ellas y se muestra cómo ha sido su implementación.

**Dibujando una ruta**

Overlay es una capa de dibujo en general, en la que se pueden mostrar datos como: líneas, círculos, cuadros con texto, entre otros.

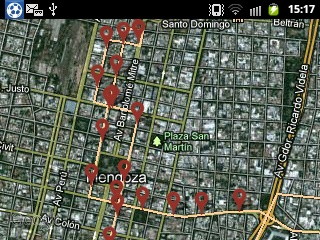
ItemizedOverlay es una subclase de Overlay (lo que hace posible que pueda ser dibujado su contenido sobre un mapa), que hace que sea más fácil crear una capa con una serie de puntos o elementos marcados sobre un mapa, dado que ofrece un conjunto de métodos muy útiles para agregar y administrar los elementos de la capa (por ejemplo algún marcador).

Cada punto o elemento de la capa es representado por medio de una clase denominada OverlayItem. Por esto, podemos decir que ItemizedOverlay es una lista que contiene un conjunto de objetos del tipo OverlayItem que serán dibujados sobre un mapa y que podrán ser fácilmente administrados.

Por lo tanto, si se está tratando de mostrar una ruta de autobús, o la sombra de una región, Overlay es la clase recomendada. En cambio, si se está tratando de mostrar una colección de restaurantes, o bodegas, hoteles o cualquier otro conjunto de puntos, ItemizedOverlay es recomendable.

En la aplicación desarrollada para esta tesina se ha utilizado la clase Overlay para dibujar una capa que representa el recorrido de los micros calle a calle a través de líneas rectas en el mapa. Entonces esta primera capa consta de un conjunto de líneas ininterrumpidas para mostrar el recorrido.

La clase ItemizedOverlay se ha utilizado para dibujar una segunda capa que representa a través de un conjunto de marcadores todas las paradas que posee el recorrido. La siguiente imagen muestra ambos overlays (capas) sobre el mismo mapa.



**Figura 35: Dos capas sobre un mismo objeto mapview.**

Posteriormente se puede observar el código de la capa que muestra en pantalla el recorrido a través de un conjunto de líneas.

**Código para dibujar la capa que representa la ruta del recorrido:**

**public** **class** LinePathOverlay **extends** Overlay {

**private** GeoPoint GeoPointInicio;

**private** GeoPoint GeoPointFin;

**private** Projection projection;

**private** String HexColor = "#FFCC99";

**public** LinePathOverlay(GeoPoint geoPointInicio,GeoPoint geoPointFin, String color){

**if** (color.length() > 0)

HexColor = "#" + color;

**this**.GeoPointInicio = geoPointInicio;

**this**.GeoPointFin = geoPointFin;

}

**public** LinePathOverlay(GeoPoint geoPointInicio, GeoPoint geoPointFinal ){

**this**.GeoPointInicio = geoPointInicio;

**this**.GeoPointFin = geoPointFinal;

(Float.parseFloat(parada2.getGeoPointLatitud()) \* 1E6) , (int) (Float.parseFloat(parada2.getGeoPointLongitud()) \* 1E6));

}

**public** **void** draw(Canvas canvas, MapView mapview, **boolean** shadow){

**super**.draw(canvas, mapview, shadow);

Paint mPaint = **new** Paint();

mPaint.setDither(**true**);

mPaint.setColor(Color.parseColor(HexColor));

mPaint.setStyle(Paint.Style.FILL\_AND\_STROKE);

mPaint.setStrokeJoin(Paint.Join.ROUND);

mPaint.setStrokeCap(Paint.Cap.ROUND);

mPaint.setStrokeWidth(2);

Point p1 = **new** Point();

Point p2 = **new** Point();

projection = mapview.getProjection();

projection.toPixels(**this**.GeoPointInicio, p1);

projection.toPixels(**this**.GeoPointFin, p2);

Path path = **new** Path();

path.moveTo(p2.x, p2.y); //punto de inicio

path.lineTo(p1.x,p1.y); //punto de fin

canvas.drawPath(path, mPaint);

}

}

Esta clase LinePathOverlay hereda de Overlay, y se encarga de pintar una línea que va desde una coordenada a otra para formar una ruta. Posee dos constructores, el primero recibe dos objetos GeoPoints, uno de inicio y otro de fin, y además recibe como parámetro un color.

El método encargado de realizar el trabajo en esta clase es draw(), el cual se está sobrescribiendo. En él se hace una mención a **super.draw()** para llamar al método draw de la clase padre **Overlay**. Luego se implementa lo que se desea hacer, se crea un objeto tipo **Paint** con las características de la línea a dibujar, y se setean un conjunto de ciertas propiedades como color y ancho.

Una vez configurada la línea a ser dibujada, se convierten los objetos GeoPoint a coordenadas x-y para poder ser pintadas en la pantalla, mediante el objeto **Projection**. Luego se definen dos objetos de tipo Point donde tendremos el inicio y final en coordenadas x-y de la pantalla. Para esto se utiliza el objeto proyection y luego se llama a su método **toPixels()**.

Una vez que se tienen las coordenadas x-y, y las características de la línea, se debe crear la línea, para esto se debe utilizar el objeto Path.

Con **moveTo()** se indicará el punto de inicio y con **lineTo()** el punto de fin.

Por último falta pintar la línea en la capa con el objeto canvas que proporciona Overlay y el método **drawPath()**, a éste se le pasará la línea y sus características.

A continuación se muestra el fragmento de código donde se pinta la ruta:

List<PuntosRecorrido> listaPuntos = dao.listarPuntosSegunRecorrido(idRecorrido);

**for**(**int** i = 0; i < listaPuntos.size(); i++){

**if**(i > 0){

mapOverlays.add(**new** LinePathOverlay(listaPuntos.get(i-1).getGeoPoint(), listaPuntos.get(i).getGeoPoint()));

**if**(listaPuntos.get(i-1).getIsOrigen() == 1){

control.setCenter(listaPuntos.get(i-1).getGeoPoint());

control.animateTo(listaPuntos.get(i-1).getGeoPoint());

}

}

}

En este fragmento de código se puede observar que se obtiene una lista de puntos. Luego se recorre la lista mediante una sentencia for y se le pasan los puntos al método LinePathOverlay de dos en dos de manera que, en la primer vuelta del ciclo for se dibuja una línea desde un punto a otro, y así sucesivamente se irán dibujando un conjunto de líneas a medida que el ciclo for avance.

**Código para dibujar la capa que representa las paradas del recorrido.**

**public class MyItemizedOverlay extends ItemizedOverlay<OverlayItem> {**

    private List<OverlayItem> mOverlays;

    private Context mContext;

**public MyItemizedOverlay(Drawable defaultMarker,Context context) {**

        super(boundCenter(defaultMarker));

        mOverlays =new ArrayList<OverlayItem>();

        mContext = context;

**}**

@Override

**public void addOverlay(OverlayItem overlay) {**

        mOverlays.add(overlay);

        populate();

**}**

 @Override

protected OverlayItem createItem(int index) {

        return mOverlays.get(index);

}

@Override

public int size() {

        return mOverlays.size();

    }

}

@Override

protected final boolean onTap(int index) {

Toast.makeText(mContext, “Se ha pulsado el elemento “+index,Toast.LENGTH\_SHORT)

.show();

mContext.startActivity(new IntentParada(mContext, mParadas.get(index).getId()));

return true;

}

}

Se puede observar en el código anterior que la clase MyItemizedOverlay extiende de ItemizedOverlay. En ella se ha definido un ArrayList en la cual se almacenarán cada uno de los objetos OverlayItem que se quieren en el mapa. Luego se encuentra un constructor que recibe un objeto Drawable como parámetro, este objeto contendrá la imagen que servirá como marcador que se dibujará automáticamente sobre el mapa para representar cada elemento OverlayItem agregado a MyItemizedOverlay.

Para agregar un elemento OverlayItem a esta clase es necesario tener definido un método que los incorpore al ArrayList definido en ella. Esto lo hacemos mediante addOverlay(OverlayItem overlay), que como se puede observar dentro de él se ha hecho una llamada a populate(), el cual prepara cada elemento OverlayItem agregado, para ser dibujado. Cuando este método se ejecuta, éste hará una llamada a createItem(int) para retornar cada OverlayItem. Se debe sobrescribir este método para poder leer correctamente desde el ArrayList y retornar el objeto OverlayItem desde la posición especificada por el entero dado.También se debe sobrescribir el método size() para retornar el número actual de ítems en el ArrayList.

Luego se define el código para realizar una acción cuando se pulse algún elemento del mapa. La forma más simple de realizar esto es implementar el método **onTap(int)** de el itemizedOverlay, que recibe el índice correspondiente al elemento que se haya pulsado. Con él se podrá, por ejemplo, acceder al objeto de la lista.

Con esto ya se tiene definida la clase MyItemizedOverlay por completo y ya se puede utilizar en el mapa.

A continuación se muestra un fragmento de código que hace uso de la clase MyItemizedOverlay.

**public class MostrarParadas extends MapActivity{**

List<Overlay> mOverlay = mapView.getOverlays();

MyItemizedOverlay markers = new MyItemizedOverlay(getResources().getDrawable(R.drawable.marker), this);

GeoPoint point = new GeoPoint(lat, lon);

OverlayItem overlayitem = new OverlayItem(point, "Parada nº "+numero, nombre);

addOverlay(overlayitem);

for(Entity parada : mParadas){

    markers.addAllParadas(paradas);

}

mOverlays.add(markers);

mapView.postInvalidate();

**}**

En este fragmento de código se puede observar que se comienza obteniendo una referencia a la colección de elementos relacionados al mapa, ya que todos los elementos dibujados sobre un mapa son propiedad del control MapView, cuando haya que añadir algún elemento nuevo, se tendrá que conseguir una referencia a la lista que los contiene, con el método getOverlays ().

Una vez obtenida la referencia a la colección, se crea una instancia de MyItemizOverlay. El constructor de HelloItemizOverlay toma un objeto Drawable con el fin de establecer el contenido como marcador por defecto para todos los elementos que serán dibujados sobre el mapa.

Luego se crea un objeto GeoPoint pasándole como argumento la latitud y longitud, el cual servirá para crear un objeto OverlayItem y agregarlo a nuestro ItemizedOverlay.

Posteriormente se crea un objeto OverlayItem, el constructor de éste recibe un objeto GeoPoint, el cual servirá para ubicarlo en el mapa. Éste también recibe un String como título y otro como subtítulo para la instancia creada de OverlayItem.

Por último, se debe añadir este OverlayItem a la colección obtenida de MyItemizedOverlay y el objeto MyItemizedOverlay al mapView.

* 1. **Utilización de Geocoder**

Ya se mencionó anteriormente que es necesario en ocasiones llevar acabo un proceso de transformación de coordenadas a direcciones. Para esto se utilizará la clase Geocoder.

A continuación se muestra el código que hace uso de la misma.

**public** **class** **MiUbicacion{**

**public** **static** String utilizarGeoCodificacion() **throws** InformacionException{

List<Address> addresses = **null**;

String addressText = "";

**try** {

Geocoder geocoder = **new** Geocoder(context, Locale.getDefault());

addresses = geocoder.getFromLocation(latitud, longitud,1);

**if** (addresses != **null** & addresses.size() > 0) {

Address address = addresses.get(0);

addressText = String.format("%s, %s, %s,%s, %s, %s,%s,%s,%s,%s",

address.getMaxAddressLineIndex() > 0 ? address.getAddressLine(0) : "",

address.getLocality(),

address.getCountryName(),

address.getFeatureName(),

address.getLocale(),

);

}

}**catch**(IllegalFormatException ex){

**throw** **new** InformacionException("No se han recibido datos correctos del servicio de geolocalizacion de Google Maps");

}

**return** addressText;

}

**}**

Se puede observar que la clase MiUbicacion posee un método utilizarGeoCodificacion() el cual es el encargado de retornar los resultados obtenidos del proceso de geocodificación realizados por la clase Geocoder. En este método se instancia un objeto geocoder pasándole los parámetros correspondientes.

Luego se llama al método getFromLocation, el cual recibe como parámetro la latitud y la longitud que formarán la coordenada a ser convertida en una dirección. Este método retornará un objeto del tipo Address, el cual contendrá la dirección necesaria.

Una vez realizado el proceso de conversión y que su resultado sea almacenado en una variable de tipo Address podemos invocar a getLocality() para obtener el nombre de la localidad, y a getCountryName() para obtener el nombre del país asociado a la misma.

CONCLUSIÓN

Una vez finalizado el desarrollo de la aplicación se puede concluir que el objetivo planteado, como así las metas fijadas relacionadas al proyecto, han sido alcanzadas. Ésta aportará una solución real, dado que, los turistas podrán obtener rápidamente información que les permitirá mejorar la calidad de su experiencia en la provincia, ofertándoles la ubicación de la plaza hotelera disponible, de bodegas a las cuales visitar y sitios locales.

Ésta permitirá al viajante estar informado en todo momento a través de una valiosa herramienta de información y comunicación.

Gracias a las tecnologías incluidas en Android, como la utilización del dispositivo GPS, la base de datos SQLite, las librerías para trabajar con XML, entre otras, se ha logrado llevar acabo este desarrollo sin la necesidad de tener que utilizar y aprender otras fuera de él, aunque se puede llegar a potenciar mucho más la plataforma, mediante las distintas librerías existentes de terceros y el uso de servicios web ofrecidos por la API de Google Maps.

También se debe destacar como un punto a favor el gran apoyo de la comunidad de programadores, la documentación oficial, soluciones y prácticas recomendadas.

Como único punto de crítica, se puede decir que Android no es aún un entorno estable, dado que en diversas ocasiones una aplicación (ya que, hasta las desarrolladas por las grandes empresas) puede dejar de funcionar repentinamente.

Por último resaltar, la gran importancia de que esta aplicación este destinada a dispositivos móviles ya que éstos son cada día más utilizados y brindan más funcionalidad a las personas, tanto en negocios como aspectos de su vida diaria.

GLOSARIO

**ADT (Android Development Tools)** - Herramientas de desarrollo de Android.

**ANR (Application Not Responding)** - Aplicación No Responde

**API (Application Programming Interface)** - Interfaz de Programación de Aplicaciones.

**APK (Application PacKage File)** - Archivo de Paquete de Aplicación.

**ARM (Advanced RISC (Reduced Instruction Set Compute) Machines)** - **(Familia de Microprocesadores).**

**AVD (Android Virtual Device)** - Dispositivo Virtual Android.

**DEX (Dalvik Executable)** - Ejecutable Dalvik.

**DOM (Document Object Model)** - Modelo de Objetos del Documento.

**GPS (Global Positioning System)** - Sistema Global de Posicionamiento

**GSM (Global System for Mobile communication)** - Sistema Global para Comunicación Móvil.

**GUI (Grafica User Interface)** - Interfaz Gráfica de Usuario.

**HTML (HyperText Markup Language)** - Lenguaje de Marcado de Hipertexto.

|  |  |
| --- | --- |
| |  | | --- | |  |   **HTTP (Hypertext Transfer Protocol)** - Protocolo de Transferencia de Hipertexto**.** |

**I/O (Imput / Output)** - Entrada / Salida.

**ID (Identification) -** Identificación.

**IDE (Integrated Development Environment)** - Entorno integrado de desarrollo.

**JAR (Java ARchive)** -Archivo Java.

**JAVA ME (Java Micro Edition)** - Java Edición Micro**.**

**JAVA SE (Java Standar Edition)** - Java Edición Estandar**.**

**JDK (Java Development Kit)** -Kit de Desarrollo Java.

**JRE (Java Runtime Environment)** - Entorno de Ejecución Java.

**JSON (JavaScript Object Notation)** - Notación de Objetos de JavaScript.

**JVM (Java Virtual Machine)** - Máquina Virtual Java.

**MD5 (Message-Digest Algorithm 5)** -Algoritmo de Resumen del Mensaje 5.

**MIPS (Microprocessor without Interlocked Pipeline Stages)** - **(Arquitectura de Procesadores).**

**MMS (Multimedia Messaging System)** - Sistema de Mensajería Multimedia.

**NDK (Native Development Kit)** - Kit de Desarrollo Nativo.

**OPEN GL (Open Graphic Library)** - Librería Gráfica Abierta.

**REST (Representational State Transfer)** - Transferencia de Estado Representacional.

**RIM (Research In Motion Limited) - (Ahora BlackBerry)** - **(Compañía canadiense)**

**RPC (Remote Procedure Call)** - Llamadas a Procedimiento Remotos.

**S.O (Operating System)** - Sistema Opertativo.

**SAX (Simple API for XML)** - Api Simple para XML.

**SDK (Software Development Kit)** -Kit de Desarrollo de Software.

**SMS (Short Message Service)** - Servicio de Mensajes Cortos.

**SOA (Service Oriented Architecture)** - Arquitectura Orientada a Servicios.

**SOAP (Simple Object Access Protocol)** - Protocolo Simple de Acceso a Objetos.

**SQL (Structured Query Language)** -Lenguaje de Consulta Estructurado.

**SSL (Secure Sockets Layer**) - Capa de Conexión Segura.

**STAX (Streaming API for XML)** - (Streaming API para XML).

**URL (Uniform Resource Locator)** -Localizador uniforme de recursos.

**UUDI (Universal Description, Discovery and Integration)** - Descripción, descubrimiento e integración universal.

**W3C (World Wide Web Consortium)** -Consorcio de la Web.

**WIFI (Wireless Fidelity)** - **(Conjunto de estándares para redes inalámbricas).**

**WSDL (Web Services Description Language)** -Lenguaje de Descripción de Servicios Web.

**XML (Extensible Markup Language)** -Lenguaje de Marcas Extensible.

ANEXO: CURRICULUM VITAE

**Javier Alberto, Cocco**



* **DATOS PERSONALES**

DNI: 30622385

Fecha de nacimiento: 11 de octubre de 1985

Estado civil: Soltero

Dirección actual: Calle Chile 776. Ciudad

Dirección de correo electrónico alternativo: jackmedrano@gmail.com

Teléfono: 0263-154597967

* **INTERÉS PROFESIONAL**

Mis aspiraciones se centran en participar en el desarrollo de proyectos reales, desafiantes. Ser parte de un equipo de desarrollo y lograr reafirmar mis conocimientos adquiridos como así también aportar los mismos e incorporar nuevos conocimientos, en el área de sistemas informáticos.

* **PREPARACIÓN ACADÉMICA**

**Estudios Secundarios**, Escuela 4-096 Mario Anselmo Sanchez – Rivadavia – Mendoza. Modalidad: Economía y Gestión de las Organizaciones.

**Estudio Universitario:** Universidad Católica Argentina – Facultad San Francisco – Patricias Mendocinas 1475 – Mendoza Capital. Títulos Obtenidos:

**TÍTULO DE PRE-GRADO**

“ANALISTA DE SISTEMAS”

**Actualmente he finalizado el cursado de la carrera y adeudando la tesina para el título de:**

“LICENCIATURA EN SISTEMAS Y COMPUTACIÓN”

* **OTROS ESTUDIOS / CURSOS**

Curso “Java Asec.” Dictado en Universidad Tecnológica Nacional (UTN) Año: 2008. Duración 80 horas reloj. Marco Becas Control F. Aprobado y Asistido

Curso “CURSO JAVA EJB- JPA” Dictado en el Instituto Tecnológico Universitario (ITU) Duración 27 horas reloj. Año 2009. Aprobado y Asistido

* **EXPERENCIA:**

#### Desarrollo de sistema de gestión de stock de repuestos para [Taller Martín Luis](http://www.linkedin.com/search?search=&currentCompany=C&company=Torneria+Martin+Jos%C3%A9+Luis&sortCriteria=R&keepFacets=true&goback=%2Ewvp_*1_*1_*1" \o "Buscar usuarios que hayan trabajado en esta empresa)

Zona Este - Mendoza - Argentina

Mantenimiento de equipos viales en general - Manejo de máquinas herramientas.

#### MARIO VITALE 681 RIVADAVIA-MZA TEL: (263)-4442342.

* **IDIOMAS**

Inglés Técnico I y II, Lectura y escritura.

Actualmente cursando el nivel pre-intermedio en Instituto Liverpool. San Martín Mendoza

* **CONOCIMIENTO DE HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS PARA EL DESARROLLO**

Tecnologías en general: Java, Swing, JPA, JSP.

Tecnologías Móviles: Android. Framework de persistencia: ADA Framework. Framework de GUI: SlidingMenu. ActionBarSherlock

Web Server: Apache Tomcat

Base de datos: My Sql. SQL Server

IDE: Netbeans. Eclipse

* **CONOCIMIENTO DE METODOLOGIAS**

Metodologías formales: RUP

Análisis y Diseño: UML, Design Patterns Grasp, Gof

BIBLIOGRAFÍA

Reto Meier (2012). Professional Android 4 Application Development. Estados Unidos: John Wiley & Sons, Inc.

Mark L. Murphy (2009). The Busy Coder’s Guide to Android Development. Estados Unidos: CommonsWare.

**Jesús Tomás Gironés (2011) (1ra ed.). El gran libro de Android. Mexico: Alfaomega Grupo Editor.**

Sinay Damian (2009).Users- Web Services con C#. Argentina. Ciudad de Buenos Aires: [MP Ediciones](http://www.antartica.cl/antartica/servlet/LibroServlet?action=searchLibrosPorId&orderBy=-1&id=458&criterio=editorial&pagina_actual=1).

Harvey M. Deitel & Paul J. Deitel. Deitel (9na ed.)(2012) Como Programar en Java.Mexico: Editorial Pearson.

Sun Learning Services (2006) . Student Guide. Estados Unidos: Sun Microsystems, Inc **Salvador.**

***Pensamientos varios sobre programación, Android*. [en línea]. Salvador Gómez**

**Dirección URL: <** [**www.sgoliver.net**](http://www.sgoliver.net)**> [Consulta: 11 noviembre 2012].**

**Android Developers. *Layouts.* [en línea].**

**Dirección URL: <**[**http://developer.android.com/intl/es/guide/topics/ui/declaring-layout.html**](http://developer.android.com/intl/es/guide/topics/ui/declaring-layout.html)**>. [Consulta: 16 noviembre].**

Androcode. *Usando MapViews: Desde el principio*. [en línea].

Dirección URL: <<http://androcode.es/2011/12/usando-mapviews-desde-el-principio/>>.

**[Consulta: 25 noviembre 2012].**

**Universidad Politecnica Madrid**.*Tecnologías de Servicios Web*.[en línea].**Marzo 2010.**

Dirección URL:

<<http://pegaso.ls.fi.upm.es/sos/images/documentacion/material_aux_t3/3.1_arquitecturadeserviciosweb.pdf>>.

Google Developers.*Google Maps API Web Services*. [en línea].

Dirección URL: <<https://developers.google.com/maps/documentation/webservices/>>.

[Consulta: 05 diciembre 2012].

**Android Developers. *API Guides. App Resources. Resources Overview* [en línea].**

**Dirección URL:**

<[http://developer.android.com/intl/es/guide/topics/resources/index.html>.](http://developer.android.com/intl/es/guide/topics/resources/index.html%3e.%20%5bConsulta)

[[Consulta](http://developer.android.com/intl/es/guide/topics/resources/index.html%3e.%20%5bConsulta)**: 20 noviembre 2012].**

**Android.*Anroid O.S UserGuide.*[en línea].**

**Dirección URL: <**[**http://support.google.com/android/?hl=es**](http://support.google.com/android/?hl=es)**>.**

**[Consulta: 23 noviembre 2012].**

BIBLIOGRAFÍA

NosinmiUbuntu. *Programación Android. Google Maps en Android*. [en línea].

Dirección URL:

<<http://www.nosinmiubuntu.com/2012/05/google-maps-en-android-iv.html>>.

**[Consulta: 25 noviembre 2012].**

**Android Developers. *Develop. Reference AsyncTask. Class Overview*  [en línea].**

**Dirección URL:**

<[http://developer.android.com/intl/es/reference/android/os/AsyncTask.html>. [Consulta](http://developer.android.com/intl/es/reference/android/os/AsyncTask.html%3e.%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5bConsulta): 30 noviembre 2012].

**David Ehringer. *The Dalvik Virtual Machine Arquitecture*. [en línea]. Marzo 2010.**

**Dirección URL:**

**<** <http://elastos.org/elorg_files/FreeBooks/dalvik/The_Dalvik_Virtual_Machine.pdf> >.

AndroCode. *Action Bar Sherlock .Introducción a ActionBarSherlock*. [en línea].

Dirección URL:

<<http://androcode.es/2012/03/INTRODUCCION-A-ACTIONBARSHERLOCK/>>.

**[Consulta: 02 diciembre 2012].**

**Twitter. *Developers. Documentation REST API v 1.1. GET statuses/ user \_ timeline*** [en línea]

Dirección [URL: <](file:///H:\toda%20la%20info%20para%20la%20tesis\mi%20tesis\%20%3c)<https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline>>.

[Consulta: 10 diciembre 2012].

NOMEDHEL. *Como Programar en Android: Guia5. Uso de SQLITE*. [en línea].

Dirección URL:

<<http://blog.findemor.es/2011/02/como-programar-en-android-guia-5/>>.

[Consulta: 16 diciembre 2012].

Hello Green Rules. *Trucos y ejemplos para desarrolladores Android*. [en Línea].

Dirección URL: <<http://hellogreenrules.blogspot.com.ar/2012/06/crear-tu-propia-galeria-de-imagenes.html> >

[Consulta: 03 enero 2013].

**Android Developers. *Develop.* *Building a Dynamic UI with Fragments*. [en línea].**

**Dirección URL:**

**<**[**http://developer.android.com/intl/es/training/basics/fragments/index.html**](http://developer.android.com/intl/es/training/basics/fragments/index.html)**>.**

**[Consulta: 14 noviembre 2012].**

**Líderes en consumo de Smartphone. ( 2013. Abril 28 ). Diario Uno. Negocios y Economía, p.8.**

1. **Fuente:** [**ww.ist-music.berlios.de/site/middleware-android.html**](http://ist-music.berlios.de/site/middleware-android.html)**.Fecha: octubre 2012.** [↑](#footnote-ref-1)
2. **Fuente:** [**www.androiddevmx.net/java-y-la-maquina-virtual-dalvik**](http://www.androiddevmx.net/java-y-la-maquina-virtual-dalvik)**. Fecha: Octubre 2012.** [↑](#footnote-ref-2)
3. **Fuente:** [**http://www.ikeralbeniz.net/tag/java**](http://www.ikeralbeniz.net/tag/java/)**.Fecha: noviembre de 2012.** [↑](#footnote-ref-3)
4. **Fuente:** [**http://www.ikeralbeniz.net/tag/java**](http://www.ikeralbeniz.net/tag/java/)**.Fecha: noviembre de 2012.** [↑](#footnote-ref-4)
5. **Fuente:** [**www.androideity.com/2011/07/06/ciclo-de-vida-de-una-actividad**](http://www.androideity.com/2011/07/06/ciclo-de-vida-de-una-actividad)**. Fecha: Enero de 2013.** [↑](#footnote-ref-5)
6. **Fuente:** [**http://developer.android.com/guide/topics/ui/overview.html**](http://developer.android.com/guide/topics/ui/overview.html) **(Sitio Oficial de Android).Fecha: Enero de 2013.** [↑](#footnote-ref-6)
7. **Fuente:**[**www.linuxtopia.org/online\_books/android/devguide/guide/practices/design/responsiveness.html**](http://www.linuxtopia.org/online_books/android/devguide/guide/practices/design/responsiveness.html)**. Fecha: Febrero de 2013.** [↑](#footnote-ref-7)
8. **Fuente:** [**http://developer.android.com/guide/practices/tablets-and-handsets.html.(Sitio Oficial de Android)Fecha**](http://developer.android.com/guide/practices/tablets-and-handsets.html.(Sitio%20Oficial%20de%20Android)Fecha)**: Marzo de 2013.**  [↑](#footnote-ref-8)
9. **Fuente:** [**www.ccia.ei.uvigo.es/docencia/SCS/0910/transparencias/Tema4**](http://www.ccia.ei.uvigo.es/docencia/SCS/0910/transparencias/Tema4)**. Fecha: Marzo de 2013.** [↑](#footnote-ref-9)
10. **Fuente:** [**www.facebook.com/RevistaAndroid.Fecha:Marzo**](http://www.facebook.com/RevistaAndroid.Fecha:Marzo) **de 2013.** [↑](#footnote-ref-10)