UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS

FACULTAD DE MATEMÁTICA, FÍSICA Y COMPUTACIÓN

CENTRO DE ESTUDIOS DE INFORMÁTICA

Aplicación Android para simular el comportamiento de métodos de Investigación de Operaciones.

Tesis de Diploma
en
Licenciatura en Ciencia de la Computación

Autor: Pedro Alejandro Sánchez Pérez

Tutores: Dra. Yailen Martínez Jiménez

MSc. Beatríz María Méndez Hernández.

Santa Clara, 2015

El que suscribe, Pedro Alejandro Sánchez Pérez, hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ciencia de la Computación autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

"El valor de un acto realizado reside más en el esfuerzo por llevarlo a cabo que en el resultado."

Albert Einstein

DEDICATORIA

El presente trabajo va dedicado a mi madre Maricela Pérez Cepero, quien ha sido amiga, guía y ejemplo durante toda mi vida, quien se ha sacrificado por mi bienestar y mi futuro. A mi abuela Rosa Cepero y a mi tía Maritza Pérez Cepero por estar siempre a mi lado y apoyarme en la ausencia de mi madre.

AGRADECIMIENTOS

Mis agradecimientos van dirigidos a toda mi familia, amigos y profesores que me han ayudado a vencer esta etapa de mi vida. Específicamente a mi madre por apoyarme y mantenerme durante estos cinco años de carrera universitaria, a mi abuela Rosa y tía Maritza por atenderme y estar al tanto de mí en estos años, durante la misión internacionalista de mi madre. A mi tía Idalmis por sus cuidados y atenciones, a mi prima Naima por brindarme su apoyo y ayuda durante la realización de mi proyecto de tesis, a mi prima Naira por sentarse a mi lado y estar al tanto de mí, aconsejándome en momentos difíciles y de duda. A mis tutoras, las profesoras Beatríz María Méndez Hernández y Yailen Martínez Jiménez por darme la oportunidad de graduarme y haberme ayudado en la realización de mi tesis, a los profesores Yaimara Granados Hondares y Daniel Galvez Lio quienes dedicaron tiempo aclarando diferentes dudas en la realización de mi trabajo de diploma. A todos, gracias.

RESUMEN

En la actualidad, los dispositivos móviles inteligentes han llegado a ser un accesorio indispensable en la vida de cualquier persona. Su portabilidad junto con la posibilidad de realizar las más básicas acciones que van desde la comunicación, ya sea por llamadas o el envío de mensajes de texto (SMS) o multimedia (MMS) y el acceso a internet, hasta otras acciones tales como acceder a las diferentes redes sociales existentes, la recreación mediante juegos, música, videos, contar con servicio GPS, así como muchas otras, son algunas de las características que hacen a estos dispositivos ser usados por una gran cantidad de personas.

En el presente trabajo se implementa una aplicación móvil que permite la simulación de métodos de Investigación de Operaciones en dispositivos móviles inteligentes utilizando las facilidades que brinda su hardware, su cualidad de portabilidad y la gran cantidad de personas que tienen posesión de estos para de esta forma ayudar a los estudiantes que cursan las asignaturas Modelos de Optimización e Investigación de Operaciones y a las profesoras que las imparten.

ABSTRAT

The mobile intelligent devices at the present time have ended up being an indispensable accessory in the life of any person. Their portability together with the possibility of carrying out the most basic actions that they go from the communication, either for flames or the shipment of text (SMS) messages or multimedia and the access to internet, until other such actions as consenting to the existent different social networks, the recreation mediating games, music, videos, to have service GPS, as well as many others.

Presently work is aspired to implement a mobile application that allows the simulation of methods of Investigation of Operations in mobile intelligent devices using the facilities that its hardware, its portability quality and the great quantity of people that have possession of these toasts.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

No se puede negar que en la actualidad los dispositivos móviles inteligentes han llegado a ser un accesorio indispensable, no solo por la necesidad de tener un medio de comunicación a nuestro alcance en todo momento, sino también por el hecho de tener un medio con diversas aplicaciones que nos permiten recrearnos, socializar, acceder a una gran cantidad de información mediante internet, contar con geolocalización a través del GPS, e inclusive, resolver problemas sencillos que se presentan en un momento dado. Debido a esto los dispositivos móviles inteligentes se han convertido en un ítem indispensable en la vida de cualquier persona, pero sobre todo para los jóvenes. Además estos dispositivos son portables, cualidad muy bien recibida por parte del usuario, ya que le permite el poder transportarlos y usarlos prácticamente en cualquier lugar.

Todas estas facilidades vienen dadas por el hardware que presentan, esencialmente por la existencia de un microprocesador y una memoria de sistema interna, así como una gran capacidad de almacenamiento.

Existen gran variedad de sistemas operativos hoy en día para los dispositivos móviles, tales como Android, iOS, Windows Phone, BlackBerry OS y Firefox OS, entre otros, siendo Android el más usado fuera y dentro de nuestro país.

Dadas todas las facilidades que brinda el hardware de estos dispositivos, junto con la cualidad de portabilidad y la gran cantidad de personas que los poseen, se da la posibilidad de que puedan ser útiles herramientas para la docencia y vida académica.

Los algoritmos y métodos de las asignaturas Modelos de Optimización I y II e Investigación de Operaciones a menudo resultan un poco complejos, y debido al número de cálculos que se necesitan realizar para resolverlos y a la cantidad de iteraciones en muchas ocasiones a las profesoras de estas asignaturas no les alcanza el tiempo para resolver en la clase práctica todos los ejercicios que llevan propuestos, además aunque los estudiantes cuentan con un software que simula los principales métodos de estas asignaturas el mismo no posee mucha portabilidad y es necesario poseer una laptop para acceder a la herramienta.

Por lo antes expuesto se plantea como **problema de investigación** de esta tesis que los estudiantes no cuentan con una herramienta portable para la simulación de los principales

métodos de las asignaturas Modelos de Optimización I y II e Investigación de Operaciones para consultar los resultados y comprobar sino tuvieron errores en la ejecución de los mismos.

Para dar solución a este problema se plantea como **objetivo general**:

Desarrollar una aplicación Android que permita la simulación paso a paso de los principales métodos estudiados en Modelos de Optimización e Investigación de Operaciones auxiliándose de herramientas y bibliotecas gráficas del sistema operativo Android.

Este objetivo general se desglosa en los siguientes **objetivos específicos**:

1. Definir las herramientas a emplear para el diseño gráfico de los métodos que trabajan con tablas y grafos.
2. Definir el patrón de programación adecuado para la implementación de la aplicación Android.
3. Implementar una aplicación Android para simular los principales métodos de Investigación de Operaciones.

Para dar cumplimiento al objetivo general y a los objetivos específicos durante la tesis se dará respuestas a las siguientes **preguntas de investigación**:

1. ¿Cuáles son las bibliotecas para trabajar con grafos que más se ajustan a nuestro problema?
2. ¿Cuál es el patrón de programación que mejor se ajusta al código desarrollado?
3. ¿Podrá realizarse un diseño gráfico sencillo y de manera eficiente de los métodos que trabajan con tablas y grafos con las herramientas ya disponibles para Android?
4. ¿Es factible en tiempos de ejecución la simulación de métodos de Investigación de Operaciones en dispositivos móviles con sistema operativo Android?

Con las facilidades que brindan los dispositivos móviles inteligentes y con la realización del presente trabajo se podrá contar con una herramienta que apoye aquellas asignaturas que involucran en su programa de clases métodos de Investigación de Operaciones. Desde el punto de vista de la docencia, el tiempo de clases prácticas y estudio independiente podrá aprovecharse de mejor forma. Además, los estudiantes que serán los más beneficiados a partir de la implementación de esta aplicación móvil, podrán simular métodos de Investigación de

Operaciones de una manera fácil y tendrán la posibilidad de chequear las soluciones de ejercicios resueltos.

La tesis quedó estructurada de la siguiente manera: después de la Introducción, la tesis cuenta con tres capítulos. En el primero se ellos se mencionan y describen fundamentos teóricos del sistema operativo Android. El segundo capítulo menciona las diferentes bibliotecas gráficas utilizadas en la implementación de aplicaciones móviles, y cuál de ellas se ajusta más al problema a tratar, así como las herramientas utilizadas para la representación de grafos. Se abordan los principales patrones de diseño existentes y cuáles de ellos son aplicables al problema en cuestión. El capítulo también aborda aspectos del diseño visual en Android, haciendo énfasis en qué componentes permiten la representación visual de tablas. Finalmente el capítulo contará con una descripción de las herramientas utilizadas para el desarrollo de la aplicación móvil y hará énfasis en las fases de análisis y diseño de la misma. En el capítulo tres se realizará una descripción teórica sobre los diferentes métodos de Investigación de Operaciones implementados en la aplicación. Para finalizar el documento, se formulan las conclusiones y recomendaciones y se relacionan las referencias bibliográficas.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Android es un sistema operativo (SO) móvil basado en el kernel de Linux, fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas; también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en el año 2005 compró. Android fue presentado inicialmente en el 2007 junto con la fundación "Open Handset Alliance", coalición de 48 compañías de hardware, software y telecomunicaciones tales como: Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel, entre otras. Todas estas compañías estaban comprometidas con la promoción de estándares abiertos para dispositivos móviles.

En este capítulo se describe su arquitectura y los principales conceptos y fundamentos que son necesarios conocer para desarrollar aplicaciones para estos dispositivos.

1.1 Características especiales de Android

Anteriormente se había comentado de la existencia de varias plataformas para dispositivos móviles resaltando nuevamente a Windows e iOS. Sin embargo Android presenta una serie de características que lo hacen diferente (Gironés, 2012):

- **Plataforma realmente abierta.** Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es la de poder personalizar el sistema.
- **Portabilidad asegurada.** Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en una gran variedad de dispositivos, tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.
- **Arquitectura basada en componentes inspirados en Internet.** Un ejemplo sería el diseño de interfaz de usuario; esta se hace en XML, lo que permite que una misma aplicación se ejecute en un móvil de pantalla reducida o en un notebook.
- **Filosofía de dispositivo siempre conectado a Internet.** El sistema está programado para optimizar el consumo de energía ante conexiones inalámbricas prolongadas.

- **Gran cantidad de servicios incorporados.** Por ejemplo, localización basada tanto por GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, entre otros.
- **Aceptable nivel de seguridad.** Los programas se encuentran aislados unos de otros, gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de acción (servicios de localización, acceso a internet, entre otros).
- **Optimizado para baja potencia y poca memoria.** Un ejemplo sería la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.
- **Alta calidad de gráficos y sonido.** Gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en 3 dimensiones basados en OpenGL. Incorpora los códec estándar más comunes de audio y video, incluyendo H.264 (AVC), MP3, AAC, entre otros.

Como hemos visto, Android combina interesantes características, lo cual nos ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para dispositivos móviles.

1.2 Versiones de Android

Android ha visto numerosas actualizaciones desde su liberación inicial. Estas actualizaciones al sistema operativo base típicamente arreglan "*bugs"* y agregan nuevas funciones. Generalmente cada actualización del sistema operativo Android es desarrollada bajo un nombre en código de un elemento relacionado con dulces en orden alfabético.

Los números de las versiones hacen referencia a la historia por la que ha pasado el SO Android. Sin embargo lo que es importante es el nivel de la API (Application Programming Interface), ya que este determina en que dispositivos una aplicación puede o no correr basándose en la API bajo la cual se implementó.

En la Figura 1-1 se muestra un gráfico que ilustra las distribuciones globales de las distintas versiones de Android desde diciembre de 2009 a enero de 2015.

Tabla 1- 1 Versiones de Android

Versión	Nivel de API	Nombre
Android 1.0	1	Apple Pie (Tarta de manzana)
Android 1.1	2	Banana Bread (Pan de plátano)
Android 1.5	3	Cupcake (Magdalena)
Android 1.6	4	Donut (Rosquilla)
Android 2.0	5	Éclair (Pastel francés)
Android 2.01	6	Éclair (Pastel francés)
Android 2.1	7	Éclair (Pastel francés)
Android 2.2	8	Froyo (Frozen yogurt)
Android 2.3	9	Gingerbread (Pan de jengibre)
Android 2.3.3	10	Gingerbread (Pan de jengibre)
Android 3.0	11	Honeycomb (Panal de miel)
Android 3.1	12	Honeycomb (Panal de miel)
Android 3.2	13	Honeycomb (Panal de miel)
Android 4.0	14	Ice Cream Sandwich (Sandwich de helado)
Android 4.0.3	15	Ice Cream Sandwich (Sandwich de helado)
Android 4.1	16	Jelly Bean (Pastilla de goma)
Android 4.2	17	Jelly Bean (Pastilla de goma)
Android 4.3	-	Jelly Bean (Pastilla de goma)
Android 4.4	19	Kitkat (Tableta de chocolate con leche)
Android 5.0	-	Lollipop (Piruleta)
Android 5.1	-	Lollipop (Piruleta)

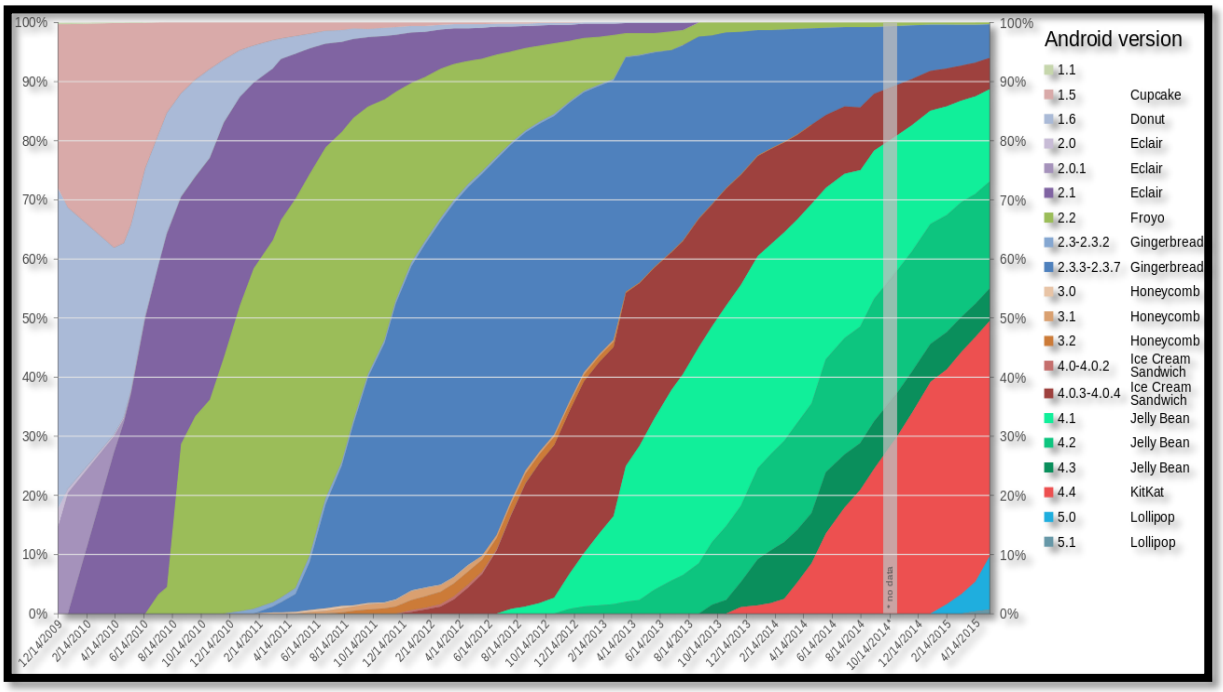

Figura 1- 1 Distribución global de las diferentes versiones de Android 2009-2015

1.3 Arquitectura del SO Android

EL SO Android puede ser visualizado como un pastel conformado por un conjunto de capas. Cada capa tienes sus propias características y propósitos, y brinda los servicios necesarios para el funcionamiento de las capas superiores a ella. Una de las características más importantes es que todas las capas están basadas en software libre.

1.3.1 Kernel de Linux

Android es un SO libre basado en el kernel de Linux, versión 2.6. Esta capa se encarga de los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, modelo de controladores, entre otros. La capa "*Linux Kernel*" se sitúa en el fondo del modelo estructural del sistema (Figura 1-2) y actúa además como capa de abstracción entre el hardware y el resto del software (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012).

Hay tres razones fundamentales del porqué Android se basa en el kernel de Linux (Gargenta, 2011):

- Portabilidad: Linux es una plataforma portable que es relativamente fácil de compilar en varias arquitecturas de hardware. Lo que Linux le aporta a Android es un nivel de abstracción del hardware. Esto permite que no que no haya que preocuparse mucho de las funcionalidades subyacentes a este. La mayoría de las partes de bajo nivel de Linux están escritas en código portable en C, lo que permite que Android sea portado por una gran variedad de dispositivos.
- Seguridad: Linux es un sistema altamente seguro.
- Funcionalidades: Linux viene con un gran conjunto de funcionalidades útiles. Android se apoya en muchas de estas funcionalidades como soporte para la administración de memoria, de energía y para trabajos con la red.

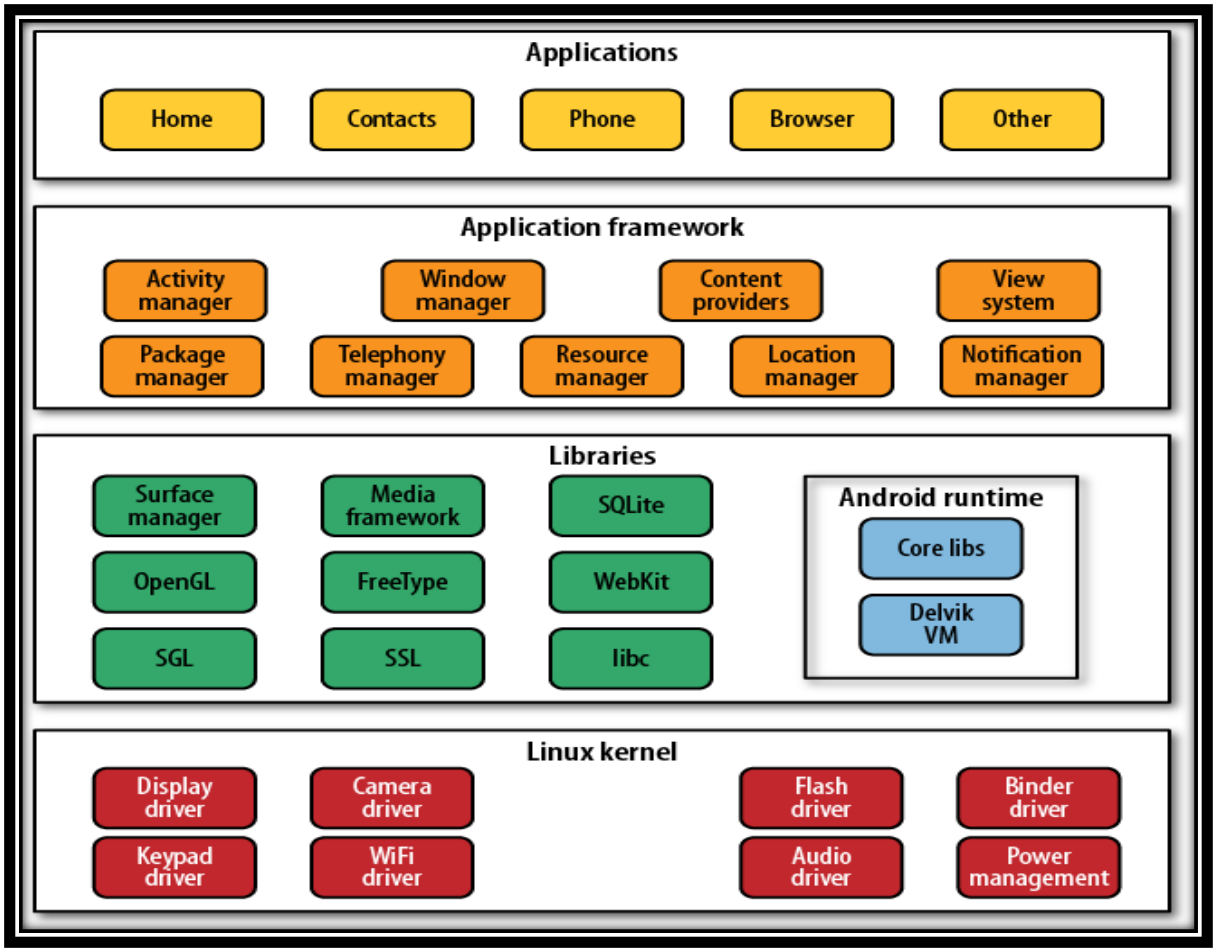

Figura 1- 2 Estructura del SO Android

1.3.2 Bibliotecas Nativas

Situada encima del kernel se encuentra la capa "*Libraries*" (Figura 1-2), la cual contiene las bibliotecas nativas de Android. Estas bibliotecas compartidas están escritas en C/C++, compiladas por la arquitectura particular del hardware usado por el dispositivo y preinstalado por el vendedor del dispositivo. Esta capa provee los servicios necesarios a la capa "*Application Framework*" (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012).

Las bibliotecas más importantes dentro de esta capa son (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012):

- Surface Manager: Permite al sistema crear todo un conjunto de interesantes efectos, tales como, ver a través de las ventanas, elegantes transiciones, entre otros. Maneja el acceso al subsistema de representación gráfica 2D y 3D.
- OpenGL: Biblioteca gráfica de Android basada en OpenGL ES 1.0 API, la cual permite el trabajo con gráficos en 3D. Las bibliotecas utilizan el acelerador de hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- SGL: motor de gráficos 2D.
- Media Framework: Permite la reproducción de audio y video en diferentes formatos, así como de imágenes.
- SQLite: Contiene un potente y ligero motor de base de datos relacionales para todas las aplicaciones.
- WebKit: Contiene el motor de navegación web para la visualización rápida de contenidos HTML.
- SSL: Proporciona servicios de encriptación Secure Socket Layer.
- FreeType: Fuentes de bitmap y renderizado vectorial.
- System C Library: una derivación de la biblioteca BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.

1.3.3 Android Runtime

Situada también encima del kernel se encuentra la capa "*Android Runtime*" (Figura 1-2), que incluye el núcleo de bibliotecas de Java y la máquina virtual Dalvik.

En esencia la máquina virtual Dalvik es una máquina virtual de Java implementada por Google, pero optimizada para dispositivos móviles. Todo el código escrito para Android será escrito en Java y ejecutado por esta máquina virtual. Algunas características importantes de Dalvik son: está basada en registros, cada aplicación corre su propio proceso Linux con su propia instancia de la máquina virtual, y delega al Kernel de Linux algunas funciones como el *threading* y el manejo de la memoria a bajo nivel (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Satya Komatineni, 2012, Meier, 2012, Rivera et al., 2012).

La máquina virtual Dalvik difiere de la de Java en dos importantes características (Burnette, 2009):

- La de Dalvik ejecuta archivos .dex, los cuales son generados en tiempo de compilación a partir de los archivos estándar .class y .jar. Estos archivos .dex (formato optimizado para ahorrar memoria) son más compactos y eficientes que los .class, lo cual representa una importante característica por los límites de memoria y batería que presentan los dispositivos que utilizan Android.
- El núcleo de bibliotecas de Java que viene con Android es diferente de las bibliotecas Java Standard Edition (Java SE) y Java Mobile Edition (Java ME). La principal diferencia está dada por el remplazo de las bibliotecas AWT y Swing por otras bibliotecas específicas de interfaz de usuario. También Android introdujo nuevas funcionalidades al estándar de Java que soportan la mayoría de las funcionalidades del estándar.

En la siguiente figura se tiene un código fuente escrito en Java, el cual es compilado por el compilador de Java, generándose así el byte code correspondiente. Seguidamente es ejecutado por la máquina virtual de Java. En Android, las cosas son prácticamente iguales hasta el punto de generación del byte code de Java, el cual es recompilado, generándose el byte code correspondiente para ser ejecutado por la máquina virtual Dalvik.

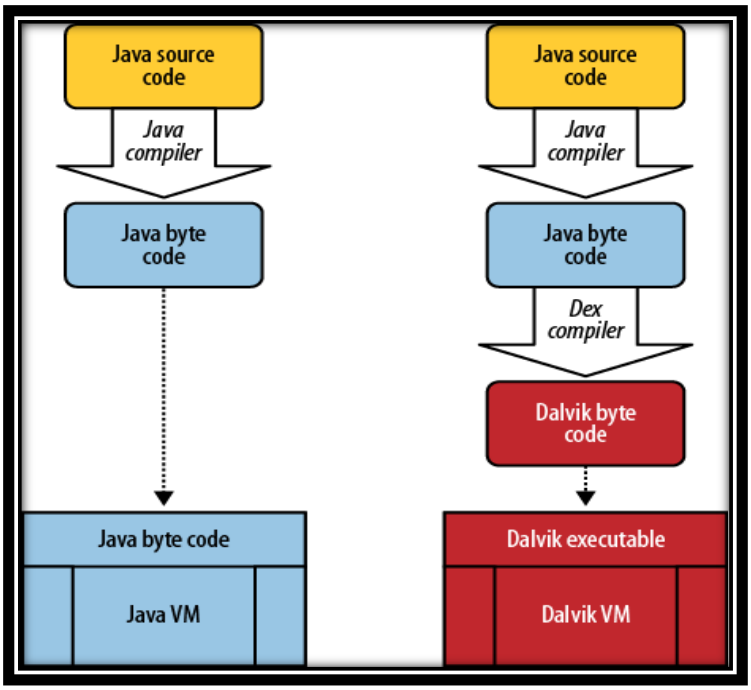

Figura 1- 3 MV Java v/s MV Dalvik

1.3.4 Application framework

La capa "*Application Framework*" (Figura 1-2), situada encima de las capas "*Libraries*" y "*Android Runtime*", brinda bloques de construcción de alto nivel para crear una aplicación. Esta es la parte de la plataforma mejor documentada y más ampliamente cubierta debido a que es la capa que permite a los desarrolladores ser creativos e implementar aplicaciones fantásticas. Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, entre otros). El "framerwork" viene ya preinstalado con Android, pero brinda la posibilidad de extenderlo con nuevos componentes según sea necesario. El diseño de esta capa está orientado a simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a restricciones de seguridad). Este mismo mecanismo permite a los usuarios remplazar componentes (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012).

Las componentes más importantes de esta capa son las siguientes (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012):

- Activity Manager: Controla el ciclo de vida de las aplicaciones y mantiene un "backstack" (pila de vuelta) común para la navegación del usuario.
- Content Providers: Este objeto encapsula información que necesita ser compartida entre aplicaciones (ej. contactos del dispositivo).
- Resource Manager: Gestiona los recursos de las aplicaciones. Dígase recursos de una aplicación a todo lo que va con esta y que no es código.
- Notification Manager: Maneja los eventos tales como, arribo de mensajes, alertas de proximidad, citas, entre otros.
- Location Manager: Maneja todo lo referente a la percepción de localización, a través de dispositivos GPS o el manejo de acelerómetros.
- Views: extenso conjunto de vistas (parte visual de los componetes).

1.3.5 Applications

EL mayor nivel del diagrama de la arquitectura de Android es la capa "*Applications*" (Figura 1-2). Esta es la capa que representa la parte visible del témpano de hielo de Android y la única capa que el usuario es capaz de percibir, ya que es incapaz de ver las acciones que ocurren debajo del nivel del agua. Contiene todas las aplicaciones con las que el usuario interactúa haciendo uso de la pantalla del dispositivo móvil (Burnette, 2009, Gironés, 2012, Lee, 2012, Gargenta, 2011, Meier, 2012, Rivera et al., 2012).

El APK

Una aplicación en Android es un archivo con extensión .apk (APK). Las APK están constituidos por tres componentes principales, los cuales son (Gargenta, 2011, Dornin et al., 2011):

- Ejecutable Dalvik: Este componente es todo el código fuente en Java compilado como un ejecutable (.dex) Dalvik. Es el código que ejecuta la aplicación.
- Recursos: Los recursos son todo lo que no es código. Las aplicaciones pueden contener varios archivos, tales como, audio, video, imágenes, así como numerosos archivos XML que describen layouts, paquetes de lenguaje, entre otros.
- Bibliotecas nativas: Opcionalmente, una aplicación puede incluir algún código nativo, tales como bibliotecas en C/C++. Estas bibliotecas pueden ser unidas y empaquetadas en la APK.

Firma de Aplicaciones

Las aplicaciones Android deben ser firmadas antes de poder instalarse en un dispositivo. Por propósitos de desarrollo, inicialmente las aplicaciones son firmadas con una llave ya existente en la plataforma de desarrollo. Sin embargo, cuando la aplicación sale de la fase de desarrollo puede ser firmada con una llave propia del desarrollador (Gargenta, 2011).

Aplicaciones nativas

En todos los dispositivos que llevan Android normalmente vienen por defecto con una suite de aplicaciones ya preinstaladas que son parte de "*Android Open Source Proyect*" (AOSP), incluyendo, pero no necesariamente limitado a las siguientes aplicaciones (Meier, 2012):

- Cliente de correo electrónico
- Aplicación para el manejo de SMS
- Una suite completa PIM (manejador de información personal), que incluye un calendario y una lista de contactos.
- Un navegador web basado en WebKit
- Una aplicación de música y una galería de imágenes
- Una cámara y aplicación de grabación de videos
- Una calculadora
- Aplicación "Home Screen"
- Despertador
- Radio

También estos dispositivos vienen en la mayoría de los casos con una suite de aplicaciones propietarias, normalmente pertenecientes a Google y a la compañía que representa la marca del dispositivo (Meier, 2012).

Ejecución de aplicaciones en Android

En Android, hay una aplicación en primer plano, la cual utiliza la pantalla casi en su totalidad exceptuando la línea de estado ubicada en la parte superior de la pantalla. Cuando el dispositivo móvil es encendido, la primera aplicación que se muestra es la aplicación "Home Screen". Esta aplicación normalmente muestra una imagen de fondo, widgets y aplicaciones nativas, así como una lista desplegable que contiene las demás aplicaciones instaladas en el dispositivo. Cuando

se ejecuta una aplicación, Android la inicia y la trae a primer plano. Desde esa aplicación se pueden invocar otras aplicaciones u otras pantallas de la misma aplicación, una y otra vez. Todas estas aplicaciones y pantallas son guardadas en una pila que mantiene el Administrador de Actividades. En cualquier momento, el usuario puede presionar el botón de retroceso y volver a la pantalla anterior (Burnette, 2009).

1.4 Principales bloques de construcción en Android

Los bloques de construcción en Android son componentes usados por los desarrolladores para la construcción de aplicaciones. Estos ítems conceptuales son similares a los elementos básicos utilizados en la construcción de aplicaciones en otros lenguajes, tales como Java o .NET.

Conceptos como ventana, control, eventos o servicios, son conceptos utilizados también por Android, pero con un pequeño cambio en la terminología y el enfoque.

1.4.1 Actividad (Activity)

Una ***actividad*** normalmente es una pantalla de la aplicación que se visualiza en la pantalla del dispositivo móvil, siendo esta el componente principal de la interfaz gráfica de una aplicación Android y la parte más visible de esta. Se puede pensar en una ***actividad*** como el elemento análogo a una ventana en cualquier otro lenguaje visual. Una aplicación típicamente tiene múltiples ***actividades***, donde el usuario avanza y retrocede sobre estas. Las diferentes ***actividades*** creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Toda actividad ha de pertenecer a una clase descendiente de la clase ***Activity*** (Gironés, 2012, Burnette, 2009, Gargenta, 2011, Oliver, 2011, Dornin et al., 2011, Meier, 2012).

Diferencia entre procesos y actividades

Internamente, cada pantalla de la interfaz de usuario es representada por una clase "Activity" (actividad) como anteriormente se expuso, donde cada actividad tiene su propio ciclo de vida. Una aplicación es un conjunto de una o varias actividades más un proceso de Linux para contenerlas. Sin embargo, en Android, una aplicación puede estar viva inclusive si su proceso ha sido eliminado. Puesto de otra manera, el ciclo de vida de una actividad no está atado al ciclo de vida del proceso. Los procesos son solo contenedores desechables para las actividades (Burnette, 2009).

Ciclo de vida de una actividad

El lanzamiento de una actividad es un proceso costoso. La ejecución de esta acción involucra crear un nuevo proceso, reservar memoria para todos los objetos de la interfaz gráfica, inflar todos los objetos a partir de los XML que describen los layouts, y mostrar todo esto en la pantalla. Debido al gran trabajo que consiste lanzar una actividad, sería un desperdicio tirar la actividad una vez que el usuario pasara a una nueva pantalla de la aplicación. Para evitar este desperdicio, el *ciclo de vida* de una actividad es manejado por el "Activity Manager". El "Activity Manager" es responsable de crear, destruir y manejar las actividades. Cuando una aplicación es ejecutada por primera vez, el "Activity Manager" crea la actividad y la muestra en la pantalla. Después, cuando el usuario cambia de actividad, la anterior a ella es puesta en el "backstack" (pila que mantiene el "Activity Manager"). De esta manera, si se retrocede hasta una actividad anterior, esta puede iniciarse nuevamente de una manera más rápida. Actividades en segundo plano que llevan un tiempo sin usarse son destruidas para dar cabida a una nueva actividad activa (Burnette, 2009, Gargenta, 2011, Dornin et al., 2011, Meier, 2012, Rivera et al., 2012, Satya Komatineni, 2012).

Este mecanismo está diseñado para ayudar a mejorar la velocidad de la interfaz de usuario y en general la experiencia del usuario.

Durante su tiempo de vida, cada actividad de una aplicación Android puede estar en diferentes estados (Burnette, 2009, Gargenta, 2011, Meier, 2012, Rivera et al., 2012) (Figura 1-3):

Starting State

Cuando una actividad no existe en memoria, se encuentra en "starting state" (estado de inicialización). Mientras esté iniciando, la actividad pasará por todo un conjunto de llamadas a diferentes métodos. Eventualmente la actividad pasará a "running state" (estado de ejecución).

Esta transición del estado de inicialización al estado de ejecución es una de las más costosas operaciones en términos de tiempo de cómputo, y afecta directamente la vida de la batería del dispositivo. Principal razón por la que las actividades que no están siendo mostradas no son destruidas automáticamente.

Running State

Una actividad en estado de ejecución, es una actividad que se está visualizando en la pantalla e interactuando con el usuario. También se dice que la actividad está en "focus", en otras palabras, que todas las interacciones del usuario están siendo manejadas por la actividad. Por lo tanto, siempre hay una sola actividad en estado de ejecución.

La actividad que se encuentre en este estado tiene prioridad en términos de obtener memoria y recursos necesarios para correr lo más rápido posible.

Paused State

Cuando una actividad no está en "focus", es decir, no hay interacción con el usuario, pero aun es visible en la pantalla, se dice que la actividad está en "paused state" (estado pausado). Esto no es un escenario típico, porque la pantalla del dispositivo es generalmente pequeña, y una actividad está o bien ocupando toda la pantalla o en segundo plano. Normalmente ocurre cuando cuadros de dialogo salen en frente de la actividad, causando que entre en estado pausado. Todas las actividades pasan por este estado antes de entrar en "stopped state" (estado detenido).

Las actividades en estado pausado aún siguen teniendo una alta prioridad en términos de obtener memoria y otros recursos.

Stopped State

Cuando una actividad no es visible, pero sigue estando en memoria, se dice que está en estado detenido. Una actividad detenida puede volver a ser mostrada en pantalla convirtiéndose en una actividad en ejecución nuevamente o puede ser destruida y removida de memoria en cualquier momento para ceder dicha memoria a nuevas actividades.

Destroyed State

Una actividad destruida no se encuentra en memoria. El "Activity Manager" decide que esta actividad no es ya necesaria y la remueve. Antes de que la actividad sea destruida, se pueden efectuar ciertas acciones, tales como guardar cualquier información no guardada. Sin embargo no hay garantías que una actividad entre en estado detenido antes de ser destruida. También es posible que una actividad sea destruida estando en estado pausado.

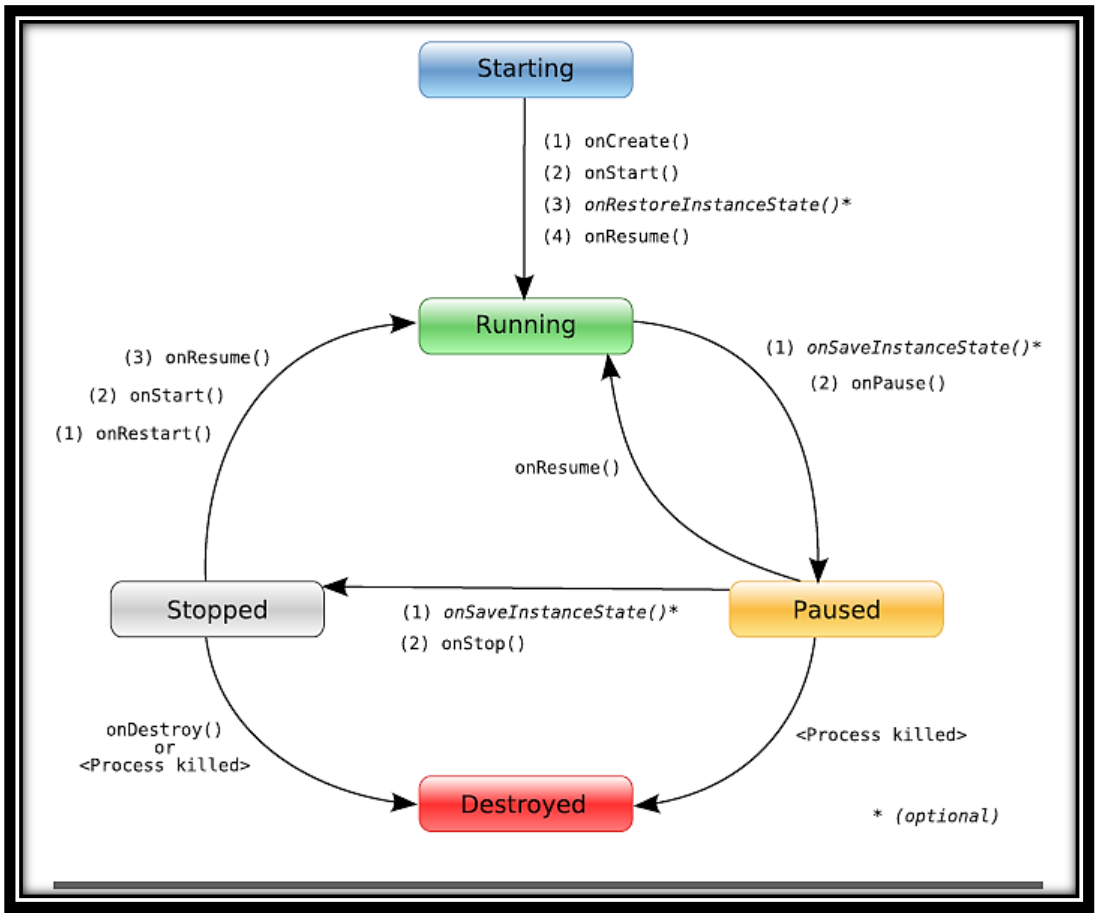

Figura 1- 4 Ciclo de vida de una Actividad en Android

Métodos onAction()

Los desarrolladores no tienen control sobre en qué estado se encuentra una actividad, esto solo es manejado por el sistema. Sin embargo, el desarrollador tiene la oportunidad de determinar que sucede durante la transición de un estado a otro, a partir de llamadas a los métodos ***onAction()***.

Estos métodos son sobrescritos en la clase "Activity" y Android los llama en el momento apropiado (Burnette, 2009, Dornin et al., 2011, Rivera et al., 2012, Satya Komatineni, 2012, Darwin, 2012):

- onCreate(Bundle): Es llamado cuando la actividad se inicia por primera vez. Es usado para realizar una inicialización en primer tiempo, como la creación de la interfaz de

usuario. Este método toma un parámetro que toma null o alguna información de estado anteriormente salvada por el método onSaveIntanceState().

- onStart(): Indica cuando la actividad va a ser visualizada en la pantalla.
- onResume(): Es llamado cuando la actividad puede empezar a interactuar con el usuario.
- onPause(): Es ejecutado cuando la actividad no está interactuando con el usuario o está al pasar a segundo plano, normalmente debido a que otra actividad ha sido lanzada al primer plano.
- onStop(): Es llamado cuando la actividad no es visible al usuario y no va a ser necesitada dentro de un tiempo. Si la memoria está casi llena, este método nunca será llamado, el sistema simplemente terminará la actividad.
- onRestart(): Si este método es llamado, indica que la actividad va a ser nuevamente mostrada al usuario a partir del estado "stopped".
- onDestroy(): Es llamado justo antes de que la actividad sea destruida. Si la memoria está casi llena, el método no es llamado, el sistema simplemente terminará la actividad.
- onSaveInstanceState(): Android llama a este método para permitir a la actividad salvar el estado de la instancia. Normalmente este estado no necesita ser sobrescrito debido a que la implementación por defecto salva el estado de todos los controles de interfaz de usuario automáticamente.
- onRestoreInstanceState(): Es llamado cuando la actividad está siendo reinicializada del estado anterior guardado por el método onSaveInstanceState(). La implementación por defecto restaura el estado de la interfaz de usuario.

En la Figura 1-5 se muestra cada uno de los de los métodos anteriormente descritos haciendo énfasis en tres aspectos: momento en que se llama, acción a realizar y aspectos a tener en cuenta. En la Figura 1-6 se muestra el riesgo de que el sistema destruya una actividad dada, al encontrarse en un estado determinado.

Método	¿Cuándo se llama?	¿Qué debería hacer?	¿Qué hay que tener en cuenta?
onCreate()	Al crearse	Crear views, unir datos a listas, etc	Entrega datos en un Bundle si la activity ha sido re-creada
onRestart()	Fue pausada y vuelve a ejecutarse		
onStart()	Al hacerse visible para el usuario	Recuperar el estado	
onResume()	Al comenzar la iteración con el usuario		En este momento la activity se sitúa en la cima de la pila
onPause()	Al perder el foco (cuando ya no interactúe con el usuario)	Se suele usar para guardar los cambios no guardados, para detener animaciones u otras consuman procesador	Cuando el onPause() termine, se realizará el onResume() de la nueva activity. Se recomienda código rápido
onStop()	Al no ser visible para el usuario	Guardar el estado	Ejemplo: otra nueva activity ha hecho su onResume() y a cubierto a esta
onDestroy()	Justo antes de ser destruida	Liberar sus recursos y limpiar su estado	Por la llamada a finish() o porque Android la haya matado

Figura 1- 5 Métodos onAction()

Estado	Causa	Mantiene estado interno	Unido al gestor de ventanas	Riesgo de que Android la mate
Activa o en ejecución	Si está en primer plano (encima de la pila)	Sí	Sí	Bajo
Pausada	Si pierde el foco pero está todavía visible	Sí	Sí	Medio
Parada	Está completamente oculta	Sí	No	Alto
Matada	No existe	No	No	-

Figura 1- 6 Riesgo de destrucción de una actividad ante un estado dado

1.4.2 Intent

Los ***intents*** son mensajes o peticiones que son enviados entre los bloques de construcción más importantes. Estos disparan el inicio de una actividad, pasan información de una actividad a otra, le dicen a un servicio que inicie o se detenga, o simplemente juegan el papel de "broadcast". Un ***intent*** es asincrónico, lo cual significa que el código que lo mandó no necesariamente tiene que esperar a que la acción del ***intent*** se ejecute (Gironés, 2012, Burnette, 2009, Gargenta, 2011, Oliver, 2011, Dornin et al., 2011, Meier, 2012, Rivera et al., 2012).

Este tipo de bloque de construcción puede ser explícito o implícito. En un ***intent*** explícito, el emisor especifica quien es el receptor al cual va dirigido. En un ***intent*** implícito, el emisor

especifica el tipo de receptor. Por ejemplo, una actividad puede mandar un ***intent*** diciendo que simplemente quiere abrir una página web determinada. En ese caso, cualquier aplicación que sea capaz de abrir la página web competirá para realizar esta acción. El sistema en estos casos le muestra al usuario las aplicaciones que están compitiendo (Gironés, 2012, Burnette, 2009, Gargenta, 2011, Oliver, 2011, Dornin et al., 2011, Meier, 2012, Rivera et al., 2012).

La Figura 1-7 muestra como los ***intents*** pueden usarse para saltar entre diferentes actividades, dentro de la misma aplicación o en otra cualquiera.

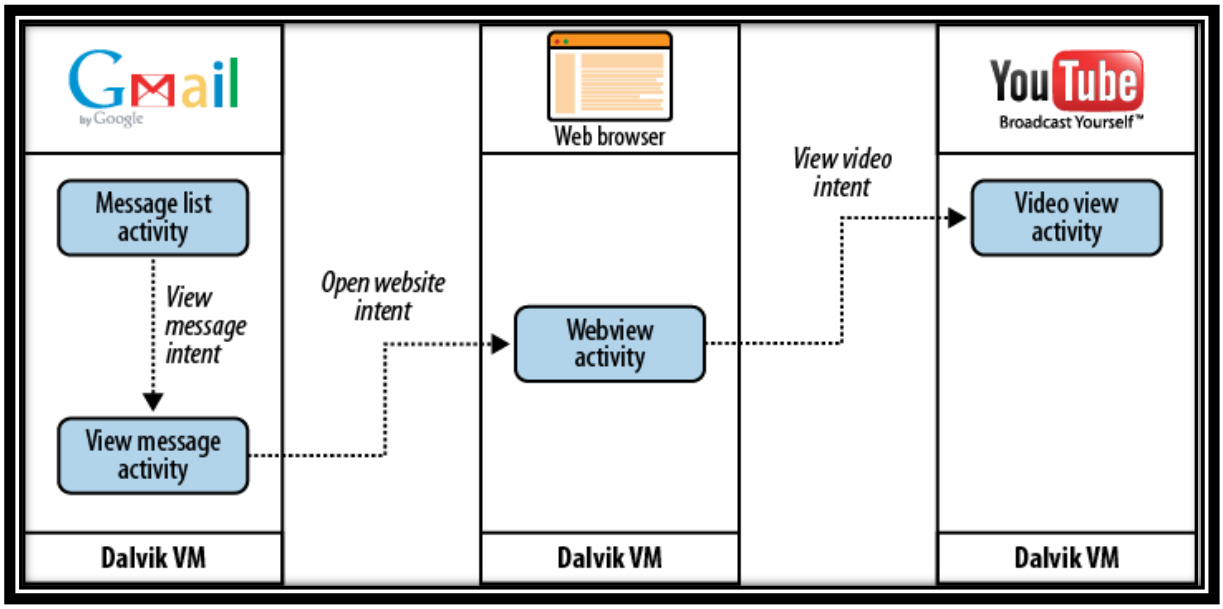

Figura 1- 7 Intents

1.4.3 Vista (View)

Las ***vistas*** son los elementos que componen la interfaz de usuario de una aplicación. Son por ejemplo, un botón, una entrada de texto, entre otros. Todas las ***vistas*** van a ser objetos dependientes de la clase ***View***, y por tanto pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las ***vistas*** utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajo es muy similar a la definición de una página web utilizando código HTML (Gironés, 2012, Oliver, 2011).

1.4.4 Layout

Un ***layout*** es un conjunto de vistas agrupadas de una determinada forma. Se va a disponer de diferentes tipos de ***layouts*** para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los ***layouts*** también son objetos descendientes de la clase ***View***. Igual que las vistas, los ***layouts*** pueden ser definidos en código Java, aunque la forma habitual de definirlos es utilizando código XML (Gironés, 2012).

1.5 Estructura de un proyecto en Android

Cuando se crea un nuevo proyecto Android en Eclipse se genera automáticamente la estructura de carpetas necesaria para poder implementar posteriormente la aplicación. Esta estructura será común a cualquier aplicación, independientemente de su tamaño y complejidad (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

La Figura 1-11 muestra los elementos creados inicialmente para un nuevo proyecto Android.

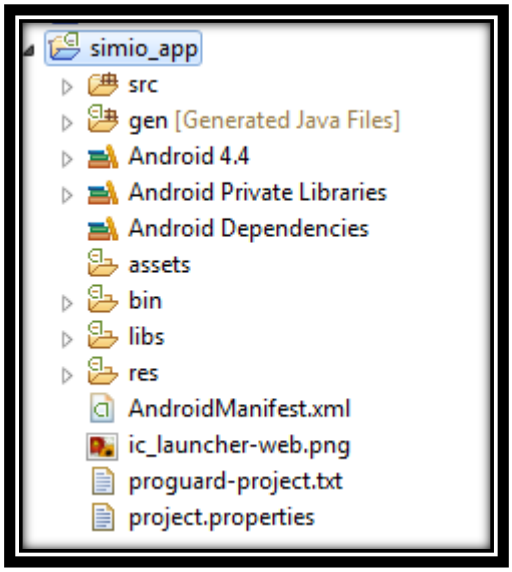

Figura 1- 8 Estructura de una aplicación Android

Carpeta /src/

La carpeta /src/ contiene todo el código fuente de la aplicación, el código de la interfaz gráfica, las clases auxiliares, entre otros. Inicialmente Eclipse creará el código básico de la pantalla principal de la aplicación, siempre bajo la estructura del paquete Java definido (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

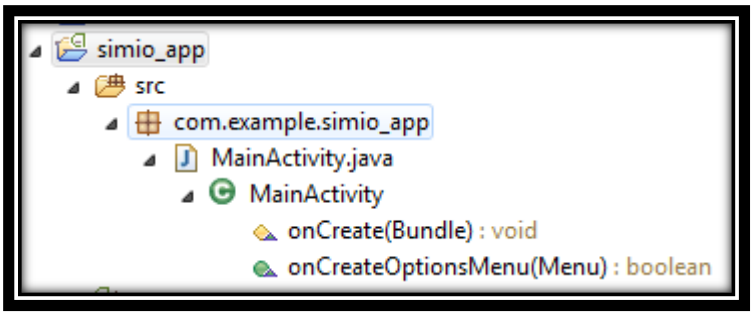

Figura 1- 9 Carpeta /src/

Carpeta /res/

La carpeta ***/res/*** contiene todos los ficheros de los recursos necesarios para el proyecto, díganse imágenes, videos, cadenas de texto, entre otros. Los diferentes tipos de recursos se distribuyen en diferentes carpetas (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

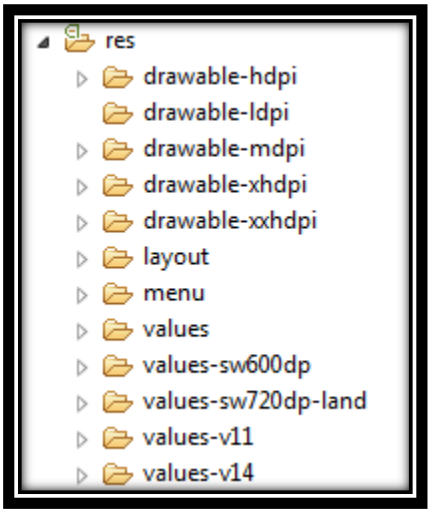

Figura 1- 10 Carpeta /res/

La distribución de los recursos sobre las diferentes carpetas se realiza basado en el tipo del recurso (Tabla 1-2) (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

Tabla 1- 2 Carpeta /Res/

Carpeta	**Descripción**
/res/drawable/	Contiene las imágenes de la aplicación. Para utilizar diferentes recursos dependiendo de la resolución del dispositivo se suele dividir en varias supcarpetas: ▪ /drawable – ldpi ▪ /drawable – mdpi ▪ /drawable – hdpi ▪ /drawable – xhdpi ▪ /drawable – xxhdpi
/res/layout/	Contiene los ficheros de definición de las diferentes pantallas de la interfaz gráfica. Para definir distintos layouts dependiente de las dimensiones de pantalla se suele dividir en 4 carpetas: ▪ layout-small ▪ layout-normal ▪ layout-large ▪ layout-xlarge Para definir distintos layouts dependiente de la orientación del dispositivo se suele dividir en dos subcarpetas: ▪ /layout-<tipo de dimensión> ▪ /layout -<tipo de dimensión>-land
/res/anim/	Contiene la definición de las animaciones utilizadas por la aplicación.
/res/menu/	Contiene la definición de los menús de la aplicación.
/res/values/	Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), entre otros.
/res/xml/	Contiene los ficheros XML utilizados por la aplicación
/res/raw/	Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

Carpeta /gen/

Todo el contenido de esta carpeta es generado automáticamente por el SDK (Software Development Kit). Cada vez que generemos nuestro proyecto la maquinaria de compilación de Android genera por nosotros una serie de ficheros fuente en Java dirigidos al control de los recursos de la aplicación (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

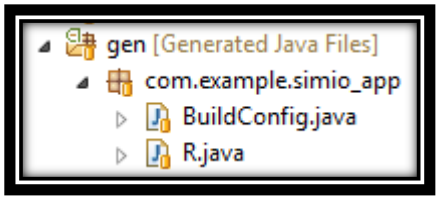

Figura 1- 11 Carpeta /gen/

Los archivos que podemos encontrar son (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012):

- BuildConfig.java: Indica si la aplicación está en desarrollo.
- R.java: Clase automática que asocia el contenido de la carpeta ***/res/*** con identificadores para poder llamar a los recursos de la aplicación desde Java.

Carpeta /assets/

Contiene recursos auxiliares necesarios para la aplicación, como por ejemplo ficheros de configuración, de datos, de texto, HTML, bases de datos, entre otros. La diferencia entre los recursos incluidos en la carpeta ***/res/raw/*** y los incluidos en la carpeta ***/assets/*** es que para los primeros se generará un ID en la clase ***R*** y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Para los segundos sin embargo no se generarán ID alguno, y se podrá acceder a ellos por su ruta como a cualquier otro fichero del sistema (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012).

Android x.y

Bibliotecas oficiales y liberadas de Android de la **versión elegida** (versión especificada por el desarrollador que señala la versión de Android tope en que se podrá ejecutar una aplicación; normalmente es la máxima versión liberada de Android) (Gironés, 2012, Ramón Invarato Menéndez, 2014, Lee, 2012).

Carpeta /bin/

Esta carpeta contiene los archivos generados por el ADT (Android Development Tools) durante el proceso de compilación. En lo particular es donde se genera el ***APK*** de la aplicación. (Ramón Invarato Menéndez, 2014, Lee, 2012)

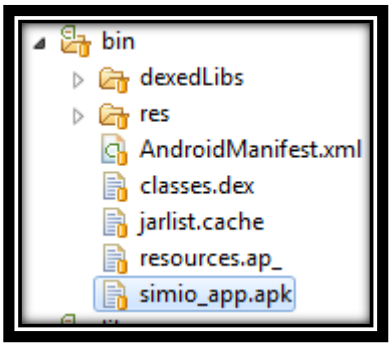

Figura 1- 12 Carpeta /bin/

Carpeta /libs/

En este lugar es donde pondremos las bibliotecas JAR asociadas al proyecto. Esta carpeta viene con una biblioteca llamada "android-support-vX.jar" que incluye, tanto soporte para versiones antiguas de Android como utilidades beta o que todavía no han sido liberadas (Ramón Invarato Menéndez, 2014).

AndroidManifest.xml

Contiene la definición en XML de la información esencial de la aplicación que debe conocer el sistema operativo, antes de que la ejecute. Entre otras cosas, contiene (Gironés, 2012, Oliver, 2011, Ramón Invarato Menéndez, 2014, Lee, 2012):

- Aspectos de identificación (versión, icono, nombre, …)
- Los componentes esenciales de una aplicación tales como: actividades, servicios, BroadcastReceivers y ContentProvider.
- Los permisos para acceder a las partes protegidas de la API o para interactuar con otras aplicaciones.
- Declara el nivel mínimo y máximo del API de Android que requiere la aplicación.
- La lista de bibliotecas que están vinculadas.

Proguard-Project.txt

Configuración de ProGuard para que reduzca, optimice y ofusque el código (Ramón Invarato Menéndez, 2014).

Project.properties

Contiene la configuración del proyecto y el destino para ser generado. La generación de este fichero es de forma automática (Ramón Invarato Menéndez, 2014).

1.6 Conclusiones parciales

Se describen los principales elementos del sistema operativo Android y su arquitectura. Después se estudiaron los principales bloques de construcción del sistema eligiendo los que era necesario utilizar para la implementación de este trabajo. Para finalizar el capítulo se define la estructura que va a tener el proyecto definiendo las funcionalidades de las diferentes carpetas que lo integran.

CAPÍTULO 2. DESARROLLO DE LA APLICACIÓN SIMIO

En el presente capítulo se abordarán las temáticas específicas con el desarrollo de la aplicación móvil "SIMIO". Se describirán las herramientas utilizadas para el desarrollo de la aplicación así como las fases de análisis, diseño e implementación de la misma.

2.1 Entorno de desarrollo

Para el desarrollo de la aplicación móvil SIMIO se utilizará un potente y moderno entorno de desarrollo. Al igual que Android, todas las herramientas que se utilizarán están basadas en software libre. Aunque existen varias alternativas en la actualidad para el desarrollo de aplicaciones en Android, para el desarrollo de nuestra aplicación se utilizarán las siguientes herramientas:

- Java Development Kit (JDK) 7.0 o superior.
- Eclipse (Eclipse IDE for Java Developers).
- Android SDK (Software Development Kit de Google)
- Eclipse Plug-in (Android Development Toolkit - ADT)

En los siguientes subepígrafes se dará una descripción de las herramientas SDK y ADT.

2.1.1 Software Development Kit (SDK)

El SDK de Android es una herramienta de desarrollo distribuida por Google, la cual contiene todo las herramientas y APIs necesarias para desarrollar imponentes y poderosas aplicaciones. Brinda documentación, ejemplos de código, tutoriales, y una plataforma para ejecutar las aplicaciones en desarrollo y verificar su funcionamiento. En la Figura 2-1 se muestran de manera general los elementos contenidos en el SDK de Android.

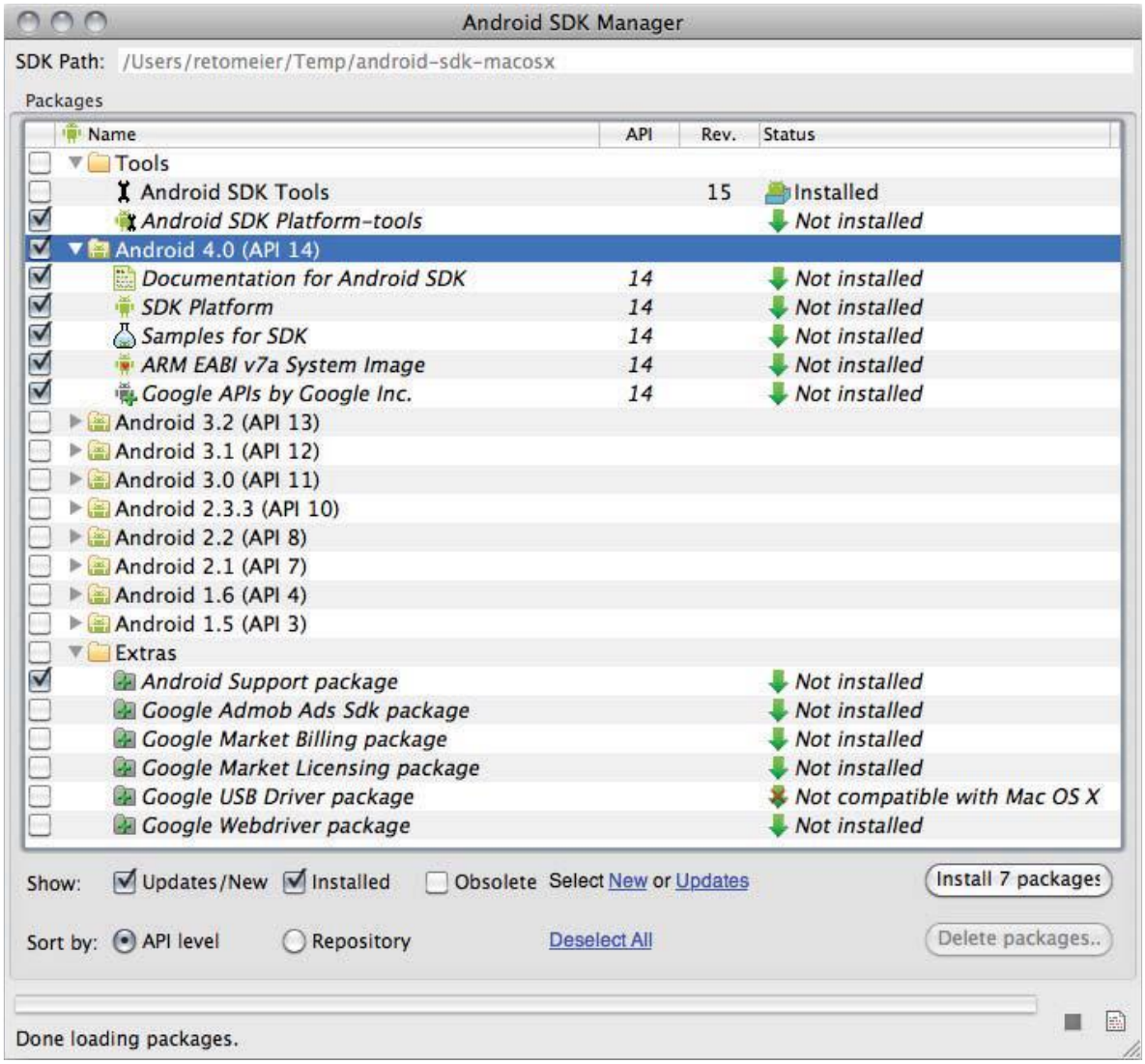

Figura 2- 1 Android SDK Manager

Set de herramientas del SDK de Android

El SDK de Android incluye una serie de herramientas y utilidades que ayudan al desarrollador a crear, probar y debuguear sus aplicaciones. A continuación se mencionan las principales herramientas y utilidades que proporciona (Meier, 2012):

- **AVD (Android Virtual Device) Manager y SDK Manager**– Usado para crear y administrar AVDs, y para descargar paquetes SDK, respectivamente. Un AVD realiza la función de host de un Emulador que corre una determinada versión liberada de Android, permitiendo especificar la versión SDK soportada, resolución de pantalla, la capacidad de almacenamiento permitida de la tarjeta SD, y capacidades de hardware permitidas (touchscreens, GPS, cámara, entre otros).

- **Android Emulator** – Constituye una implementación de la MV Dalvik, diseñado para que se ejecute dentro de un AVD para ejecutar aplicaciones como si fuera un dispositivo físico real.
- **Dalvik Debug Minitoring Service (DDMS)** – Usado para monitorear y controlar los Emuladores en los cuales se están debugueando las aplicaciones.
- **Logcat** – Una utilidad usada para mostrar y filtrar las salidas (outputs) del sistema.
- **Android Asset Packaging Tool (AAPT)** – Construye el paquete de archivos de distribución de Android (.apk).

Las siguientes herramientas adicionales están también disponibles (Meier, 2012):

- **SQLite3** – Una herramienta de base de datos que puede ser usada para acceder a los archivos creados y usados por Android provenientes de una base de datos SQLite.
- **Traceview** y **dmtracedump** – Herramientas de análisis gráfico para observar la traza de logs de una aplicación Android.
- **MkSDCard** – Crea una imagen en disco de una tarjeta SD, que puede ser usada por el Emulador para simular una tarjeta SD externa.
- **Dx** – Convierte los archivos byte code de Java (.class) en archivos byte code de Android (.dex).
- **Hierarchy Viewer** – Provee a la vez una representación visual de los layouts (pertenecientes a la jerarquía de la clase View) para debuguear y optimizar la interfaz de usuario, y una visualización aumentada para obtener pixaleado perfecto de los layouts.
- **Lint** – Una herramienta que analiza la aplicación y sus recursos para sugerir mejoras y optimizaciones.
- **Monkey** and **Monkey Runner –** Monkey corre con la MV, generando eventos seudo – aleatorios del usuario y del sistema. Monkey Runner provee una API para la escritura de programas para el control de la MV desde el exterior de una aplicación.
- **ProGuard –** Una herramienta para reducir y ofuscar el código de la aplicación, mediante el reemplazo de los nombres de las clases, variables y métodos con nombres alternativos semánticamente sin sentido alguno. Lo cual resulta de utilidad para hacer más difícil la aplicación de ingeniería inversa.

Cabe mencionar que muchas de las herramientas y utilidades anteriormente mencionadas son integradas al Eclipse por medio del plugin ADT, y pueden ser accedidas a través de la perspectiva DDMS del Eclipse.

2.1.2 Android Development Toolkit (ADT)

El plugin ADT para Eclipse simplifica el desarrollo de aplicaciones Android, debido a que integra diferentes herramientas de desarrollo, tales como, el Emulador y el convertidor de archivos .class a archivos .dex, directamente dentro del IDE. Aunque el uso del plugin ADT no es obligatorio, el uso de este permite la creación, prueba y debugueo de las aplicaciones de una manera rápida y sencilla.

A continuación se listan las integraciones que hace el plugin ADT en el Eclipse (Meier, 2012, Lee, 2012):

- Un asistente de proyectos (Android Project Wizard), que simplifica la creación de nuevos proyectos e incluye una plantilla de aplicación básica.
- Editores para ayudar a crear, editar y validar el manifiesto, layouts y demás recursos XML de la aplicación.
- Construcción automática de proyectos Android, conversión a ejecutables Android (.dex), embalaje a paquete de archivos (.apk) mediante el AAPT (Android Asset Packaging Tool), e instalación de paquetes en las MVs Dalvik (ya sea en el Emulador o en dispositivos físicos).
- AVD Manager, el cual permite crear y manejar dispositivos virtuales.
- Un emulador (Android Emulator), y la habilidad de controlar la apariencia del Emulador, las configuraciones de conexión de red, y la habilidad de simulación de llamadas entrantes y mensajes SMS.
- El DDMS, el cual incluye detalles de procesos, facilidades de captura de pantalla, entre otros aspectos.
- Acceso al sistema de archivos del dispositivo físico o del Emulador, permitiendo la navegación por el árbol de carpetas y la transferencia de archivos.
- Debugueo en tiempo de ejecución.
- Todos los logs generados por Android y la MV Dalvik, así como salidas de consola.

2.2 Layouts en Android

En Android los ***layouts*** son extensiones de la clase ***ViewGroup*** que extiende a la vez de la clase ***View***, y son usados para ubicar en una posición determinada de la interfaz de usuario de una ***actividad*** dada, diferentes ***vistas***. Cabe destacar que los ***layouts*** pueden ser anidados, es decir, pueden contener otros ***layouts***, permitiendo la creación de complejas interfaces de usuario (véase la Figura 2-2).

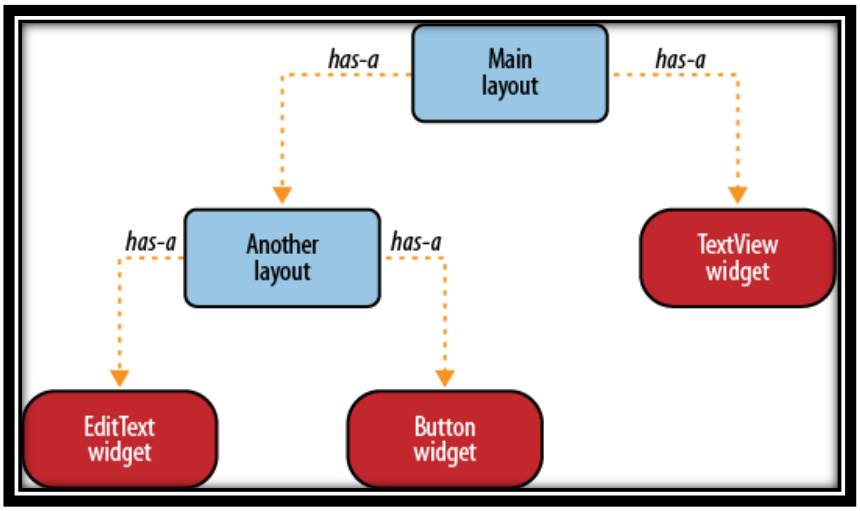

Figura 2- 2 Relación Layout - View

El SDK de Android incluye un conjunto de diferentes tipos de ***layouts***. Es decisión del desarrollador seleccionar y usar la combinación correcta para conformar una interfaz de usuario agradable, de fácil uso, y eficiente a la hora mostrar en pantalla.

La siguiente lista muestra algunos de los ***layouts*** más comúnmente utilizados (Tomás Gironés, 2011b, Gironés, 2012, Meier, 2012, Lee, 2012):

- **LinearLayout –** Posiciona los elementos en una fila o columna.
- **TableLayout –** Posiciona los elementos (comúnmente vistas) de forma de tabular.
- **FrameLayout –** Posiciona cada elemento uno encima de otro, de manera que un nuevo elemento añadido cubre al anterior, como un paquete de cartas.
- **RelativeLayout –** Posiciona los elementos en relación a otro o al padre (layout que contiene la vista).
- **AbsoluteLayout –** Posiciona los elementos de forma absoluta, es decir, en coordenadas específicas de la pantalla.

2.2.2 TableLayout - TableRow

Anteriormente se mencionó que un TableLayout permitía posicionar un conjunto de elementos en forma de tabla. Sin embargo, un TableLayout realmente no agrupa a ningún conjunto de elementos, sino un conjunto de TableRows. Un TableRow representa una fila de la tabla, y es quien realmente permite contener a otros elementos. Los elementos dentro de un TableRow están posicionados uno al lado de otro de manera horizontal, de igual forma que en un LinarLayout con orientación horizontal. La cantidad de columnas de un TableRow está dada por la cantidad de elementos contenidos en dicho TableRow (Lee, 2012, Gargenta, 2011, Developers, 2010).

En la siguiente figura se muestra un TableLayout definido mediante XML, el cual contiene varias vistas definidas también en XML.

```
<TableLayout xmlns:android="http://...
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
  <TableRow>
   <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
   <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
  </TableRow>
  <TableRow>
   <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"/>
   <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"/>
  </TableRow>
</TableLayout>
```

Figura 2- 3 TableLayout definido en XML

2.2.3 Usando TableLayout en la aplicación SIMIO

Para lograr el trabajo con tablas en la aplicación móvil SIMIO se hizo uso de TableLayout. Cada TableLayout de la aplicación fue definido mediante XML, sin embargo, debido a que las tablas debían ser dinámicas, el contenido de cada tabla (cantidad de TableRows incluidos y conjunto de elementos contenidos en cada TableRow) fue definido mediante código Java, y es generado en el momento de ejecución del método onCreate () de la actividad en cuya interfaz de usuario fue definido un TableLayout.

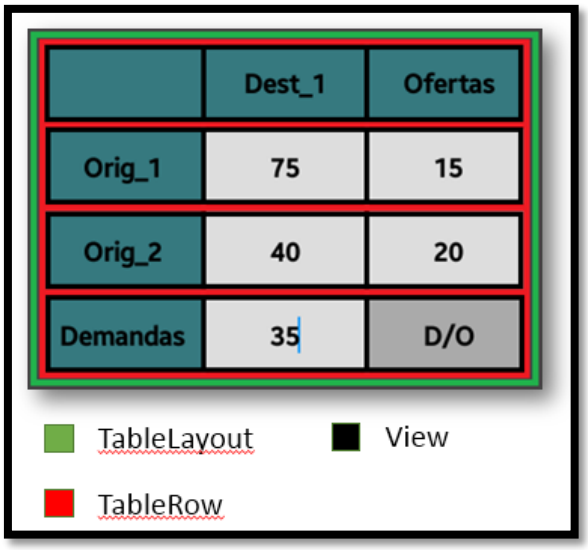

Figura 2- 4 Tabla de entrada del método "Transporte"

2.3 Gráficos en Android

Android nos proporciona, a través de su API gráfica, una potente y variada colección de funciones que pueden cubrir prácticamente cualquier necesidad gráfica de una aplicación, pudiéndose destacar la manipulación de imágenes, gráficos vectoriales, animaciones, trabajo con texto o gráficos 3D.

En el capítulo anterior se hizo mención y se describió cuáles eran los principales bloques de construcción en Android utilizados en el desarrollo de cualquier aplicación móvil. Dentro de estos bloques, se encuentran las ***vistas (views)***, las cuales, como bien se dijo, son los elementos que conforman la interfaz de usuario de una aplicación. Android provee una gran variedad de ***vistas*** por defecto (campos texto, radio buttons, check boxs, spinners, botones, entre otros) que cubren una gran variedad de necesidades. Sin embargo, ¿qué pasa cuando ninguna de las vistas disponibles ofrece lo que se quiere?

Una solución sería crear vistas personalizadas. Para esto, Android pone provee a los desarrolladores de las más poderosas bibliotecas gráficas disponibles para el trabajo con gráficos en el desarrollo de aplicaciones móviles. Actualmente, ofrece dos: una para gráficos 2D y otra para gráficos 3D.

Aunque el tema de gráficos en Android es un terreno bastante amplio, los siguientes subepígrafes se centrarán solo en el paquete Java ***android.graphics*** contenido en Android, el cual contiene todas las clases necesarias para la creación y manipulación de gráficos 2D.

2.3.1 Canvas

La clase ***Canvas*** representa una superficie donde podemos dibujar. La clase dispone de una serie de métodos que nos permiten dibujar líneas, círculos, óvalos, imágenes, entre otras formas y trazos.

En Android toda vista hereda de la clase View el método onDraw(), el cual tiene como parámetro un objeto ***Canvas*** asociado a dicha vista. La oportunidad de dibujar en el ***Canvas*** y así dibujar en la vista, se da sobrescribiendo este método.

Ya conocemos que en Android, la pantalla es tomada cuando ocurre el levantamiento de una actividad perteneciente a una aplicación determinada. La actividad, dentro de su método onCreate(), define mediante el método setContentView() lo que va a mostrar (puede ser una vista o un layout) por pantalla una vez que se ejecute onCreate().

La siguiente figura muestra una actividad llamada “Graphics”, la cual contiene una vista llamada “GraphicsView”.

```
public class Graphics extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(new GraphicsView(this));
   }

   static public class GraphicsView extends View {
      public GraphicsView(Context context) {
         super(context);
      }
      @Override
      protected void onDraw(Canvas canvas) {
         // Drawing commands go here
      }
   }
```

Figura 2- 5 Actividad Graphics y definición de la vista GraphicsView

Métodos de la clase Canvas

Al principio del epígrafe se mencionó que la clase ***Canvas*** contiene un conjunto de métodos para poder dibujar diferentes trazos y figuras. A continuación se listan los principales métodos de la clase (Gironés, 2012, Meier, 2012):

- Para dibujar figuras geométricas:
  - drawCircle (float cx, float cy, float radio, Paint pincel)
  - drawOval (RectF ovalo, Paint pincel)
  - drawRect (RectF rect, Paint pincel)
  - drawPoint (float x, float y, Paint pincel)
  - drawPoints (float[] pts, Paint pincel)
- Para dibujar líneas y arcos:
  - drawLine (float iniX, float iniY, float finX, float finY, Paint pincel)
  - drawLines (float[] puntos, Paint pincel)
  - drawArc (RectF ovalo, float iniAngulo, float angu1o, bcolean usarCentro, Paint pincel)
  - drawPath (Path trazo, Paint pincel)
- Para dibujar texto:
  - drawText (String texto, float x, float y, Paint pincel)
  - drawTextOnPath (String texto, Path trazo, float desplazamHor, float desplazamVert, Paint pincel)
  - drawPosText (String texto, float[] posicion, Paint pincel)
- Rellenar todo el ***Canvas***:
  - drawColor (int color)
  - drawARGB (int alfa, int rojo, int verde, int azul)
  - drawPaint (Paint pincel)
- Para averiguar el tamaño del ***Canvas***:
  - int getHeight ()
  - int getWidth ()

2.3.2 Paint

Una de las clases más importantes dentro de las bibliotecas gráficas nativas de Android, es la clase ***Paint***. Esta clase contiene toda la información necesaria, dígase transparencia, color, grosor del trazo, entre otras informaciones, para dibujar cualquier gráfico como texto o formas geométricas.

En el epígrafe anterior pudimos darnos cuenta de que todos los métodos mencionados utilizan un objeto ***Paint***.

Algunos de los métodos más importantes de la clase son (Tomás Gironés, 2011a, Gironés, 2012, Meier, 2012):

- Para definir el color del pincel (objeto ***Paint***)
    - setColor (int color) – permite indicar el color del pincel. El color puede ser obtenido de varias formas: mediante la clase Color la cual tiene definidos diferentes valores (ej: Color.BLUE); mediante el método argb de la clase Color (ej: Color.argb (int transparencia, int rojo, int verde, int azul)); mediante una constante de entero en formato hexadecimal (ej: 0x7F00FF00); y mediante el método getResources ().getColor () (ej: getResources().getColor(R.color.color_Circulo)).

      (color_Circulo ha de estar definido en res/values/colors.xml).

    - setAlpha (int alpha) – permite cambiar solo el grado de transparencia.
- Para definir el tipo de trazado
    - setStrokeWidth (int grosor) – define el grosor del trazado.
    - setStyle (Paint.Style estilo) – las primitivas gráficas son interpretadas por los valores: FILL, FILL_AND_STROKE, STROKE.
    - setShadowLayer (float radio, float dx, float dy, int color) – realiza un segundo trazado a modo de sombra.
- Para definir el tipo de texto
    - setTextAling (Paint.Aling justif) – define el tipo de justificación. Existen tres valores posibles: CENTER, LEFT, RIGHT.
    - setTextSize (float tamaño) – define el tamaño del texto.

        - setTypeface (Typeface fuente) – define el tipo de fuente: Existen por defecto los siguientes valores: MONOSPACE, SANS_SERIF, SERIF. Además desde la clase Typeface se puede definir las opciones negrita / itálica.
        - setTextScaleX (float escalaX) – define el factor de escalado horizontal. Por defecto 1.0.
        - setTextSkewX (float inclinacionX) – define el factor de inclinación. Por defecto 0.
        - setUnderlineText (boolean subrayado) – texto subrayado.
- Para mejorar la calidad del pincel con Anti-Aliasing
    - setSubpixelText (boolean decisión) – aplica la cualidad de alisado a nivel de subpíxel.
    - setAntiAlias (boolean decisión) – asegura que las líneas diagonales dibujadas por el pincel tengan la cualidad de alisado. Como costo, el rendimiento se ve afectado.

2.3.3 Creación de una vista en un fichero independiente

Como se pudo ver en la Figura 2-5, para poder dibujar en Android se tuvo que crear una clase descendiente de la clase ***View***. Esta clase era creada dentro de una actividad, por lo que solo podía utilizarse dentro de la misma. Sin embargo resulta mucho más interesante crear una vista de forma independiente. De esta forma, podremos utilizarla desde cualquier parte del proyecto que la incluye, o desde otros proyectos. Incluso estará visible en el editor de layouts del Eclipse.

A continuación se mencionan aspectos que se deben tener en cuenta a la hora de crear nuevas vistas (Gironés, 2012):

- La clase que representará la vista debe de extender de la clase ***View***.
- Se tiene que escribir un constructor de la clase, como mínimo con dos parámetros: un parámetro tipo ***Context*** que permitirá acceder al contexto de la aplicación, por ejemplo, para utilizar los recursos de la aplicación; y el segundo, de tipo ***AttributeSet***, nos permitirá acceder a los atributos de la vista cuando esta sea creada desde un layout.
- Como mínimo se deben de sobrescribir los métodos onDraw () y onSizeChange ().

La Figura 2-5 nos muestra la creación de una vista en un fichero independiente para que pueda ser utilizada desde cualquier parte y el esquema que debe seguir.

```
public class MiVista extends View {
    public MiVista(Context context, AttributeSet attrs) {
        super(context, attrs);
        //Inicializa la vista
        //Ojo: Aún no se conocen sus dimensiones
    }

    @Override protected void onSizeChanged(int ancho, int alto,
           int ancho_anterior, int alto_anterior){
        //Te informan del ancho y el alto
    }
    @Override protected void onDraw(Canvas canvas) {
        //Dibuja aquí la vista
    }
}
```

Figura 2- 6 Creación de una vista desde un fichero independiente

Realmente el lugar indicado para crear todos los componentes de la vista es en su constructor. Sin embargo se debe de tener cuidado, pues en este punto aún se desconocen las dimensiones que tendrá la vista.

Método onSizeChange ()

Android realiza un proceso de varias pasadas para determinar el ancho y alto de cada vista dentro de un layout. Cuando finalmente ha establecido las dimensiones de una vista llama a su método ***onSizeChange ()***. Este método nos indicará como parámetros, el ancho y alto asignado. En caso de tratarse de un reajuste de tamaños, el método pasará el ancho y alto anterior. La primera vez que se ejecuta el método estos valores son cero.

2.3.4 Gráficos 2D en la aplicación SIMIO

En la aplicación móvil SIMIO se realizó un trabajo con gráficos 2D para lograr la representación gráfica de grafos. Para esto, se crearon dos vistas en ficheros independientes: *"VistaCaptacion"* y *"VistaGrafo"*, para ser usadas desde layouts, y así aprovechar las ventajas brindadas por el editor de layouts en el Eclipse.

Ambas vistas hacen uso de las clases ***Canvas*** y ***Paint*** para el trabajo gráfico. También se aplicó la cualidad ***anti-aliasing*** para el mejoramiento de la calidad gráfica.

Figura 2- 7 Vista " VistaCaptacion"

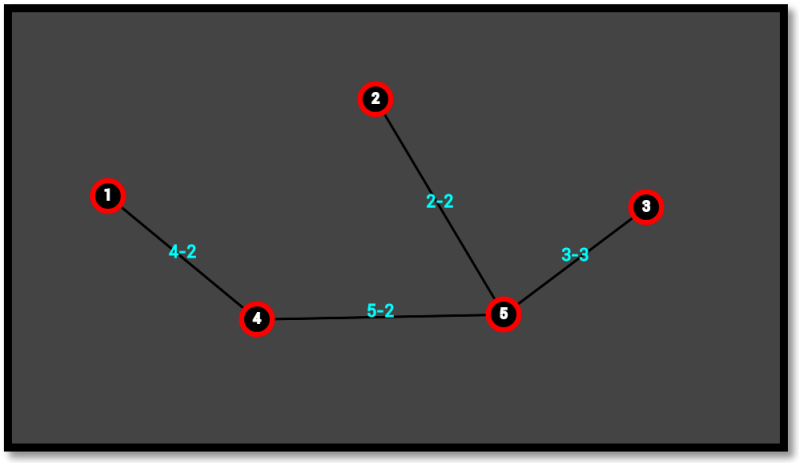

Figura 2- 8 Vista "VistaGrafo" generando el "Árbol de recubrimiento mínimo"

2.4 Patrones de Diseño

Los patrones de diseño se han vuelto una forma popular y complementaria para describir y desarrollar diseños de software. Un patrón de diseño puede ser caracterizado como un modelo de trabajo, el cual expresa una relación entre un cierto *contexto*, un *problema*, y una *solución*. Para el diseño de software, el contexto permite entender el entorno en el que el problema reside, y que solución puede ser apropiada dentro de ese ambiente. Un conjunto de requerimientos, incluyendo limitaciones y restricciones, actúa con un *sistema de fuerzas* que

influencia como el problema puede ser interpretado dentro de su contexto y como la solución puede ser efectivamente aplicada.

Aunque no existe una definición general de ***patrón de diseño***, podemos decir que un patrón es una ***solución*** a un ***problema*** en un ***contexto*** determinado.

- El ***contexto*** es la situación donde el patrón se aplica, debe de ser una situación recurrente.
- El ***problema*** se refiere al objetivo que se está tratando de lograr en el contexto y a las restricciones que existen en ese contexto.
- La ***solución*** es un diseño general que cualquiera puede aplicar y que resuelve el problema teniendo en cuenta sus restricciones.

2.4.1 Clasificación de los patrones de diseño

Tal y como, el número de patrones de diseño descubiertos crece, toma sentido la partición de estos en distintas clasificaciones con el objetivo de organizarlos. El esquema más conocido, realiza una partición en tres tipos de categorías, basadas en sus propósitos (Freeman et al., 2004):

- **Patrones creacionales:** Están relacionados con la instanciación de objetos y todos brindan una manera de desacoplar al cliente del objeto que necesita instanciar.
- **Patrones de comportamiento:** Están relacionados con la manera en que las clases y objetos interactúan y distribuyen responsabilidades.
- **Patrones estructurales:** Permiten componer clases u objetos en estructuras más grandes.

En la siguiente figura se muestran las tres categorías de patrones de diseño, con algunos de los patrones asociados a estas categorías.

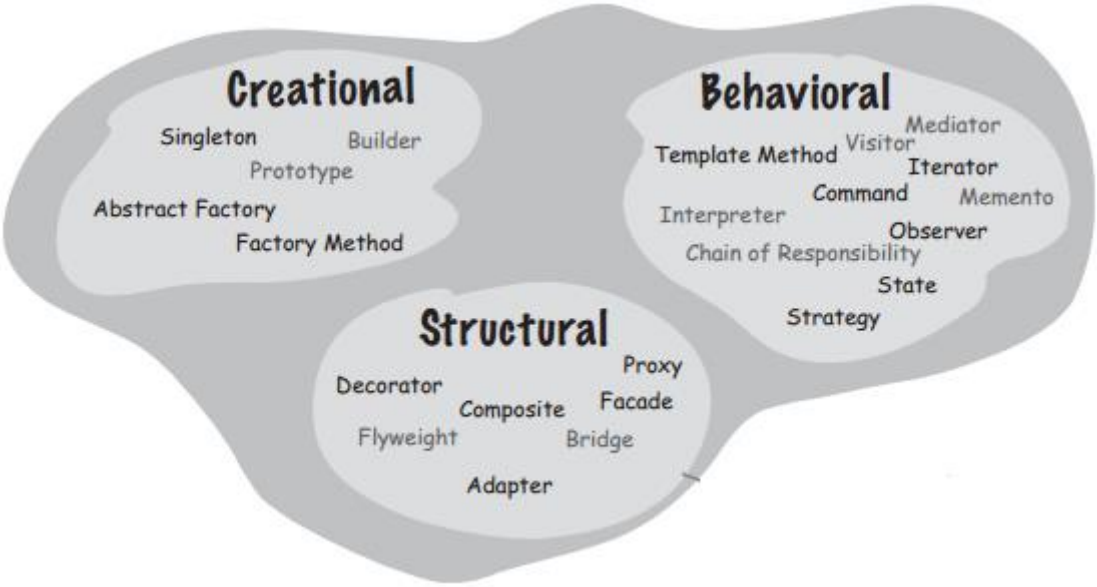

Figura 2- 9 Categorías de patrones de diseño

2.4.2 Patrón Strategy

El patrón ***Strategy*** consiste en un conjunto de algoritmos encapsulados en un contexto determinado (*Context*). El cliente puede elegir el algoritmo que prefiera de entre los disponibles o puede ser el mismo objeto *Context* el que elija el más apropiado para cada situación.

De forma general podemos definir al patrón Strategy de la siguiente manera: Define una familia de algoritmos, los encapsula y los hace intercambiables. Permite que el algoritmo varíe independientemente del cliente que lo utiliza (Freeman et al., 2004).

A continuación se listan las clases y/o objetos que participan en este patrón y además se muestra el diagrama de clases asociado al patrón Strategy en la Figura 2-6.

- **Strategy -** Declara una interface común a todos los algoritmos soportados. **Context** usa esta interface para llamar el algoritmo definido por ConcreteStrategy.
- **ConcreteStrategy -** Implementa el algoritmo usando la interfaz definida en Strategy.
- **Context -** Mantiene una referencia a un objeto Strategy.

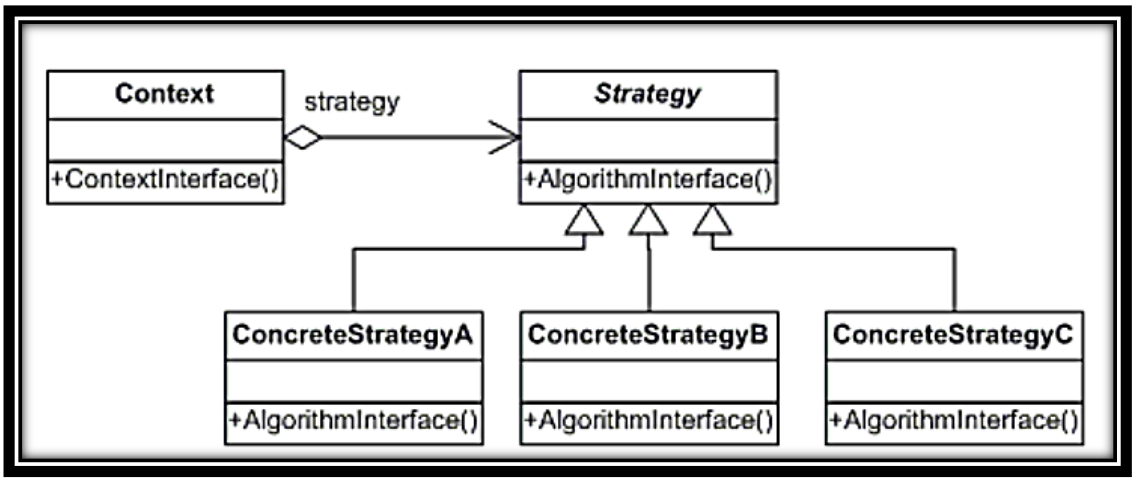

Figura 2- 10 Diagrama de clases del patrón Strategy

2.4.3 Patrón Factory Method

Este tipo de patrón se usa bastante debido a su utilidad. Su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente todos estos objetos provienen de una misma clase padre mientras que se diferencian entre ellos por algún aspecto de comportamiento.

De forma general podemos definir el patrón ***Factory Method*** de la siguiente manera: Especifica una interfaz para crear un objeto, pero deja a las subclases decidir qué clase instanciar. El ***Factory Method*** permite a una clase diferir la instanciación a las subclases (Freeman et al., 2004).

Como todo *Factory*, el patrón ***Factory Method*** nos da una manera de encapsular la instanciación de tipos concretos. Mirando la Figura 2-7, podemos observar el diagrama de clases asociado al patrón ***Factory Method***.

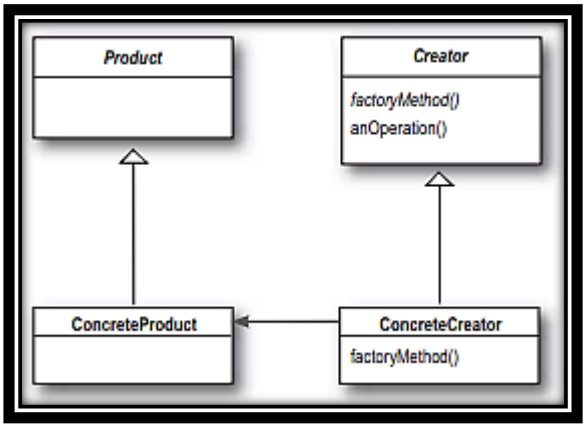

Figura 2- 11 Diagrama de clase del patrón Factory Method

A continuación se listan aspectos importantes a tener en cuenta:

- Todos los productos tienen que implementar la misma interfaz (Product) de modo que las clases que usen los productos se refieran a la interface, no a una clase concreta.
- La clase ConcreteCreator es responsable por la creación de uno o más ConcreteProducts. Es la única clase que sabe cómo crear un ConcreteProduct.
- La clase ConcreteCreator implementa el factoryMethod(), que es el método que realmente produce productos.
- La clase Creator implementa un factoryMethod() abstracto, método que todas las subclases deben implementar.
- La clase Creator contiene la implementación de todos los métodos que manipulan productos, exceptuando el método factoryMethod().
- Usualmente los desarrolladores dicen que el patrón factoryMethod() permite que las subclases decidan que clase instanciar. Quien realmente decide que clase instanciar es el desarrollador, al instanciar al ConcreteCreator que necesite.

2.4.4 Patrón Modelo-Vista-Controlador (MVC)

El patrón de diseño MVC es uno de los patrones existentes más citados. Desde su creación ha jugado un papel influyente en la mayoría de los frameworks de interfaz de usuario y en la manera de pensar acerca del diseño de la interfaz de usuario.

Este patrón define tres componentes principales: el *Modelo*, la *Vista* y el *Controlador*. A continuación se describen brevemente la funcionalidad de cada componente (Sommerville, 2009, Fowler, 2002):

- **EL Modelo**: Maneja el sistema de datos y las operaciones asociadas a dichos datos.
- **La Vista**: Representa la visualización del Modelo. Define y maneja como los datos serán mostrados al usuario.
- **El Controlador**: Maneja las interacciones del usuario, manipula el *Modelo* y causa la actualización de la *Vista*.

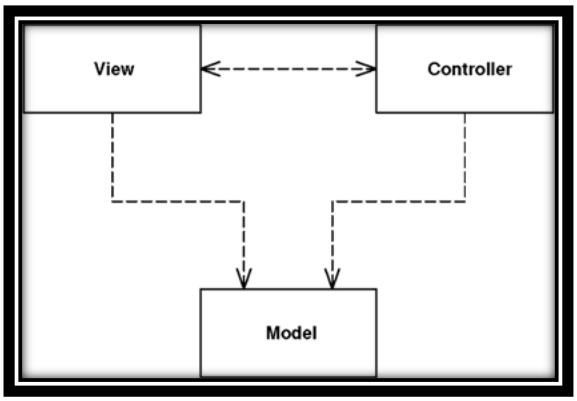

Figura 2- 12 Diagrama clásico del MVC

MVC en Android

El *Framework de Interfaz de Usuario de Android* está organizado alrededor del patrón MVC ilustrado en la Figura 2-9. Este framework provee la estructura y herramientas para construir un *Controlador* que trate las entradas del usuario (ej. toques de pantalla) y una *Vista* que muestre de forma gráfica información en la pantalla (Gargenta, 2011).

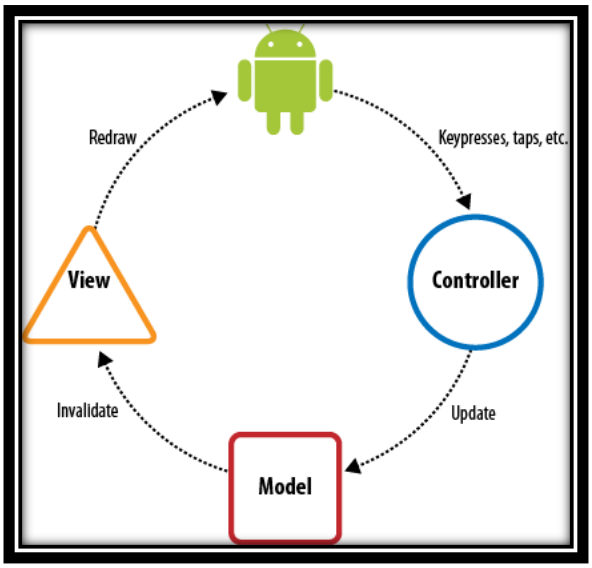

Figura 2- 13 Patrón Modelo-Vista-Controlador en Android

Métodos para el manejo de eventos en Android

En Android normalmente el *Controlador* estará reflejado mediante una *actividad*. Para el manejo de los diferentes eventos lanzados por las entradas realizadas por el usuario, Android provee la interfaz *"EventListener"* de la clase *"View"*, dicha interfaz posee un método *callback*, que será llamado por Android cuando se produzca una acción determinada.

Se tienen a los siguientes interfaces *"EventListener"* con sus respectivos métodos *callback* (Gironés, 2012):

- **onClick () –** Método de la interfaz *View.OnClickListener*. Se llama cuando el usuario selecciona un elemento. Se puede utilizar cualquier medio como la pantalla táctil o las teclas de navegación.
- **onLongClick () -** Método de la interfaz *View.OnLongClickListener*. Se llama cuando el usuario selecciona un elemento durante más de un segundo.
- **onFocusChange ()** - Método de la interfaz *View.onFocusChangeListener*. Se llama cuando el usuario navega dentro o fuera de un elemento.
- **onKey ()** - Método de la interfaz *View.onKeyListener*. Se llama cuando se pulsa o se suelta una tecla del dispositivo.
- **onTouch ()** - Método de la interfaz *View.onTouchListener*. Se llama cuando se realiza una interacción con la pantalla táctil.
- **onCreateContextMenu () -** Método de la interfaz *View.OnCreateContextMenuListener*. Se llama cuando se crea un menú de contexto.

Método invalidate ()

Ante la necesidad de refrescar la visualización de una *vista* en pantalla, normalmente debido a un cambio del *Modelo*, Android brinda el método ***invalidate ()***. Dicho método hará que una *vista* sea actualizada invocando nuevamente su método ***onDraw ()***.

2.4.5 Patrones de diseño en la aplicación SIMIO

En la aplicación móvil SIMIO se hizo uso de tres patrones de diseño, cada uno de ellos descritos anteriormente, estos son: Strategy, Factory Method, y MVC. A continuación se expondrán los principales motivos por los que fue necesario la utilización de los patrones anteriormente mencionados.

Strategy

- Existencia de diferentes estrategias de construcción de tablas (tablas de entrada y salida), tales como:
  - ✓ Construir tabla sin diagonal
  - ✓ Construir tabla con *n* filas y *m* columnas
  - ✓ Construir tabla triangular superior

- Necesidad de obtener los valores contenidos en las tablas de entrada generadas:
  - ✓ Obtener valores a partir de tablas sin diagonal
  - ✓ Obtener valores a partir de tablas con *n* filas y *m* columnas
  - ✓ Obtener valores a partir de tablas triangular superior

Factory Method

- Existencia de un total de quince tablas asociadas a los diferentes métodos de la aplicación.
- Necesidad de encapsular y centralizar la creación de objetos tipo ***Tabla***.

Modelo - Vista - Controlador (MVC)

- Necesidad de representar gráficamente el grafo "Árbol de recubrimiento mínimo" a partir de los datos generados por la ejecución del método "Árbol de recubrimiento mínimo".
- Posibilidad de mostrar paso a paso la generación del grafo "Árbol de recubrimiento mínimo" mediante acciones sobre la pantalla por parte del usuario.

2.5 Análisis y Diseño

Hasta este punto ya se han introducido todos los conocimientos básicos sobre SO Android y el desarrollo de aplicaciones para dicha plataforma. También se introdujo el ambiente de desarrollo en el que se va a trabajar, así como aspectos teóricos y prácticos que serán utilizados para el desarrollo de la aplicación móvil SIMIO.

En el presente epígrafe se comentará la fase de desarrollo del proyecto, abordando los aspectos de análisis y diseño de la aplicación.

2.5.1 Descripción del Negocio

El negocio se desarrolla en el Departamento de Inteligencia Artificial, perteneciente al CEI (Centro de Estudios Informático), centro el cual pertenece a la Universidad Central "Marta Abreu" de Las Villas.

Los profesores que imparten las asignaturas de Modelos de Optimización I y II e Investigación de Operaciones, en la facultad MFC (Matemática - Física - Computación), al cuarto año de la carrera de Ciencia de La Computación y a cuarto año de Ingeniería Informática, pertenecientes al Departamento de Inteligencia Artificial, pidieron extender la

aplicación desktop para Windows: SIMIO (Simulación de Métodos de Investigación de Operaciones), a aplicaciones móviles para Android, con el objetivo de apoyar la impartición de las asignaturas.

Los principales procesos de este negocio están orientados a la posibilidad de simular los distintos métodos de Investigación de Operaciones impartidos en estas asignaturas.

2.5.2 Requisitos de la aplicación

A continuación de especificarán los requisitos que deberá cumplir la aplicación. Dichos requisitos serán divididos en dos categorías: funcionales y no funcionales.

Requisitos funcionales

A continuación se definen las funcionalidades que el sistema será capaz de realizar:

- Ejecutar siete métodos de Investigación de Operaciones, tales como:
  - ✓ Simplex
  - ✓ Asignación
  - ✓ Transporte
  - ✓ Flujo máximo
  - ✓ Flujo de costo mínimo
  - ✓ Árbol de recubrimiento mínimo
  - ✓ Camino mínimo desde una misma fuente
- Visualizar en la pantalla del dispositivo, después de la ejecución de un método, el resultado de las iteraciones en forma de tabla. Para esto se contará con las siguientes acciones de movimiento:
  - ✓ Avanzar un paso
  - ✓ Retroceder un paso
  - ✓ Avanzar hasta el final
  - ✓ Retroceder hasta el principio
- Generar y mostrar por pantalla el grafo "árbol de recubrimiento mínimo", a partir de la ejecución del método "Árbol de recubrimiento mínimo".
- Visualizar en la pantalla del dispositivo, una vez ejecutado del método "Árbol de recubrimiento mínimo", el resultado de las iteraciones mediante el grafo "árbol de recubrimiento mínimo".

Requisitos no funcionales

- **Usabilidad**
  - ✓ La aplicación deberá visualizarse correctamente en cualquier dispositivo Android con las siguientes dimensiones de pantalla: pequeña, normal o grande.
  - ✓ La interfaz visual de la aplicación debe ser atractiva y sencilla, permitiendo al usuario facilidad de uso y entrenamiento.
  - ✓ La aplicación deberá ser soportada en la mayor cantidad posible de versiones del SO Android.
  - ✓ La aplicación deberá ser capaz de brindar información al usuario mediante mensajes, para ayudarlo y guiarlo durante su interacción con la aplicación.
- **Ayuda y documentación**
  - ✓ Se brindarán manuales de ayuda que documenten cómo trabajar de forma adecuada con el software.
- **Interfaz**
  - ✓ En la pantalla principal se tendrá acceso a todos los métodos con que contará la aplicación de manera sencilla y con un solo clic.
  - ✓ La interfaz de usuario de la aplicación deberá ser diseñada de forma tal que permita el aprovechamiento del espacio.
  - ✓ Los componentes Android de interfaz de usuario para la entrada de datos, de ser posible, deberán ser configurados de manera que el riesgo de entrada de datos inválidos por parte del usuario disminuya.

2.5.3 Actores y Casos de Uso del Sistema

A continuación se presenta el diagrama de Actores y Casos de Uso del Sistema, el cual describe las funcionalidades que brindará el sistema para cada usuario.

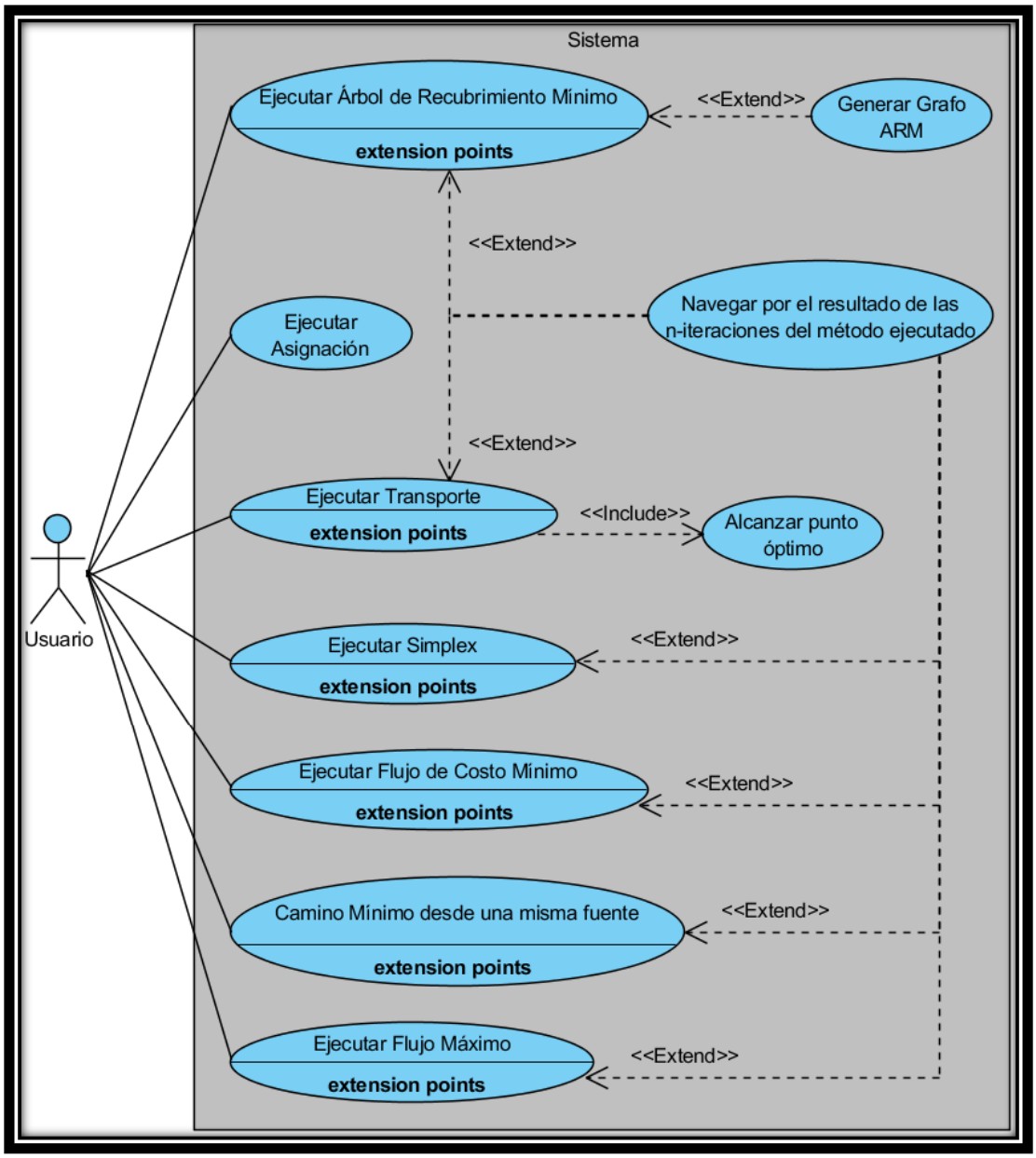

Figura 2- 14 Diagrama de Actores y Casos de Uso del Sistema

Descripción de los Casos de Uso del Sistema

Tabla 2- 1 Descripción del Caso de Uso "Ejecutar Simplex"

Nombre	Ejecutar Simplex
Descripción: Permite la ejecución del método “Simplex”, aplicado a un problema determinado.	

Actores	Usuario

Precondiciones:

- Debe estar visible la pantalla principal de la aplicación.

Flujo normal:

1. El usuario selecciona la opción “Simplex” en la pantalla principal.
2. El sistema mostrará una pantalla para la entrada de datos necesarios para la ejecución del método “Simplex”.
3. El usuario selecciona la opción “Generar tabla”.
4. Si los datos entrados son correctos el sistema mostrará una pantalla con una tabla (tabla de entrada) mediante la cual se entrarán los datos del problema a resolver.
5. El usuario selecciona la opción “Ejecutar método”.
6. Si los datos son correctos el sistema ejecutará el método “Simplex”.
7. Finaliza el flujo.

Flujo alternativo:

3A. El usuario da retroceso y se finaliza el flujo.

4A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 3.

5A. El usuario da retroceso y se vuelve al punto 2.

6A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 5.

Post-condiciones:

- Se tendrán almacenadas las soluciones y otras informaciones deseadas, correspondientes a cada iteración de la ejecución del método “Simplex”.
- Se mostrará una pantalla en la cual estará contenida una tabla (tabla de salida) mediante la cual se mostrarán los resultados de cada iteración de la ejecución del método. (Por defecto se mostrará los resultados de la iteración cero)

La anterior descripción es común a todos los siguientes Casos de Uso:

- Ejecutar Árbol de Recubrimiento Mínimo
- Ejecutar Flujo Máximo
- Ejecutar Flujo de Costo Mínimo
- Camino Mínimo desde una misma Fuente

Tabla 2- 2 Descripción de Caso de Uso "Ejecutar Asignación"

Nombre	Ejecutar Asignación
Descripción: Permite la ejecución del método “Asignación”, aplicado a un problema determinado.	
Actores	Usuario
Precondiciones: ▪ Debe estar visible la pantalla principal de la aplicación.	
Flujo normal: 1. El usuario selecciona la opción “Asignación” en la pantalla principal. 2. El sistema mostrará una pantalla para la entrada de los datos necesarios para la ejecución del método “Asignación”. 3. El usuario selecciona la opción “Generar tabla”. 4. Si los datos entrados son correctos el sistema mostrará una pantalla (tabla de entrada) con una tabla mediante la cual se entrarán los datos del problema a resolver. 5. El usuario selecciona la opción “Ejecutar método”. 6. Si los datos son correctos el sistema ejecutará el método “Asignación”. 7. Finaliza el flujo.	
Flujo alternativo: 3A. El usuario da retroceso y se finaliza el flujo. 4A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 3. 5A. El usuario da retroceso y se vuelve al punto 2.	

6A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 5.
Post-condiciones: ▪ Se mostrará una pantalla en la cual estará contenida una tabla (tabla de salida) mediante la cual se mostrará el resultado final de la ejecución del método.

Tabla 2- 3 Descripción del .Caso de Uso "Ejecutar Transporte"

Nombre	Ejecutar Transporte
Descripción: Permite la ejecución del método “Transporte”, aplicado a un problema determinado.	
Actores	Usuario
Precondiciones: Debe estar visible la pantalla principal de la aplicación.	
Flujo normal: 1. El usuario selecciona la opción “Transporte” en la pantalla principal. 2. El sistema mostrará una pantalla para la entrada de datos necesarios para la ejecución del método “Transporte”. 3. El usuario selecciona la opción “Generar tabla”. 4. Si los datos entrados son correctos el sistema mostrará una pantalla con una tabla mediante la cual se entrarán los datos del problema a resolver. 5. El usuario selecciona la opción “Ejecutar método”. 6. Si los datos son correctos el sistema ejecutará la primera fase del método “Transporte” (Cálculo de la solución básica factible). 7. El sistema mostrará una pantalla en la cual estará contenida una tabla (tabla de salida) mediante la cual se mostrarán los resultados de cada iteración de la ejecución de la primera fase del método. (Por defecto se mostrará los resultados de la iteración cero de la ejecución de la primera fase del método “Transporte”). 8. El usuario selecciona la opción “Ir al paso dos”.	

9. El sistema mostrará una pantalla en la cual estará contenida una tabla (tabla de salida) mediante la cual se irá mostrando el resultado de la ejecución de la segunda fase del método “Transporte” (Por defecto muestra la última iteración de la ejecución de la ejecución de la primera fase, junto con otras informaciones de interés). La pantalla contendrá en adición un conjunto de componentes visuales necesarios para la ejecución de la segunda fase.
10. El usuario determina que variable se hace cero.
11. El usurario selecciona la opción “Calcular valores”.
12. El sistema verifica condición de optimalidad.
13. Si la solución actual representa un punto óptimo, finaliza el flujo.

Flujo alternativo:

3A. El usuario da retroceso finaliza el flujo.

4A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 3.

5A. El usuario da retroceso y se vuelve al punto 2.

6A. Si los datos entrados son incorrectos el sistema informará al usuario. Volver al punto 5.

8A. Se realiza la acción "Navegar por el resultado de las n-iteraciones del método ejecutado". Al finalizar la acción se vuelve al punto 8.

8B. El usuario da retroceso y se vuelve al punto 4.

10A. El usuario no determina que variable se va a hacerse cero. El sistema escoge la opción por defecto. Se vuelve al punto 4.

10B. El usuario da retroceso y se volverá al punto 7

11A. El usuario da retroceso y se volverá al punto 7.

13A. Se realiza la acción “Alcanzar punto óptimo”. Al finalizar la acción se vuelve al punto 10.

Post-condiciones del punto 6:

▪ Se tendrán almacenadas las soluciones y otras informaciones deseadas, correspondientes a cada iteración de la ejecución de la primera fase de ejecución del método “Transporte”.

Tabla 2- 4 Descripción del Caso de Uso "Alcanzar punto óptimo"

Nombre	Alcanzar punto óptimo
Descripción: Verificará la condición de optimalidad a partir de la solución básica factible obtenida tras la ejecución de la primera fase del método “Transporte”, se irá mejorando dicha solución hasta no llegar a un punto óptimo.	
Actores	Usuario
Precondiciones: ▪ Tiene que haberse ejecutado la primera fase del método “Transporte” y haberse almacenado el resultado generado en la última iteración de la ejecución de la primera fase del método. ▪ Debe estar mostrada la pantalla en la cual estará contenida la tabla de salida, mediante la cual se mostrarán los resultados de la ejecución de la primera fase del método “Transporte”.	
Flujo normal: 1. El usuario buscará un ciclo válido en una tabla determinada. 2. EL sistema captará las posiciones de la tabla referentes a cada punto del ciclo. 3. Al encontrarse un ciclo válido, el usuario seleccionará la opción “Balancear”. 4. El sistema modificará la solución actual a partir de las posiciones obtenidas en el punto 2. 5. Finaliza el flujo.	

Tabla 2- 5 Descripción del Caso de Uso "Navegar por el resultado de las n-iteraciones del método ejecutado"

Nombre	Navegar por el resultado de las n-iteraciones del método ejecutado

Descripción:

Permite realizar un recorrido paso a paso e ir visualizando los resultados generados en cada una de las iteraciones de la ejecución de un método dado. Para esto utilizará las estructuras de datos que contienen almacenadas las soluciones de las iteraciones al finalizar la ejecución del método.

Actores	Usuario

Precondiciones:

- Tiene que haberse ejecutado previamente un método dado (excluyendo el método "Asignación") y haberse almacenado los resultados generados en las iteraciones de la ejecución.
- Debe estar visible la pantalla en la cual estará contenida la tabla de salida, mediante la cual se mostrarán los resultados de cada iteración de la ejecución del método.

Flujo normal:

1. El usuario tendrá la posibilidad de seleccionar la siguientes acciones:
    a. Si no se ha llegado a la última iteración, seleccionar opción "avanzar a la siguiente iteración".
    b. Si no se ha llegado a la iteración cero, seleccionar opción "retroceder a la anterior iteración".
    c. Si no se ha llegado a la última iteración, seleccionar opción "avanzar hasta la última iteración".
    d. Si no se ha llegado a la iteración cero, seleccionar opción "retroceder hasta la iteración cero".
2. Si se seleccionó la opción:
    a. Se mostrarán los resultados de la siguiente iteración en la tabla de salida.
    b. Se mostrarán los resultados de la anterior iteración en la tabla de salida.
    c. Se mostrarán los resultados de la última iteración en la tabla de salida.

d. Se mostrarán los resultados de la iteración cero en la tabla de salida. 3. Se volverá al punto 1.
Flujo alternativo: 1A. El usuario da retroceso y finaliza el flujo. 1B. El usuario selecciona la opción “Grafo ARM”.

Tabla 2- 6 Descripción del Caso de Uso “Generar Grafo ARM”

Nombre	Generar Grafo ARM
Descripción: Permitirá la generación y visualización del grafo “Árbol de recubrimiento mínimo”	
Actores	Usuario
Precondiciones: ▪ Tiene que haberse ejecutado el método “Árbol de recubrimiento mínimo” y haberse almacenado los resultados generados en las iteraciones de la ejecución del método. ▪ Debe estar mostrada la pantalla en la cual estará contenida la tabla de salida, mediante la cual se mostrarán los resultados de cada iteración de la ejecución del método.	
Flujo normal: 1. El usuario selecciona la opción “Grafo ARM”. 2. El sistema mostrará una pantalla para la asignación de las posiciones de los nodos del grafo por parte del usuario. 3. Si todas las posiciones han sido ya asignadas, el usuario seleccionará la opción “Visualizar grafo ARM”. 4. El sistema mostrará una pantalla con los nodos del grafo posicionados según la ubicación asignada por el usuario (iteración cero). 5. El usuario tendrá la posibilidad de seleccionar la siguientes acciones: a. Si no se ha llegado a la última iteración, seleccionar opción “avanzar a la siguiente iteración”.	

b. Si no se ha llegado a la iteración cero, seleccionar opción “retroceder a la anterior iteración”.
c. Si no se ha llegado a la última iteración, seleccionar opción “avanzar hasta la última iteración”.
d. Si no se ha llegado a la iteración cero, seleccionar opción “retroceder hasta la iteración cero”.

6. Si se seleccionó la opción:
   a. Se mostrarán los resultados de la siguiente iteración en el grafo.
   b. Se mostrarán los resultados de la anterior iteración en el grafo.
   c. Se mostrarán los resultados de la última iteración en el grafo.
   d. Se mostrarán los resultados de la iteración cero en el grafo.
7. Se volverá al punto 5.

Flujo alternativo:

3A. El usuario da retroceso y finaliza el flujo.

3B. El usuario selecciona la opción “Limpiar” y se vuelve al punto 3.

5A. El usuario da retroceso y se vuelve al punto 2.

2.5.4 Estructura principal de la aplicación SIMIO

La estructura principal del proyecto se centrará en las siguientes carpetas: la carpeta ***src***, y las carpetas ***layout***, ***menu*** y ***drawable***, contenidas en la carpeta ***res***. A continuación se muestra la relación entre las carpetas ***src*** y ***res***.

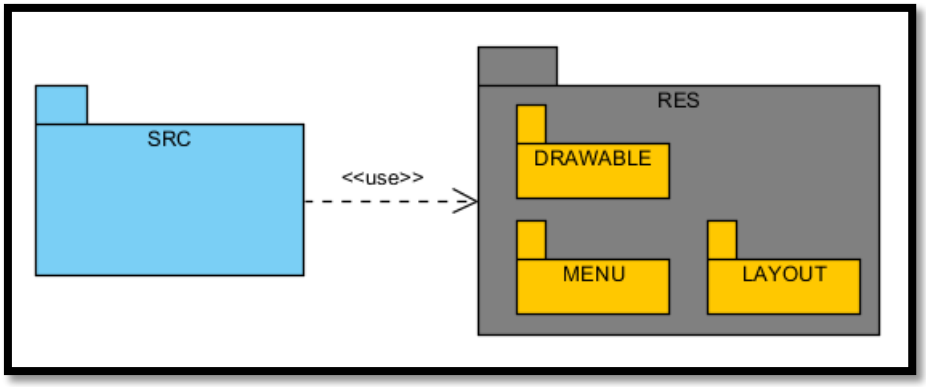

Figura 2- 15 Estructura principal de la aplicación SIMIO

Como puede verse, la carpeta ***src***, la cual contiene todo el código de la aplicación, hace uso de la carpeta ***res***, en la cual están contenidos todos los recursos de la aplicación.

2.5.5 Estructura de la carpeta “src”

La carpeta ***src***, como bien se ha mencionado anteriormente, es la que contiene todo el código de la aplicación. La aplicación SIMIO dentro de esta carpeta posee una estructura de cinco paquetes con los siguientes nombres: com.example.simio_app, graficos, métodos, tablas y útil. En la siguiente figura se muestra la relación entre los paquetes anteriormente mencionados.

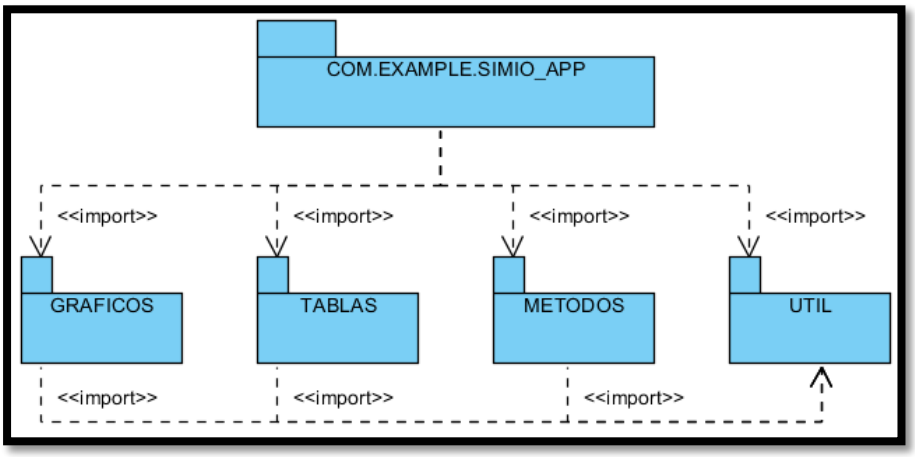

Figura 2- 16 Diagrama de paquetes de la carpeta "src"

2.5.5 Paquete “com.example.simio_app”

El paquete “com.example.simio_app”, se puede decir que es la base de la aplicación. En su interior se encuentran todas las actividades de la aplicación, cargando con toda la lógica de la interfaz de usuario, así como todo el comportamiento en general de la aplicación SIMIO. Este paquete posee un total de 25 actividades. Véase el siguiente diagrama de clases.

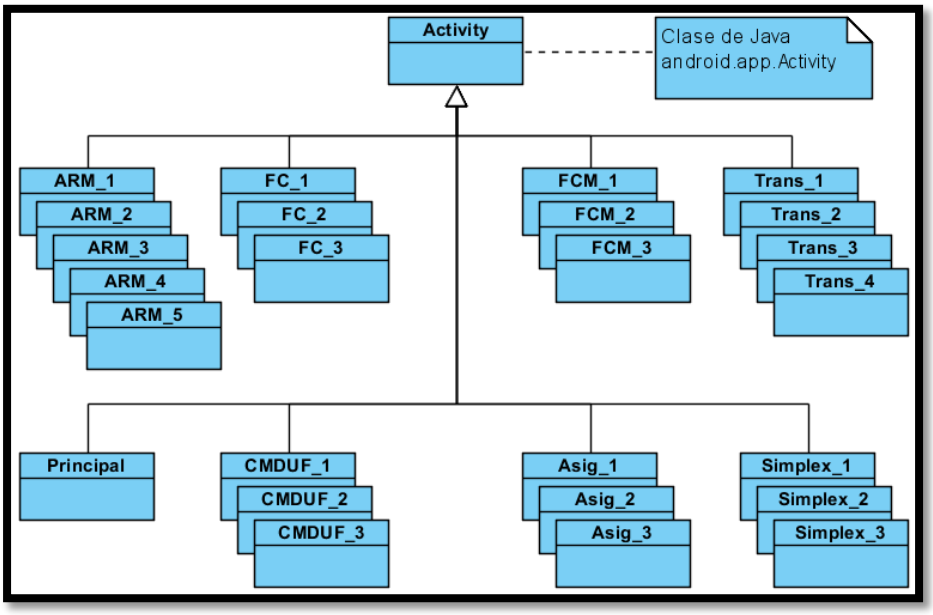

Figura 2- 17 Diagrama de clases del paquete "com.example.simio_app"

En la Figura 2-17, se puede observar que las actividades (exceptuando la actividad "Principal") están agrupadas. Este agrupamiento se realiza según el método al cual estas están asociadas:

- Actividades ARM_* - Actividades asociadas al método de optimización "Árbol de recubrimiento mínimo".
- Actividades FC_* - Actividades asociadas al método de optimización "Flujo máximo".
- Actividades FCM_* - Actividades asociadas al método de optimización "Flujo de costo mínimo".
- Actividades Simplex_* - Actividades asociadas al método de optimización "Simplex".
- Actividades Trans_* - Actividades asociadas al método de optimización "Transporte".
- Actividades Asig_* - Actividades asociadas al método de optimización "Asignación".
- Actividades CMDUF_* - Actividades asociadas al método de optimización "Camino mínimo desde una misma fuente".

Es necesario aclarar que cada una de las actividades del paquete "com.example.simio_app" hace uso de un layout y menú específico, contenidos en las carpetas ***layout*** y ***menú*** respectivamente.

Funcionalidad de las actividades contenidas en "com.example.simio_app"

A continuación se especificarán las funcionalidades, de manera general, de las actividades contenidas dentro del paquete "com.example.simio_app". Dichas funcionalidades son:

- **Actividades *_1**: Captarán los datos necesarios, entrados por el usuario mediante sus interfaces gráficas (IGs), para la creación de la "tabla de entrada" y la "tabla de salida" en cada una de las IGs de las actividades ***_2** y ***_3** respectivamente. Los datos captados serán necesarios también para la ejecución del método de optimización al cual cada una de ellas está asociada.
- **Actividades *_2**: Captarán los datos del problema de optimización a resolver, mediante una "tabla de entrada" contenida en cada una de sus IGs.
- **Actividades *_3**: Mostrarán los datos obtenidos, a partir de la ejecución del método de optimización al cual cada una de ellas está asociada. Los datos serán mostrados a través de una "tabla de salida" contenida en cada una de sus IGs, mediante las "*acciones de movimiento*" mencionadas en los requisitos funcionales del epígrafe 2.5.2. Vale aclarar que el método "Transporte" cuenta con dos fases de ejecución. La actividad **Trans_3** solo mostrará los datos obtenidos en la primera fase. Otra aclaración necesaria es que la actividad **Asig_3** no hace uso de las "*acciones de movimiento*".
- **Actividad ARM_4**: Captará datos necesarios para generar el grafo "Árbol de recubrimiento mínimo" mediante la vista "VistaCaptacion" contenida en su IG.
- **Actividad ARM_5**: Mostrará los datos obtenidos, a partir de la ejecución del método de optimización al cual dicha actividad está asociada. Los datos serán mostrados a través del grafo "Árbol de recubrimiento mínimo", contenido en su respectiva IG, mediante las "*acciones de movimiento*" mencionadas en los requisitos funcionales del epígrafe 2.5.2.
- **Actividad Trans_4**: Mostrará los datos que se irán obteniendo en la segunda fase de ejecución del método de optimización "Transporte". Los datos serán mostrados en una "tabla de salida" contenida en su IG. Vale aclarar que la segunda fase de ejecución

del método “Transporte” puede realizarse más de una vez. La IG de la actividad contiene una serie de componentes visuales para la captación de información necesaria para la realización de la segunda fase de ejecución.

2.5.6 Paquete “metodos”

El paquete “metodos” contiene las clases asociadas a los métodos de optimización implementados en la aplicación. Dichas clases son:

Tabla 2- 7 Clases del paquete “metodos” – Método de IO correspondiente

Nombre de la clase	Método de optimización
EdmonK (Edmonds Karp)	Flujo máximo
Kruscal	Árbol de recubrimiento mínimo
Dijkstra	Camino mínimo desde una misma fuente
Simplex	Simplex
Transporte	Transporte
Asignación	Asignación
FCM	Flujo de costo mínimo

A continuación se muestra el diagrama de clases del paquete “métodos”.

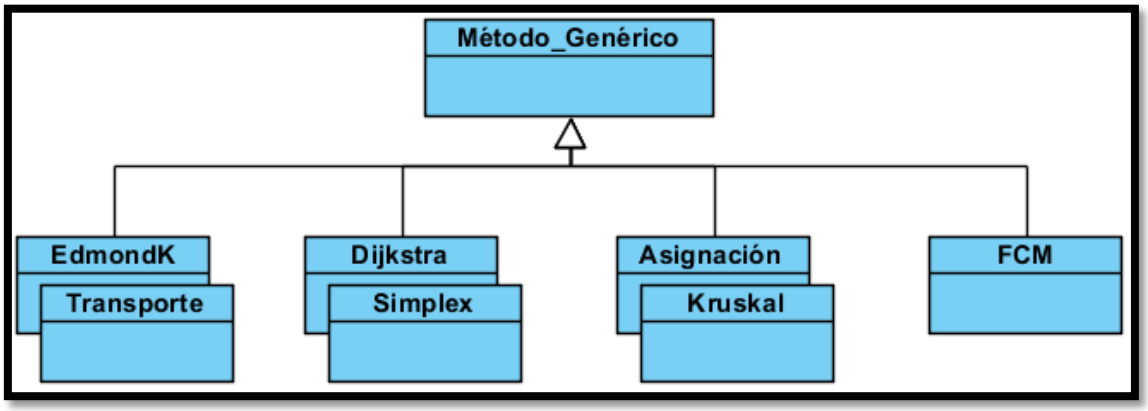

Figura 2- 18 Diagrama de clases del paquete "métodos"

2.5.6 Paquete “graficos”

El paquete “grafos” contiene dos clases que representan **vistas**. Dichas clases son: *“VistaCaptacion”* y *“VistaGrafo”*; las cuales permitirán la representación gráfica en pantalla, del grafo “Árbol de recubrimiento mínimo”.

En la siguiente figura se muestra el diagrama de clases de dicho paquete.

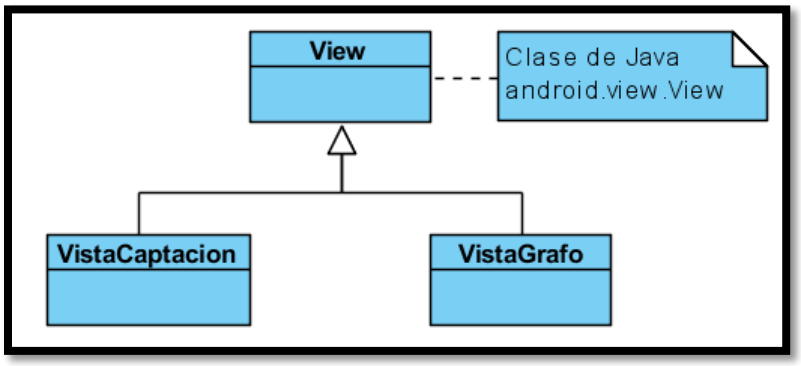

Figura 2- 19 Diagrama de clases del paquete "grafos"

Funcionalidad de las clases del paquete “metodos”

Seguidamente se especificarán las funcionalidades, de forma general, de las actividades contenidas dentro del paquete "metodos". Dichas funcionalidades son:

- **VistaCaptacion**: Permitirá que se visualice en pantalla, la posición asignada por el usuario a cada uno de los nodos perteneciente al grafo “Árbol de recubrimiento mínimo”. También permitirá guiar al usuario a la hora de asignar las posiciones de cada nodo ante la existencia de una cantidad relativamente grande de nodos.
- **VistaGrafo**: Permitirá la creación y visualización del grafo “Árbol de recubrimiento mínimo”.

2.5.7 Paquete “tablas”

En el paquete “tablas” están contenida toda la jerarquía de clases tipo *“Tabla”*, así como todas las clases involucradas en la creación de las *“tablas de entrada y salida”*. También podremos encontrar las clases que nos permitirán obtener los valores entrados por el usuario mediante las *“tablas de entrada”*.

Estrategias de construcción y obtención de valores

En el epígrafe 2.4.5 se mencionaron diferentes estrategias para la creación de las *"tablas de entrada y salida"*, así como para la obtención de los valores entrados por el usuario mediante las *"tablas de entrada"*. Debido a lo anteriormente expuesto se implementó el patrón de diseño *"Strategy"*.

Siguiendo el diagrama de clases básico de patrón *"Strategy"* (Figura 2-10), se crearon las siguientes tres interfaces:

- ComportamientoMetodoConstruir_Entrada – Contiene la firma del método *"contruir()"*. Este método contendrá todo el código necesario para la construcción de las "*tablas de entrada"*.
- ComportamientoMetodoConstruir_Salida – Contiene la firma del método *"contruir()"*.Este método contendrá todo el código necesario para la construcción de las "*tablas de salida"*.
- ComportamientoMetodoObtenerValores – Contiene la firma del método *"obternerValores()"*. Este método contendrá todo el código necesario para la obtención de los valores entrados por el usuario mediante las "*tablas de entrada"*.

En el siguiente diagrama de clases podremos observar las diferentes clases que implementan las interfaces anteriormente mencionadas. Cada una de estas clases representa una estrategia determinada, y redefinen el método de la interfaz que implementan de acuerdo a la estrategia que desean seguir.

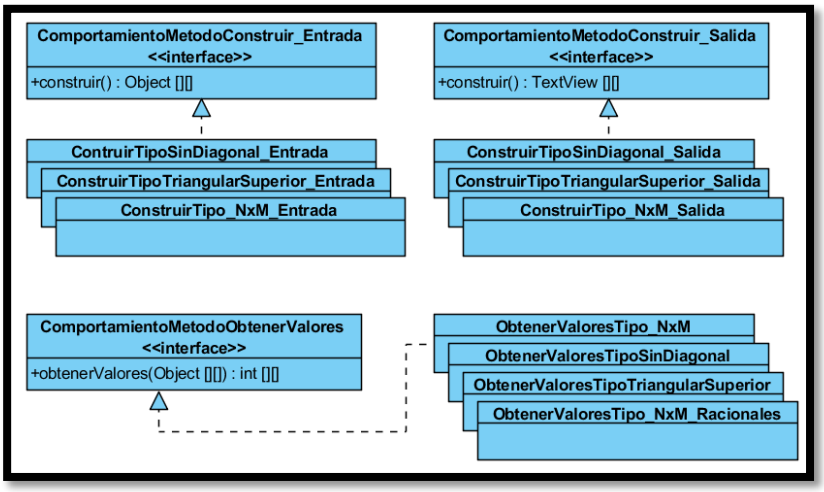

Figura 2- 20 Estrategias para la creación de las "tablas de entrada y salida" y para la obtención de valores a partir de las "tablas de entrada"

Jerarquía de clases tipo "Tabla"

El paquete "tablas" cuenta con una jerarquía de clases tipo *"Tabla"* de tres niveles. El primer nivel jerárquico contiene a la clase *"Tabla"*, la cual constituye el elemento raíz de la jerarquía. En el segundo nivel se tienen las clases *"ModoEntrada"* y *"ModoSalida"*, las cuales representan los dos tipos de tablas con que se contará: tablas de entrada y tablas de salida respectivamente. Por último, se tienen las clases del último nivel. Estas clases están asociadas a un método de optimización dado y están agrupadas de acuerdo a uno de los dos tipos de tablas existente.

En la siguiente figura se puede observar la jerarquía de clases tipo *"Tabla"* con las clases contenidas en cada uno de sus niveles.

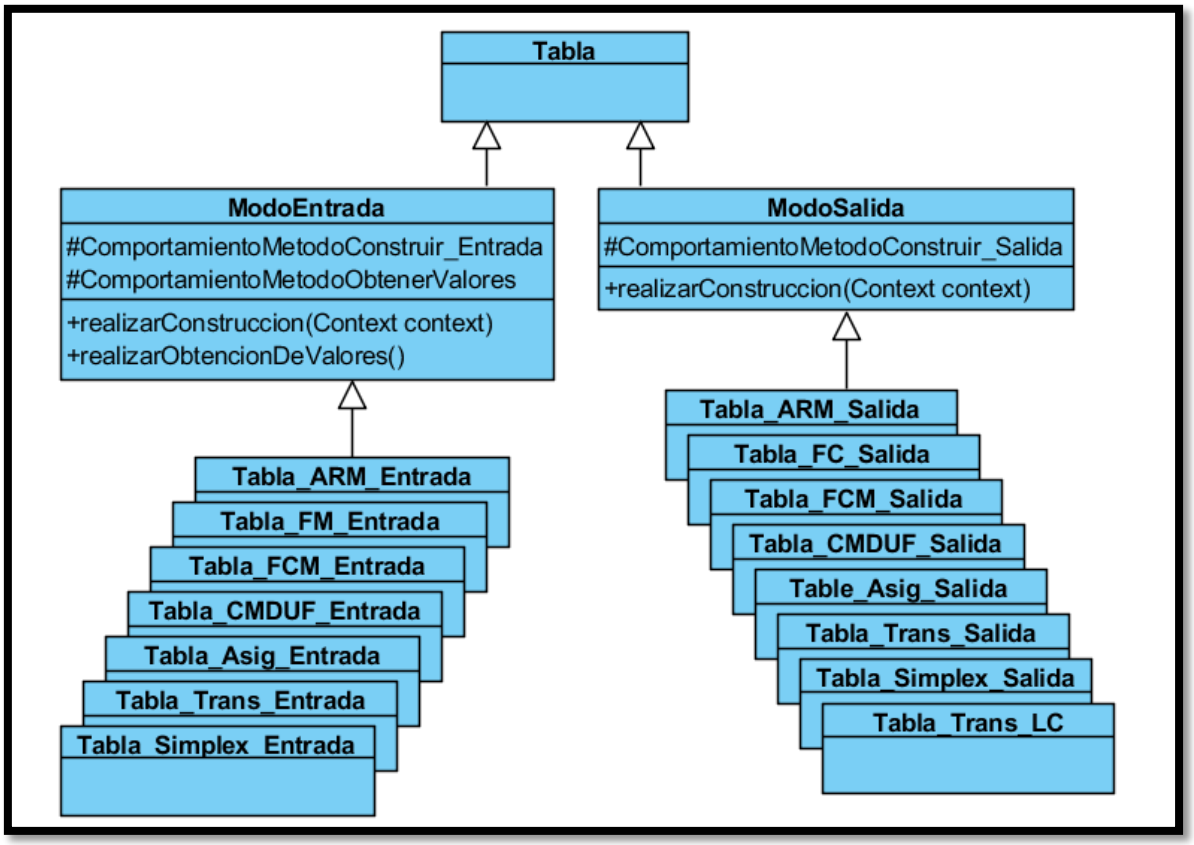

Figura 2- 21 Diagrama de clases tipo "Tabla" en el paquete "tablas"

Nótese que las clases del último nivel hacen referencia al método al cual están asociadas de la siguiente forma: *_<método>_*. La secuencia de caracteres <método> es equivalente a la usada en las actividades para indicar el método al cual se asocian.

En la Figura 2-21 se pudo apreciar que en el caso de las clases *"ModoEntrada"* y *"ModoSalida"* se especificaron ciertos atributos. Cada atributo constituye una referencia a una de las tres interfaces mencionadas con anterioridad, y permiten utilizar las diferentes estrategias de creación y obtención de valores ya definidas.

Otra observación que pudo realizarse a partir de la Figura 2-21 fue la mención de diferentes operaciones en las clases *"ModoEntrada"* y *"ModoSalida"*. La invocación de estas operaciones permite la ejecución del método correspondiente a la estrategia a la cual hacen referencia los atributos resaltados en las clases *"ModoEntrada"* y *"ModoSalida"*.

Definiendo el patrón "Factory Method"

El paquete "tablas" hace uso de un segundo patrón de diseño. Dicho patrón es "*Factory Method"*, y es utilizado con el objetivo de centralizar y encapsular la creación de objetos tipo *"Tablal_*_*"* (conjunto de clases contenidas en el tercer nivel de la jerarquía de clases tipo *"Tabla"*. Ver Figura 2.21).

Siguiendo el diagrama de clases básico del patrón *"Factory Method"* (Figura 2-11), se generó la siguiente jerarquía de clases.

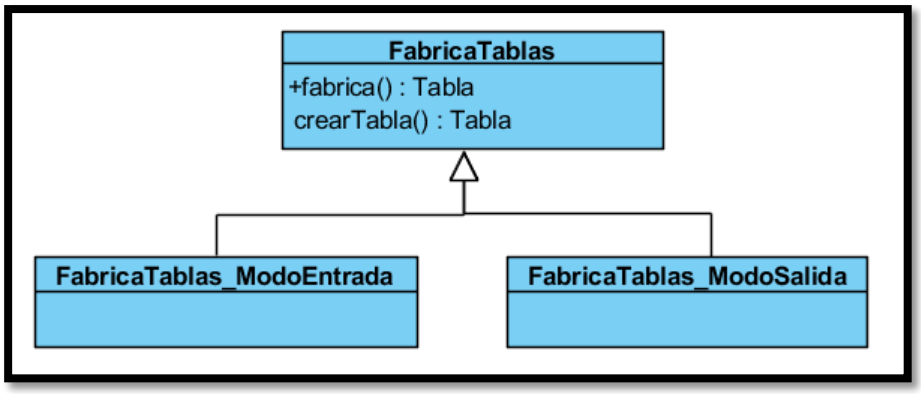

Figura 2- 22 Diagrama de clases referente al patrón "Factory Method" en el paquete "tablas"

En la figura anterior pudimos observar una jerarquía de dos niveles. En el primer nivel se encuentra la clase *"FabricaTablas"*. Esta clase contiene al método "*fabrica()"*, el cual permite crear una instancia de una de las quince clases de tipo *"Tabla_*_*"* contenidas en el tercer nivel de la jerarquía de clases tipo *"Tabla"*. Otro método es el método *"crearTabla()"*, el cual es utilizado por el método *"fabrica"* para la creación de objetos. El método es abstracto y será definido por las subclases *"FabricaTablas_ModoEntrada"* y

"FabricaTablas_ModoSalida". Estas últimas clases mencionadas, representan una fábrica de tablas de acuerdo a los dos tipos de tablas que pueden crearse: tablas de entrada y tablas de salida. Vale aclarar que el método que realmente participa en la creación de objetos será el método *"crearTabla()"*.

2.5.8 Paquete "util"

El paquete "util" contiene todo un conjunto de clases utilitarias. Este paquete es utilizado por los demás paquetes contenidos en la carpeta *"src"* (Figura 2-16). En la siguiente figura se muestra el diagrama de clases del paquete *"util"*.

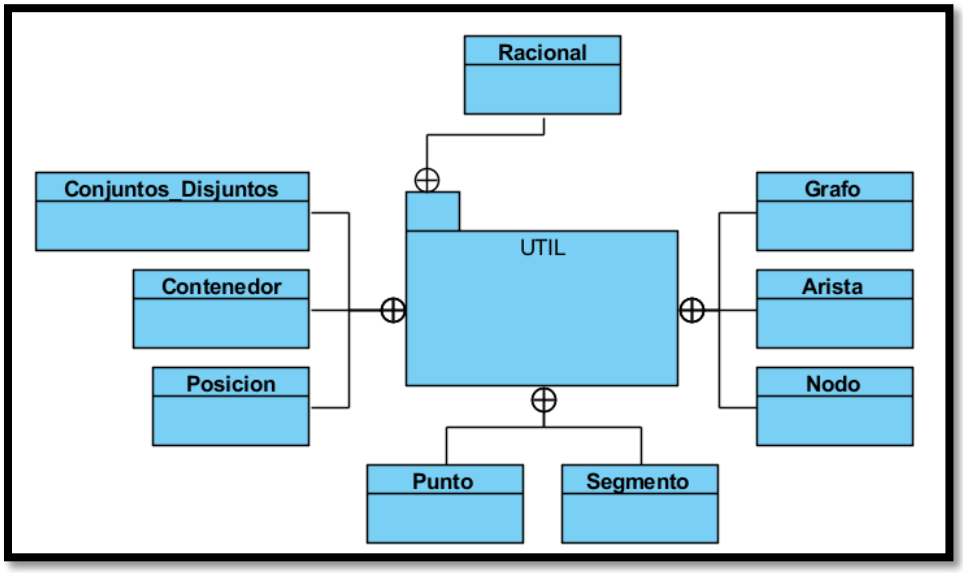

Figura 2- 23 Diagrama de clases del paquete "util"

2.5.9 Clases involucradas en el patrón MVC

Son tres los paquetes involucrados en el patrón de diseño MVC aplicado, estos son *"com.example.simio_app"*, *"graficos"* y *"util"*, quienes contienen las clases que conforman cada uno de los componentes que define el patrón.

En la Figura 2.24 podremos observar el diagrama de clases del patrón MVC aplicado.

El Modelo

El *Modelo* está conformado por tres clases contenidas en el paquete *"util"*: Las clases *"Grafo"*, *"Arista"* y *"Nodo"*. La clase *"Grafo"* será una composición de las clases *"Nodo"* y *"Arista"*, y contiene todo un conjunto de operaciones, las cuales permiten la variación del *Modelo*.

La Vista

La *Vista* está conformada por las clases *"VistaGrafo"* y *"VistaCaptacion"*, las cuales heredan de la clase *"View"*. Ambas clases contienen una referencia a un objeto tipo *"Grafo"*, y realizan la generación de gráficos mediante su método *"onDraw()"*, atendiendo al estado del *Modelo*.

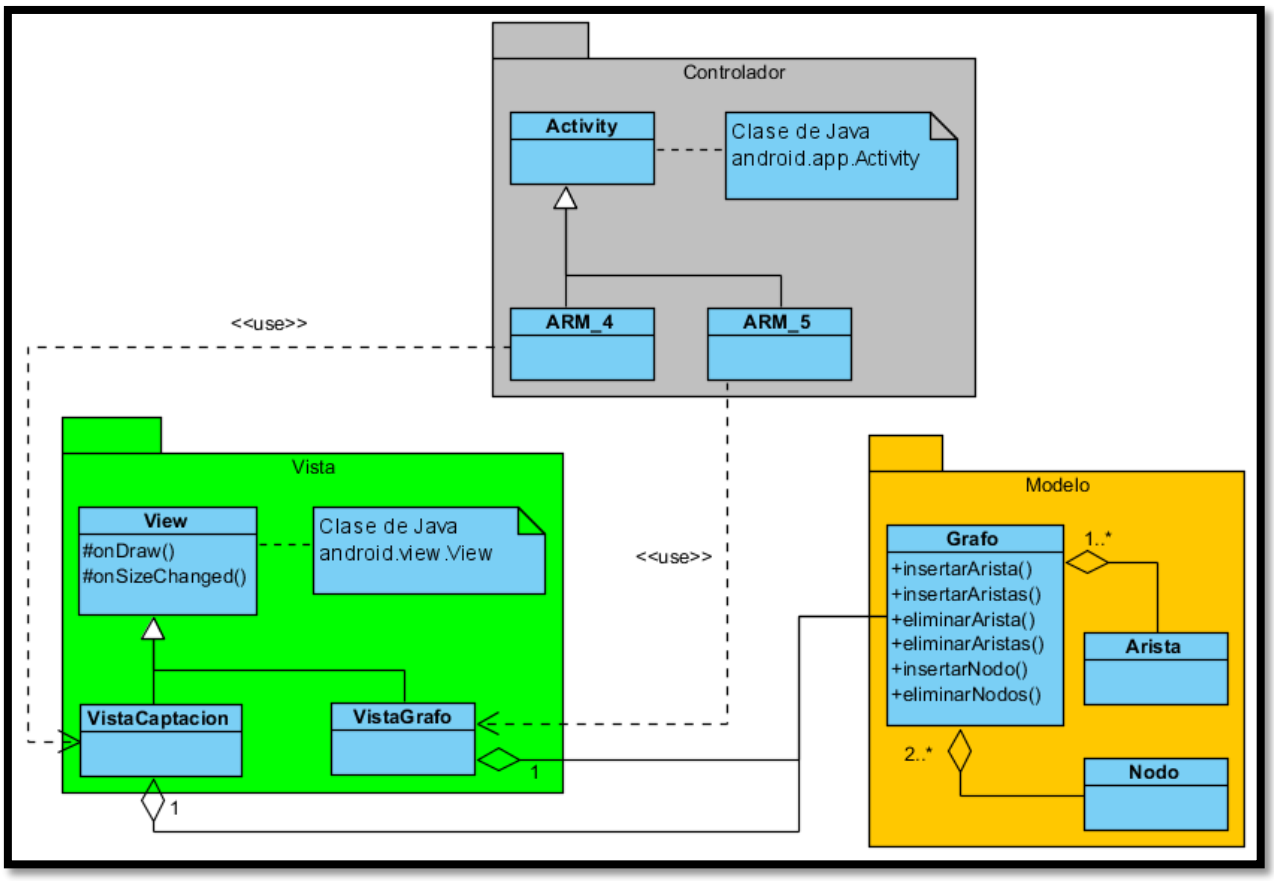

Figura 2- 24 Diagrama de clases del patrón MVC

El Controlador

El *Controlador* está conformado por las actividades *"ARM_4"* y *"ARM_5"*. Estas clases realizan el manejo de los eventos generados por las diferentes acciones realizadas por el usuario sobre la pantalla del dispositivo, e interpretarán que operación debe realizarse sobre el objeto "Grafo" contenido en las clases *"VistaGrafo"* y *"VistaCaptacion"* respectivamente.

2.6 Conclusiones parciales

Como conclusiones parciales del capítulo se tiene:

- Se utilizaron las herramientas SDK y Eclipse con plug-in ADT para el desarrollo de la aplicación.
- Se utilizaron las clases TableLayout y TableRow para la representación visual de tablas.
- Se utilizaron las clases Canvas y Paint para el manejo de gráficos 2D para la representación visual del grafo “Árbol de recubrimiento mínimo”.
- Se implementaron tres patrones de programación: Strategy, Factory Method y MVC, para dar flexibilidad al código a la hora de realizar futuros cambios.

CAPITULO 3. MÉTODOS DE INVESTIGACION DE OPERACIONES

La Investigación de Operaciones (conocida también como teoría de la toma de decisiones o programación matemática) (IO) es una rama de las matemáticas que consiste en el uso de modelos matemáticos, estadística y algoritmos con objeto de realizar un proceso de toma de decisiones. Frecuentemente trata el estudio de complejos sistemas reales, con la finalidad de mejorar (u optimizar) su funcionamiento. La Investigación de Operaciones permite el análisis de la toma de decisiones, para determinar cómo se puede optimizar un objetivo definido, como la maximización de los beneficios o la minimización de costos.

En el presente capítulo se dará un descripción de cada método de Investigación de Operaciones implementado en la aplicación, así como, una descripción de la implementación realizada para llevar a cabo le ejecución del mismo.

3.1 Simplex

El método Simplex es un procedimiento iterativo que permite mejorar la solución de una función objetivo en cada paso, solamente es posible aplicarlo a la programación lineal. El proceso concluye cuando no es posible continuar mejorando dicho valor, es decir, se ha alcanzado la solución óptima (el mayor o menor valor posible, según el caso, para el que se satisfacen todas las restricciones).

Partiendo del valor de la función objetivo en un punto cualquiera, el procedimiento consiste en buscar otro punto que mejore el valor anterior. Como bien puede verse en el método Gráfico (Figura 3-1), dichos puntos son los vértices del polígono que constituye la región determinada por las restricciones a las que se encuentra sujeto el problema (llamada región factible). La búsqueda se realiza mediante desplazamientos por las aristas del polígono, desde el vértice actual hasta uno adyacente que mejore el valor de la función objetivo. Siempre que exista región factible, como su número de vértices y de aristas es finito, será posible encontrar la solución.

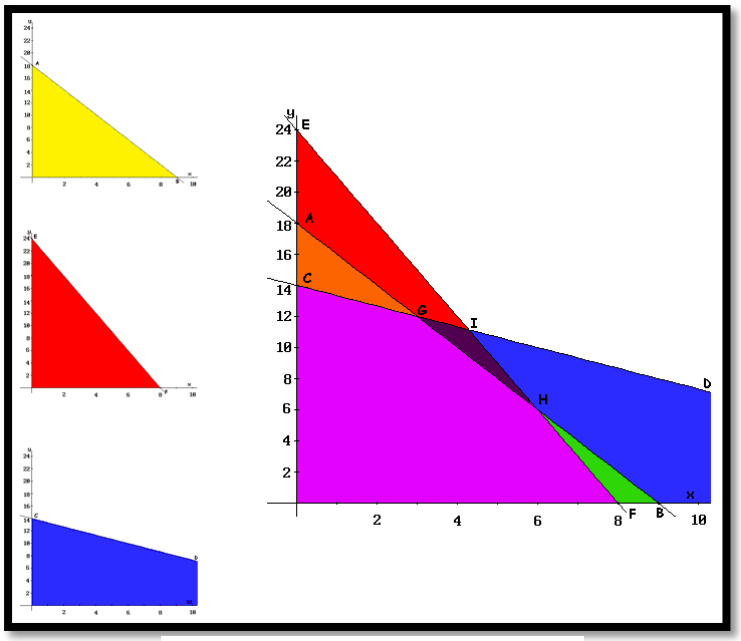

Figura 3- 1 Método Gráfico del Simplex

El método Simplex se basa en la siguiente propiedad: si la función objetivo Z no toma su valor máximo en el vértice A, entonces existe una arista que parte de A y a lo largo de la cual el valor de Z aumenta.

Será necesario tener en cuenta que el método Simplex implementado únicamente trabaja con restricciones del problema cuyas inecuaciones sean del tipo "≤" (menor o igual) y sus coeficientes independientes sean mayores o iguales a 0. Por tanto habrá que estandarizar las restricciones para que cumplan estos requisitos antes de iniciar el algoritmo del Simplex. En caso de que después de éste proceso aparezcan restricciones del tipo "≥" (mayor o igual) o "=" (igualdad), o no se puedan cambiar, será necesario emplear otros métodos de resolución, siendo el más común el método de las Dos Fases.

La forma estándar del modelo de problema consta de una función objetivo sujeta a determinadas restricciones:

Función objetivo: $c_1x_1 + c_2x_2 + \ldots + c_nx_n$

Sujeto a:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2$$
$$\ldots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m$$

$$x_1,\ldots, x_n \geq 0$$

El modelo debe cumplir las siguientes condiciones:

1. El objetivo consistirá en minimizar el valor de la función objetivo (por ejemplo, reducir pérdidas).
2. Todas las restricciones deben ser ecuaciones de desigualdad (≤).
3. Todas las variables (x_i) deben tener valor positivo o nulo (condición de no negatividad).
4. Los términos independientes (b_i) de cada ecuación deben ser no negativos.
5. Al menos un coeficiente de la función objetivo debe ser negativo.
6. Si el problema es de maximización deberá multiplicarse por (-1) toda la función objetivo para llevar el problema a minimización.

Para ejecutar el método Simplex una vez estandarizado el modelo, el problema deberá ser transformado. Dicha transformación consiste en agregar una variable de holgura *hi* y cambiar la desigualdad de las restricciones a igualdades (=).
Transformación del modelo:

Función objetivo: $c_1x_1 + c_2x_2 + ... + c_nx_n$

Sujeto a:
$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n + h_1 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n + h_2 = b_2$$
$$...$$
$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n + h_m = b_m$$
$$x_1,..., x_n \geq 0$$

Es necesario adaptar el problema modelado a la forma estándar para poder aplicar la transformación y así poder aplicar el algoritmo del Simplex.

3.2 Flujo de costo mínimo

El problema básico "Flujo de costo mínimo", es una generalización del problema del "Transporte".

Considérese una red conexa y dirigida con *n* nodos. Para cada nodo *i* hay un número b_i que representa la *oferta* disponible en el nodo (Si $b_i < 0$, entonces hay una demanda requerida). Se supone que la red esta *equilibrada* en el sentido de que

$$\sum_{i=1}^{n} b_i = 0$$

Asociado a cada arco *(i, j)* hay un número c_{ij} que representa el costo unitario del flujo en este arco. El problema de "Flujo de costo mínimo" consiste en determinar los flujos $x_{ij} \geq 0$ en cada arco de la red, para que el flujo neto que entra a cada nodo *i* sea b_i, al tiempo que se minimiza el costo total. Expresado matemáticamente, el problema es

$$\text{Minimizar} \sum c_{ij} * x_{ij}$$

$$Sujeto\ a \sum_{i=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = b_i \quad i = 1,2, \dots, n$$

$$x_{ij} \geq 0 \quad i, j = 1.2 \dots, n.$$

3.3 Transporte

En el contexto de los problemas de "Transporte", se trabaja con *m* orígenes que tienen ciertas cantidades de una mercancía que ha de enviarse a *n* destinos para satisfacer las necesidades de la demanda. En concreto, el origen *i* contiene una cantidad a_i y el destino *j* necesita una cantidad b_j. Se supone que el sistema está *equilibrado* en el sentido de la oferta total es igual a la demanda total. Esto es,

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

Se supone que los números a_i y b_j, $i = 0, 1, 2, \dots, m$; $j = 0, 1, 2, \dots, n$, son no negativos, y de hecho, en muchas aplicaciones enteros no negativos. Hay un costo unitario c_{ij} relacionado con el envío de la mercancía del origen *i* al destino *j*. El problema es hallar el modelo de envío entre orígenes y destinos que satisfaga todas las necesidades y minimice el costo total de envío.

$$\text{Minimizar} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} * x_{ij}$$

$$Sujeto\ a \sum_{j=1}^{n} x_{ij} = a_i \ \ para\ i = 1, 2, \dots, m$$

$$Sujeto\ a \sum_{i=1}^{m} x_{ij} = b_j \ \ para\ j = 1, 2, \dots, n$$

$$x_{ij} \geq 0 \ \ para\ todo\ i\ y\ j.$$

El problema matemático junto con la suposición anterior es el problema general de trasporte. En el contexto de los envíos, las variables x_{ij} representan las cantidades de mercancía enviadas del origen *i* al destino *j*.

3.4 Asignación

El problema de la Asignación es, por dos razones, un caso muy especial del problema del Transporte. Primera, las áreas de aplicación en las que aparece suelen ser bastante diferentes de las correspondientes al problema más general del Transporte, y segunda, su estructura es teóricamente significativa.

El ejemplo clásico del problema de la Asignación es el de la asignación de *n* obreros a *m* tareas. Si al obrero *i* se le asigna la tarea *j*, hay un beneficio de c_{ij}. A cada obrero debe de asignársele exactamente una tarea y cada tarea debe de terne asignado un obrero. El objetivo a alcanzar es realizar una asignación para maximizar o minimizar el valor total de la asignación.

La formulación general del problema de la Asignación es hallar x_{ij}, $i = 0, 1, 2, \dots, n$; $j = 0, 1, 2, \dots, n$ de forma tal que

$$\text{Minimizar} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} * x_{ij}$$

$$Sujeto\ a \sum_{j=1}^{n} x_{ij} = 1 \ \ para\ i = 1, 2, \dots, n$$

$$Sujeto\ a \sum_{i=1}^{n} x_{ij} = 1 \ \ para\ j = 1, 2, \dots, n$$

$$x_{ij} \geq 0 \ \ para \ \ i = 1, 2, \dots, n \ \ y \ \ j = 1, 2, \dots, n.$$

Es necesario que toda x_{ij} tome valores cero o uno, pues de no ser así la solución no es significativa, ya que no se pueden hacer asignaciones fraccionarias. Cumpliéndose así lo siguiente.

Teorema: Para cualquier solución factible básica del problema de Asignación toda x_{ij} igual a cero o a uno.

De esto se desprende que hay a lo sumo *n* variables básicas que tengan valor uno, pues no puede haber más que un solo uno, como máximo, en cada fila y columna.

3.5 Flujo máximo

En el contexto de los problemas de redes el "Flujo máximo" trabaja con una red dirigida y conexa, con un solo nodo *fuente* y un solo nodo *sumidero*. Por ejemplo, los nodos *l* y *m* respectivamente. El resto de los nodos deben de satisfacer el estricto requisito de conservación, esto es, el flujo neto de estos nodos debe ser cero. Sin embargo, la fuente puede tener un flujo neto de salida, y el sumidero, un flujo neto de entrada. El flujo de salida *f* de la fuente será igual al flujo de entrada del sumidero, como consecuencia del requisito de conservación de los demás nodos. Cada arco *(i, j)* tiene asociado una capacidad máxima no negativa k_{ij}. El flujo que pasa por un arco dado es denotado por x_{ij}.

Un conjunto de flujos de arco que satisfaga estas condiciones se dice que es un flujo en la red de valor *f*. EL problema del "Flujo máximo" es determinar el flujo máximo que se puede establecer en dicha red. Su forma estricta es

$$Maximizar \quad f(i,j)$$

$$Sujeto\ a \ \sum_{j=1}^{n} x_{lj} = \sum_{j=1}^{n} x_{jl} - f = 0$$

$$Sujeto\ a \ \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = 0, \qquad i, j \neq l, m$$

$$Sujeto\ a \ \sum_{j=1}^{n} x_{mj} - \sum_{j=1}^{n} x_{jm} + f = 0$$

$$0 \leq x_{ij} \leq k_{ij}, \qquad toda\ i, j,$$

donde solo se permiten los pares *i, j* correspondientes a los arcos.

3.6 Camino más corto (RMC)

En el marco de los problemas de "Camino más corto" se considera una red conexa y no dirigida con dos nodos especiales, llamados origen y destino. A cada una de los arcos no dirigidos se asocia una distancia no negativa. El objetivo del análisis es encontrar la ruta más corta, es decir, la trayectoria con la mínima distancia total, que va del origen al destino.

3.7 Árbol de expansión mínima

En el contexto de los problemas de "Árbol de expansión mínima" se considera una red no dirigida y conexa, donde cada arco está asociado a una medida de longitud positiva (tiempo, distancia, costo, etc.). El objetivo a alcanzar tiene que ver con la determinación de los arcos que pueden unir todos los nodos de la red, tal que se minimice la suma de las longitudes de los arcos escogidos. No se deben incluir ciclos en la solución del problema.

Para crear el "Árbol de expansión mínima" se tienen las siguientes características:

1. Se tienen los nodos de la red pero no los arcos. En su lugar se proporcionan los arcos potenciales y la longitud positiva para cada uno si se inserta en la red.
2. Se desea diseñar la red con suficientes ligaduras para satisfacer el requisito de que haya un camino entre cada par de nodos.
3. El objetivo es satisfacer este requisito de manera que se minimice la longitud total de las ligaduras insertadas en la red.

Una red con *n* nodos requiere sólo *(n-1)* arcos para proporcionar una trayectoria entre cada par de nodos. Los *(n-1)* arcos deben elegirse de tal manera que la red resultante forme un árbol de expansión.

CONCLUSIONES

- Se seleccionaron las clases *TableLayout* y *TableRow* para el trabajo con tablas y la clase *View* junto con las clases *Canvas* y *Paint*, contenidas en el paquete *android.graphics*, para el trabajo con grafos.
- Se utilizaron los patrones de programación *Strategy*, *Factory Method y MVC* para organizar el código y darle más flexibilidad a la hora de hacer modificaciones.
- Se utilizaron las herramientas SDK y Eclipse con plug-in ADT para el desarrollo de la aplicación.
- Se implementó una aplicación Android que hace una simulación paso a paso de los principales métodos de Investigación de Operaciones y ayuda a la docencia de las asignaturas que imparten estos métodos.

RECOMENDACIONES

- Extender la aplicación con la incorporación de nuevos métodos.
- Extender la aplicación a otros SO para dispositivos móviles.

REFERENCIAS BIBLIOGRÁFICAS

BURNETTE, E. 2009. *Hello, Android: introducing Google's mobile development platform*, Pragmatic Bookshelf.

DARWIN, I. 2012. *Android cookbook*, " O'Reilly Media, Inc.".

DEVELOPERS, A. 2010. Declaring Layout.

DORNIN, L., MEIKE, B. & NAKAMURA, M. 2011. *Programming Android*, " O'Reilly Media, Inc.".

FOWLER, M. 2002. *Patterns of enterprise application architecture*, Addison-Wesley Longman Publishing Co., Inc.

FREEMAN, E., ROBSON, E., BATES, B. & SIERRA, K. 2004. *Head first design patterns*, " O'Reilly Media, Inc.".

GARGENTA, M. 2011. *Learning android*, " O'Reilly Media, Inc.".

GIRONÉS, J. T. 2012. *El gran libro de Android*, Marcombo.

LEE, W.-M. 2012. *Beginning android 4 application Development*, John Wiley & Sons.

MEIER, R. 2012. *Professional Android 4 application development*, John Wiley & Sons.

OLIVER, S. G. 2011. *Manual-Programacion-Android-v2*.

RAMÓN INVARATO MENÉNDEZ, R. M. G., J. ALBERTO CASERO GUTIÉRRRES 2014. *Android 100%*.

RIVERA, M., JULIANA, Y., SANDOVAL CARDONA, J., FRANCO, T. & ALBERTO, S. 2012. Sistema operativo Android: características y funcionalidad para dispositivos móviles.

SATYA KOMATINENI, D. M. 2012. *Pro Android 4*, Apress.

SOMMERVILLE, I. 2009. *Software Engineering*, Addison-Wesley.

TOMÁS GIRONÉS, J. 2011a. La clase Paint en Android.

TOMÁS GIRONÉS, J. 2011b. Los Layouts en Android.