Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Redes Móviles, Inalámbricas y Distribuidas (ICARO)

Desarrollo de una Aplicación para la Captura del Tráfico Vehicular de Caracas Usando Teléfonos Inteligentes (*Smartphones*) con Sistema Operativo *Android*

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por los Bachilleres
German Enrique Mendoza Gonzalez
Julio César Arismendi Salazar
para optar al título de
Licenciado en Computación
Tutora: Profa. María Elena Villapol
Tutor: Prof. Adrián Bottini

Caracas 2 de Agosto de 2013

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Laboratorio de Redes Móviles, Inalámbricas y Distribuidas (ICARO)

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por los Bachilleres German Mendoza C.I.: 18.761.832 y Julio Arismendi C.I.: 17.857.592, con el título *Desarrollo de una Aplicación para la Captura del Tráfico Vehicular de Caracas Usando Teléfonos Inteligentes (Smartphones) con Sistema Operativo Android*, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 2 de Agosto de 2013, a las 3:00 pm, para que sus autores lo defendieran en forma pública en la Sala 1 de la Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela, lo cual se realizó mediante una exposición oral de su contenido, y luego respondieron satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas a los 2 días del mes de Agosto del año 2013, dejándose también constancia de que actuó como Coordinador del Jurado la Profesora María Elena Villapol.

_____ _____

Profa. María Elena Villapol (Tutora) Prof. Adrián Bottini (Tutor)

_____ _____

Prof. David Pérez (Jurado) Prof. Jaime Blanco (Jurado)

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Resumen

Los Sistemas de Posicionamiento Global han ofrecido a la sociedad un gran número de avances en aspectos personales, laborales y sociales. Poder obtener la ubicación precisa de un dispositivo que maneje alguna de las tecnologías de posicionamiento global permite efectuar tareas que antes eran difíciles o prácticamente imposibles de realizar. Los dispositivos móviles de última generación, en especial los teléfonos inteligentes, han incluido estas tecnologías en sus servicios, permitiendo a los desarrolladores la implementación de un gran número de programas que hagan uso de las mismas. La aplicación para dispositivos móviles desarrollada en este trabajo tiene como principal objetivo la captura de datos asociados a coordenadas y recorridos, los cuales pueden servir como base para diferentes modelos de movilidad, de tráfico vehicular, entre otros. Estos a su vez son base para muchos proyectos que busquen alcanzar algún tipo de mejora o automatización en este ámbito. Para lograr que la aplicación cumpliera con los requisitos establecidos fueron realizadas diferentes pruebas de funcionalidad, precisión y frecuencia en la actualización de posición. Al finalizar dichas pruebas se realizó una captura en un ambiente real, utilizando una unidad de transporte público.

Palabras claves: GPS, Tránsito Vehicular, Android, Smartphone, Aplicación Móvil.

Capítulo 1
Introducción

En los últimos años se ha observado un aumento considerable en la cantidad de vehículos a nivel global. Los embotellamientos de tráfico, el aumento de accidentes vehiculares y la desorganización en el sistema vial son solo algunas de las consecuencias acarreadas por esta situación, y que, particularmente en las grandes ciudades de Venezuela, se han ido acentuando día tras día.

En Caracas, una de las ciudades más pobladas de Venezuela, a la situación antes planteada se le suman otras propias de la región y que son bien conocidas por los habitantes que allí residen, como el irrespeto a las normas de tránsito, el mal estado de muchas de las calles, avenidas y autopistas, la imprudencia de algunos conductores y el deterioro de unidades de transporte público, entre otras. Lo que dificulta aún más la organización y control del sistema de tránsito de la ciudad.

Una tecnología que podría mejorar un poco esta situación y que ha tenido un gran auge en los últimos años es el Sistema de Posicionamiento Global GPS (Global Positioning System), el cual permite obtener una ubicación precisa de cualquier dispositivo que posea un receptor de señales GPS, como es el caso de la gran mayoría de teléfonos inteligentes que se han desarrollado en los últimos años. Todo esto ha impulsado notablemente el desarrollo de aplicaciones móviles que hacen uso de los servicios de posicionamiento global para distintos sectores de la sociedad.

El siguiente trabajo tiene como objetivo principal la explicación detallada del diseño, desarrollo e implementación de una aplicación para teléfonos inteligentes que permite recolectar automáticamente información sobre el tráfico en la ciudad de Caracas. Dicha aplicación nutrirá de datos un modelo de tráfico vehicular que se realizará posteriormente y que funcionará como base para la búsqueda de futuras soluciones a los distintos problemas antes planteados.

A continuación se muestran los temas que conforman cada uno de los 7 capítulos que se desarrollan en el siguiente trabajo: Capítulo 1: Introducción, Capítulo 2: GPS (Sistema de Posicionamiento Global), Capítulo 3: Sistema Operativo *Android* para Teléfonos Inteligentes, Capítulo 4: Metodología y Herramientas, Capítulo 5: Diseño e Implementación, Capítulo 6: Pruebas, Captura y Análisis de Datos, Capítulo 7: Conclusiones y Trabajos Futuros.

Capítulo 2
GPS (Sistema de Posicionamiento Global)

El GPS es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de incluso centímetros, aunque lo habitual son unos pocos metros de precisión.

El GPS tiene sus inicios en la década de 1960, donde varias organizaciones del gobierno estadounidense, incluyendo al DOD (*Department of Defence* - Departamento de Defensa), la NASA (*National Aeronautics and Space Administration* - Administración Nacional de Aeronáutica y del Espacio), y el DOT (*Department of Transport* - Departamento de Transporte), estaban interesados en la elaboración de un sistema satelital para detección de posición en tres dimensiones [1]. Para que el sistema fuera considerado óptimo debía tener los siguientes atributos: cobertura global, funcionamiento continuo bajo cualquier tipo de clima y capacidad para servir a plataformas dinámicas que requieran una alta precisión.

De esta idea nació un sistema conocido como *Transit*, el cual comenzó a funcionar en 1964 y fue ampliamente aceptado para su uso en plataformas poco dinámicas. Sin embargo, dicho sistema presentaba muchas limitaciones, lo que obligó a la marina a trabajar en el desarrollo de otro sistema de navegación satelital que cumpliera con las capacidades deseadas. Algunas variantes del sistema original *Transit* fueron propuestas por sus desarrolladores en los Laboratorios de Física Aplicada de la Universidad Johns Hopkins. Al mismo tiempo, el NRL (*Naval Research Laboratory* - Laboratorio de Investigación Naval) estaba llevando a cabo experimentos con relojes de gran estabilidad y precisión para lograr una transferencia precisa del tiempo. Este programa fue denotado como Tima Rim [1].

En 1969, la OSD (*Office of the Secretary of Defense* - Oficina de Secretaria de la Defensa) estableció el DNSS (*Defense Navigation Satellite System* - Sistema de Defensa por Navegación Satelital), programa para consolidar los esfuerzos independientes de desarrollo de cada servicio militar y así formar un solo uso conjunto del sistema. La OSD también estableció un grupo de dirección ejecutiva de navegación satelital, el cual estaba encargado de determinar la viabilidad de la DNSS y la planificación de su desarrollo. A partir de este esfuerzo, el concepto de sistema de NAVSTAR GPS se formó.

2.1 Composición del Sistema GPS

Al hablar del GPS se piensa inmediatamente en un equipo que proporciona una posición, no obstante, el sistema GPS no se limita a dicho instrumento, pues está compuesto de tres elementos distintos, denominados segmentos. El primer segmento, formado por los satélites, es llamado segmento espacial. El segundo segmento que comprende las estaciones de control, se denomina segmento de control. El tercero corresponde a los receptores GPS y se conoce como segmento del usuario [2].

2.1.1 Segmento Espacial

El SS (*Space Segment* - Segmento Espacial) está formado por una constelación de 24 satélites, llamados SV (*Space Vehicles* - Vehículos Espaciales). Circundan la Tierra a 20.200 kilómetros de altitud y forman 6 órbitas diferentes con 4 satélites en cada una. La figura 2.1 muestra esquemáticamente la disposición de los satélites alrededor de la Tierra [1]. Cada satélite efectúa una vuelta completa alrededor de la Tierra cada 12 horas, siguiendo el mismo recorrido todos los días (visto desde la Tierra). Luego de 24 horas (menos 4 minutos a causa del desplazamiento de la Tierra alrededor del Sol) se presenta exactamente en el mismo lugar y con la misma configuración respecto a los demás satélites.

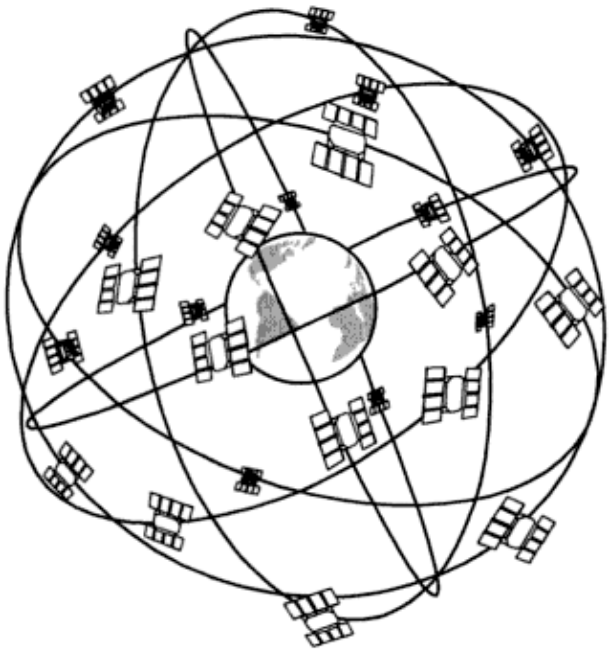

Figura 2.1: Constelación de satélites.

Cada satélite transmite, de manera permanente, un mensaje de navegación indicando su posición orbital así como la hora exacta de la emisión de dicho mensaje. También se transmite un almanaque que proporciona la posición y el estado operativo de cada satélite. Dicho almanaque permite a los receptores GPS que puedan localizar todos los demás satélites a partir de la detección de uno de ellos. Los primeros satélites puestos en órbita hasta 1989 eran de estudio y no poseían toda la capacidad de los actuales. El conjunto de dichos satélites fue denominado bloque I. Actualmente no hay ninguno de ellos que siga activo.

Los actuales satélites pertenecen al bloque II. Han sido puestos en órbita satélites más eficaces que son denominados del bloque IIA y luego del bloque IIR. Los satélites de los bloques II, IIA y IIR van equipados con cuatro relojes atómicos, dos de cesio (Cs) y dos de rubidio (Rb). Así pueden permanecer 14 días sin contactar con las estaciones de tierra, y siguen conservando suficiente precisión. Disponen de un *software* de diagnóstico interno que les permite detectar gran parte de las anomalías de funcionamiento y tomar las medidas pertinentes.

En la hipótesis de una destrucción de las estaciones terrestres, los satélites del bloque IIA pueden seguir transmitiendo sus mensajes durante un periodo de seis meses. Dichos satélites no son capaces de modificar sus mensajes, por lo que su precisión merma a medida que varía la órbita del satélite. Los satélites del bloque IIR son capaces de crear sus propios mensajes en función de su órbita y pueden permanecer mucho tiempo sin contactar con la Tierra. La vida útil de un satélite es de seis a siete años y medio.

La disposición de los satélites hace posible tener, en el 99,9% de los casos, un mínimo de 4 satélites visibles a 5° o más por encima del horizonte sobre cualquier punto sobre la tierra. Esto significa que, durante un minuto y medio por día, el sistema GPS puede indicar una posición que no sea lo bastante fiable. Además, si la zona no está completamente despejada, es decir si no puede verse el cielo por encima de 5° del horizonte, la cobertura de los satélites puede ser insuficiente. En ciertos lugares raros del globo, en las llamadas zonas de recepción difícil, la cobertura no es segura por aproximadamente 45 minutos por día.

Regularmente también sucede que uno o varios satélites se hallan momentáneamente fuera de servicio. Por lo general, esto es debido a un mantenimiento periódico del satélite o a un problema técnico. Lo normal es que un satélite no esté más de 24 horas fuera de servicio; no hay más de cuatro satélites fuera de servicio cada mes y nunca más de tres al mismo tiempo. Cuando hay tres satélites fuera de servicio, la cobertura puede ser insuficiente durante cerca de una hora por día. La cantidad de satélites visibles varía a lo largo del día, existiendo ocho como promedio. La figura 2.2 muestra el porcentaje de satélites visibles durante un periodo de 24 horas [1].

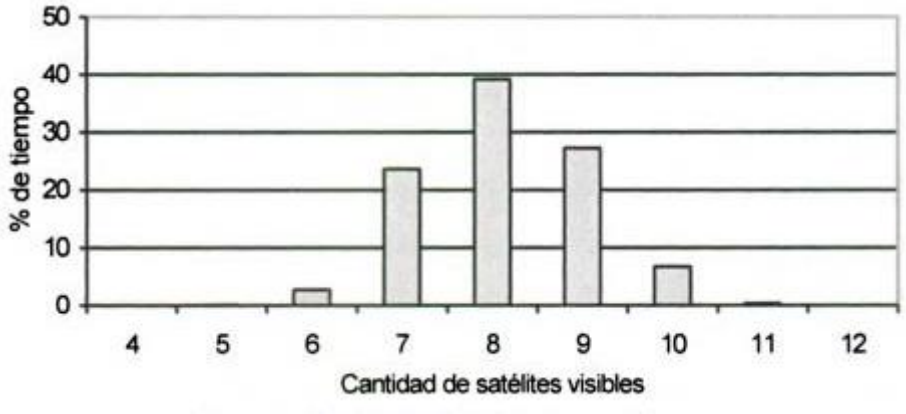

Figura 2.2: Visibilidad de los satélites.

Los satélites no siempre respetan la cobertura ideal esperada. Ello se debe al hecho de que los satélites jamás se encuentran exactamente en la órbita prevista y a que dicha órbita oscila permanentemente debido a fenómenos imprevisibles, tales como las variaciones de los campos magnéticos.

2.1.2 Segmento de Control

El CS (*Control Segment* - Segmento de Control) está formado por cinco estaciones de vigilancia distribuidas alrededor del planeta. También incluye una estación principal que asegura el correcto funcionamiento del sistema calculando las correcciones a aplicar a los mensajes emitidos por los satélites. Hay tres antenas terrestres que transmiten tales correcciones a los satélites. Las cinco estaciones se encuentran en Hawai, en Kwajalein en las islas Marshall, en la isla de Ascensión, en Diego García y en Colorado Springs. Su misión es captar todas las señales emitidas por los satélites, acumular los mensajes recibidos y transmitir todas las informaciones recogidas a la estación principal.

2.1.3 Segmento del Usuario

El segmento del usuario comprende la antena de recepción y el receptor/microprocesador GPS que realiza todos los cálculos a partir de los mensajes de navegación recibidos de los satélites. También puede proporcionar informaciones sobre posición, velocidad, ruta, hora y fecha, así como todas las demás informaciones necesarias para la navegación. Existen varios tipos de receptores según su aplicación y grado de precisión deseada.

El GPS proporciona dos tipos de posicionamientos, el PPS (*Precise Positioning Service* - Servicio de Posicionamiento Preciso) y el SPS (*Standard Positioning Service* - Servicio de Posicionamiento Estándar). Solo es de libre disposición el segundo, estando el PPS reservado al ejército norteamericano.

2.2 Control y Política del GPS

El GPS está totalmente realizado, financiado y controlado por el Departamento de Defensa de los Estados Unidos (DOD). Este organismo es el único que decide la implantación del GPS y controla el funcionamiento del sistema, con la coordinación del Departamento de Transportes Americano (DOT). Sin embargo, el DOD está dispuesto a estudiar cualquier solicitud que pueda reforzar la seguridad de los Estados Unidos. Además, no queda excluido que algún día se encargue del control y financiación del GPS un servicio especial que agrupe militares y civiles [2].

2.3 Funcionamiento

El principio utilizado por el GPS para determinar la ubicación se basa en la medición de la distancia entre el receptor GPS y varios satélites. Mediante el receptor se conoce la posición de cada satélite en el espacio con mucha precisión. Cada satélite transmite permanentemente su posición exacta con respecto a la Tierra. Además, también indica la hora exacta de la transmisión del mensaje. Calculando el tiempo requerido por las señales para llegar al receptor, se establece la distancia al satélite.

Gracias a la distancia y a la posición de un satélite, puede trazarse un círculo imaginario en la superficie de la Tierra dentro del cual se halla forzosamente el receptor. La intersección de varios de dichos círculos permite conocer la posición exacta del receptor.

2.4 Medición de la Distancia de un Satélite

La medición de la distancia que separa un satélite del receptor se basa en la propagación de las ondas electromagnéticas. El tiempo empleado por una señal para llegar al receptor es directamente proporcional a la distancia recorrida. Las señales se propagan a una

velocidad del orden de 300.000 kilómetros por segundo; cuanto más alejado esté el receptor del satélite, más tiempo tardará la señal en llegar a él.

Para que el receptor pueda medir el tiempo que tarda la señal en llegar, el satélite proporciona la hora exacta a que se ha emitido, el receptor compara la hora de emisión con la de recepción de la señal y calcula la distancia al satélite. Dicha distancia recibe el nombre de pseudodistancia (*pseudorange*).

2.5 Sincronización del Reloj del Receptor

Para ajustar su reloj, el receptor GPS utiliza la intersección de los círculos de posición. Si el reloj del receptor va adelantado, el tiempo de propagación de la señal será mayor que el realmente empleado por dicha señal para llegar hasta el receptor. Esto hará que los satélites parezcan más lejanos de lo que realmente están, y los círculos de posición también serán más grandes de lo debido.

Si se toman tres círculos de posición con respecto a tres satélites, dichos círculos quedan parcialmente solapados, formando una zona en cuyo centro se encuentra la ubicación del receptor. Entonces, el receptor retrasará su reloj hasta que dicha zona sea lo más pequeña posible. La figura 2.3 presenta un ejemplo de círculos de posición cuando el reloj del receptor va adelantado [1].

Figura 2.3: Reloj del receptor adelantado.

Por el contrario, cuando el reloj del receptor va retrasado, ve los círculos de posición menores de lo que son en la realidad. Los círculos se separan unos de otros. La figura 2.4 muestra un ejemplo en que el receptor debe adelantar su reloj hasta que los tres círculos se corten en el mismo lugar [1].

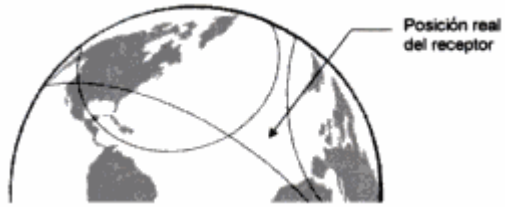

Figura 2.4: Reloj del receptor retrasado.

Cuando el reloj del receptor GPS está perfectamente sincronizado con el de los satélites, los tres círculos se cortan exactamente en un solo punto (figura 2.5).

Figura 2.5: Reloj del receptor sincronizado.

2.6 Receptores GPS

Aunque la indicación de la posición constituye la función básica de cualquier receptor GPS existen otras informaciones útiles que también son proporcionadas. Hay receptores de características muy diferentes, cada uno de los cuales ofrece servicios interesantes. Ante la cantidad de modelos existentes y la variedad de sus servicios, no siempre resulta fácil elegir el más conveniente. Cada receptor responde a una determinada necesidad, en función de sus características técnicas. Un mejor conocimiento de tales características permitirá aprovechar mejor las funciones de un receptor.

2.7 Precisión de la Posición

Según el tipo de filtrado y los algoritmos de posicionamiento utilizados por el receptor, la precisión de la posición puede variar en proporciones nada despreciables. Debido a que esta información no suele estar disponible, se requiere de estudio comparativo para obtener información sobre las prestaciones de un determinado equipo.

2.8 Formato de Presentación de Datos

La posición se presenta en varios formados, según el uso del equipo y los países [2]. Es de gran utilidad poder modificar el modo de presentación de la latitud y la longitud según los siguientes formatos:

- hddd.ddddd°: horas y fracciones de hora.
- hddd°mm.mmm': horas, minutos y fracciones de minuto.
- hddd°mm'ss.s": horas, minutos, segundos y fracciones de segundo.

Para utilizar en tierra, es necesaria la presentación de coordenadas UTM (*Universal Transverse Mercator* - Universal Transversal de Mercator). Cerca de las regiones polares, la presentación de coordenadas UPS (*Universal Polar Stereographic* - Proyección Estereográfica Polar) suele ser indispensable. Algunos países disponen de un sistema de referencia propio, entre ellos figuran Suiza, Irlanda o Inglaterra. Para usar el GPS con mapas de estos países, hay que vigilar que el receptor pueda presentar la posición en el formato requerido. Ciertos receptores también permiten personalizar el modo de presentación.

2.9 Tiempo Requerido para la Primera Posición

Cada vez que se pone un receptor en marcha, antes de calcular la posición ha de asegurarse que sólo utiliza informaciones y efemérides recientemente actualizadas. Igualmente tiene que asegurarse que no usa nuevas informaciones combinadas con otras más antiguas. Por tanto, un receptor debe recibir un cierto número de información antes de poder calcular la primera posición. El tiempo requerido para dicha primera posición depende fundamentalmente del tiempo transcurrido desde la última actualización de las efemérides, de la visibilidad de los satélites y también de la calidad del receptor.

2.10 Actualización de la Posición

Para alcanzar un suficiente nivel de confianza en la posición, las indicaciones deben ser actualizadas cada segundo. No obstante, algunos receptores disponen de una modalidad que permite actualizar la posición a un menor ritmo (cada 2 a 5 segundos), con objeto de ahorrar en

el consumo de las baterías del receptor. Esta modalidad puede utilizarse cuando las variaciones de rumbo y velocidad tienen poca importancia.

2.11 Velocidad

La velocidad de desplazamiento se calcula a partir del cambio de posición entre dos o más mediciones. También puede obtenerse a partir del efecto Doppler producido por las señales recibidas de los satélites. El efecto Doppler es el fenómeno que explica por qué el ruido de un vehículo, por ejemplo, nos parece más agudo a medida que se nos aproxima y más grave cuando se aleja. Lo mismo ocurre con las señales emitidas por los satélites. Los receptores GPS pueden presentar diversas velocidades.

2.12 Teléfonos Inteligentes (*Smartphones*) como Receptores

La mayoría de los S*martphones* cuentan con un receptor de señales GPS integrado, es por esto que suelen ser utilizados como dispositivos de navegación. De hecho, algunos pueden tener incluso ventaja sobre un receptor GPS dedicado, ya que, al poder estar siempre conectados a la red celular y a Internet, permite la realización de búsquedas, detección de tráfico en tiempo real, entre muchos otros servicios disponibles a través de dicha conexión.

Incluso si el teléfono inteligente no posee un receptor GPS integrado o si las señales de los satélites no se encuentran disponibles, este puede proveer información acerca de su localización basándose en mediciones de la señal, como por ejemplo, el ángulo de aproximación a las torres celulares, el tiempo que tarda la señal en viajar entre torres y la fuerza con la que llega la señal a las torres [3].

Capítulo 3
Sistema Operativo *Android* para Teléfonos Inteligentes

Dentro de los dispositivos móviles, un *Smartphone* es una evolución del teléfono móvil tradicional que cuenta con ciertas características y prestaciones que lo acercan más a una computadora personal que a un teléfono tradicional.

Entre dichas características, se puede encontrar una mejora en la capacidad de proceso y almacenamiento de datos, conexión a Internet mediante *Wi-Fi*, pantalla táctil, posicionamiento geográfico, teclado QWERTY y diversas aplicaciones de usuario como navegador web, cliente de correo, aplicaciones ofimáticas, reproductores de vídeo y audio, etc., incluyendo la posibilidad de descargar e instalar otras nuevas.

Dado que los *Smartphones* se han convertido en omnipresentes computadores portátiles, accesibles a cualquier usuario, el desarrollo de aplicaciones ha aumentado debido a las diferentes necesidades surgidas. Por lo que usuarios y desarrolladores necesitan de un entorno de desarrollo que permitan la creación de aplicaciones únicas y especializadas que sean asequibles para todos.

3.1 Sistemas Operativo

Un Sistema Operativo es una capa de *software* sobre el *hardware* de la máquina que permite realizar dos funciones básicas: administración de recursos y ofrecer una interfaz amigable [4].

Con respecto a la primera función, un computador posee diferentes recursos de *hardware* y *software*. Ejemplos de dichos recursos son: CPU (*Central Processing Unit* – Unidad Central de Procesamiento), memoria principal, dispositivos de entrada/salida, y diferentes tipos de *software* (compiladores, enlazador/cargador, archivos, etc.). Es el Sistema Operativo el encargado de manejar los recursos y asignarlos a los programas de usuario de forma eficiente y justa. Las labores de administración abarcan lo siguiente:

- Administración del tiempo (planificación de CPU y disco).

- Administración del espacio (almacenamiento principal y secundario).
- Manejo de sincronización de procesos y abrazo mortal (interbloqueo).
- Manejo de estadísticas y estado del sistema.

Además, un sistema operativo esconde los detalles no placenteros de bajo nivel con la finalidad de proveer una interfaz amigable hacia ella. Con dicha interfaz la carga, manipulación, impresión, y ejecución de programas se realiza mediante comandos de alto nivel sin la necesidad de preocuparse por los detalles de bajo nivel. La capa provista por el Sistema Operativo transforma el *hardware* crudo de la máquina, en una máquina virtual o abstracta con funcionalidades agregadas (como la administración de recursos de forma automática). Además, los usuarios de la máquina tienen la ilusión de que son los únicos usuarios en el sistema, aun cuando la máquina podría operar en un ambiente multiusuarios. La interfaz amigable toma en consideración las siguientes tareas:

- Ambiente de ejecución (administración de procesos – creación, control y finalización, manipulación de archivos, manejo de interrupciones, soporte de entrada/salida, etc.).
- Detección y manejo de errores.
- Protección y seguridad.
- Tolerancia a fallas y recuperación de las mismas.

3.2 Sistema Operativo *Android*

Android constituye una pila de *software* pensada especialmente para dispositivos móviles y que incluye tanto un Sistema Operativo, como *middleware* y diversas aplicaciones de usuario. Representa la primera incursión seria de *Google* en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector. Su diseño cuenta, entre otras, con las siguientes características:

- Busca el desarrollo rápido de aplicaciones, que sean reutilizables y verdaderamente portables entre diferentes dispositivos.
- Los componentes básicos de las aplicaciones se pueden sustituir fácilmente por otros.
- Cuenta con su propia máquina virtual, *Dalvik*, que interpreta y ejecuta código escrito en *Java*.
- Permite la representación de gráficos 2D y 3D.

- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM (*Goblal System for Mobile* - Sistema Global para Móviles)
- Controla los diferentes elementos *hardware*: *Bluetooth*, *Wi-Fi*, cámara fotográfica o de video, GPS, acelerómetro, infrarrojos, etc., siempre y cuando el dispositivo móvil lo contemple.
- Cuenta con un entorno de desarrollo muy cuidado mediante un SDK (*Software Development Kit* - Conjunto de Herramientas para Desarrollar Software) disponible de forma gratuita.
- Ofrece un *plugin* para uno de los entornos de desarrollo más populares, *Eclipse*, y un emulador integrado para ejecutar las aplicaciones.

Todas las aplicaciones para *Android* se programan en lenguaje *Java* y son ejecutadas en una máquina virtual especialmente diseñada para esta plataforma, que ha sido bautizada con el nombre de *Dalvik*. El núcleo de *Android* está basado en Linux 2.6.

La licencia de distribución elegida para *Android* ha sido *Apache* 2.0 [5], lo que lo convierte en *software* de libre distribución. A los desarrolladores se les proporciona de forma gratuita un SDK y la opción de un *plugin* para el entorno de desarrollo *Eclipse*, así como un emulador integrado para su ejecución. Existe además disponible una amplia documentación de respaldo para este SDK [15].

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

Mejorar el desarrollo y enriquecer la experiencia del usuario se convierte, por tanto, en la gran filosofía de *Android* y en su principal objetivo.

3.2.1 Arquitectura del Sistema

La arquitectura de *Android* está formada por capas de *software* donde cada una puede utilizar los servicios de la capa inferior (ver figura 3.1).

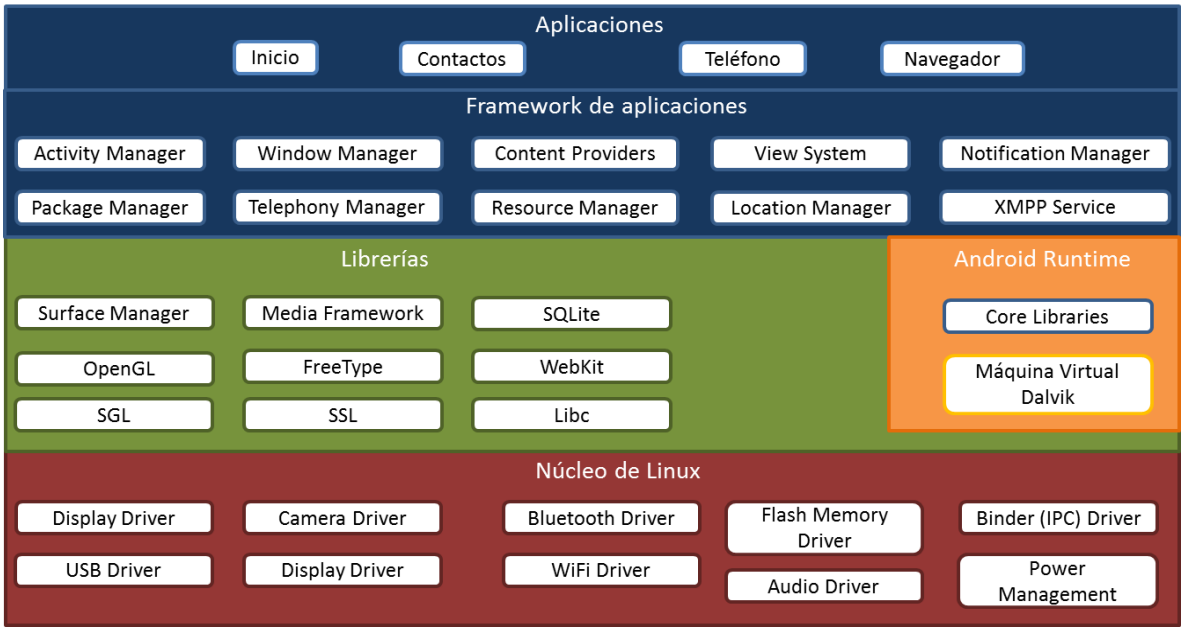

Figura 3.1: Arquitectura de *Android*.

La capa más inmediata es la que corresponde al núcleo. *Android* utiliza el núcleo de *Linux* 2.6 como una capa de abstracción para el *hardware* disponible en los dispositivos móviles. Esta capa contiene los *drivers* necesarios para que cualquier componente de *hardware* pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluya un nuevo elemento de *hardware*, lo primero que se debe realizar para que pueda ser utilizado desde *Android* es crear las librerías de control o *drivers* necesarios dentro de este *kernel* de Linux embebido en el propio *Android*.

La siguiente capa se corresponde con las librerías utilizadas por *Android*. Éstas han sido escritas utilizando *C/C++* y proporcionan a *Android* la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de *Android*.

Entre las librerías más importantes de este nivel, se pueden mencionar las siguientes:

- La librería *libc* incluye todas las cabeceras y funciones según el estándar del lenguaje *C*. Todas las demás librerías se definen en este lenguaje.
- La librería *Surface Manager* es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
- *OpenGL/SL y SGL* representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de *Android*. *OpenGL/SL* maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el *hardware* encargado de proporcionar gráficos 3D. Por otro lado, SGL proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de *Android* es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.
- La librería *Media Libraries* proporciona todos los *códecs* necesarios para el contenido multimedia soportado en *Android* (video, audio, imágenes estáticas y animadas, etc.).
- *FreeType*, permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.
- La librería SSL *(Secure Sockets Layer* - Capa de Conexión Segura) posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.
- A través de la librería *SQLite*, *Android* ofrece la creación y gestión de bases de datos relacionales, pudiendo transformar estructuras de datos en objetos fáciles de manejar por las aplicaciones.
- La librería *WebKit* proporciona un motor para las aplicaciones de tipo navegador, y forma el núcleo del actual navegador incluido por defecto en la plataforma *Android*.

Al mismo nivel que las librerías de *Android* se sitúa el entorno de ejecución. Éste lo constituyen las *CoreLibraries*, que son librerías con multitud de clases de *Java*, y la máquina virtual *Dalvik*.

Los dos últimos niveles de la arquitectura de *Android* están escritos enteramente en *Java*. El *framework* de aplicaciones representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para *Android*, ya sean las propias del dispositivo, las desarrolladas por *Google* o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API (*Application Programming Interface* - Interfaz de Programación de Aplicaciones) y el mismo *framework*, representado por este nivel.

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

- *Activity Manager*, importante conjunto de API que gestiona el ciclo de vida de las aplicaciones en *Android* (del que se hablará más adelante).
- *Window Manager*, gestiona las ventanas de las aplicaciones y utiliza la librería ya vista *Surface Manager*.
- *Telephone Manager*, incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).
- *Content Providers*, permite a cualquier aplicación compartir sus datos con las demás aplicaciones de *Android*. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.
- *View System*, proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, *checkboxes*, tamaño de ventanas, control de las interfaces mediante tacto o teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
- *Location Manager*, posibilita a las aplicaciones la obtención de información de localización y posicionamiento, y funcionar según ésta.
- *Notification Manager*, mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión *Wi-Fi* disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en *Android* denominada *Intent*, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple *click*.
- *XMPP Service*, colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.

El último nivel del diseño arquitectónico de *Android* son las aplicaciones. Este nivel incluye tanto las incluidas por defecto de *Android* como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

3.2.2 La Máquina Virtual *Dalvik*

En *Android*, todas las aplicaciones se programan en el lenguaje *Java* y se ejecutan mediante una máquina virtual de nombre *Dalvik*, específicamente diseñada para *Android*. Esta máquina virtual ha sido optimizada y adaptada a las peculiaridades propias de los dispositivos

móviles (menor capacidad de proceso, baja memoria, alimentación por batería, etc.) y trabaja con archivos de extensión *.dex* (*Dalvik Executables*). *Dalvik* no trabaja directamente con el *bytecode* de *Java*, sino que lo transforma en un código más eficiente que el original, pensado para procesadores pequeños.

Gracias a la herramienta "*dx*", esta transformación es posible: los archivos *.class* de *Java* se compilan en archivos *.dex*, de forma que cada archivo *.dex* puede contener varias clases. Después, este resultado se comprime en un único archivo de extensión *.apk* (*Android Package*), que es el que se distribuirá en el dispositivo móvil. *Dalvik* permite varias instancias simultáneas de la máquina virtual, y a diferencia de otras máquinas virtuales, está basada en registros y no en pila, lo que implica que las instrucciones son más reducidas y el número de accesos a memoria es menor [7]. Así mismo, *Dalvik* no permite la compilación *Just-in-Time*.

3.2.3 Componentes de una Aplicación

Todas las aplicaciones en *Android* pueden descomponerse en cuatro tipos de bloques o componentes principales. Cada aplicación será una combinación de uno o más de estos componentes, que deberán ser declarados de forma explícita en un archivo con formato XML denominado "*AndroidManifest.xml*", junto a otros datos asociados como valores globales, clases que implementa, datos que puede manejar, permisos, etc. Este archivo es básico en cualquier aplicación en *Android* y permite al sistema desplegar y ejecutar correctamente la aplicación.

A continuación se exponen los cuatro tipos de componentes en los que puede dividirse una aplicación para *Android*.

Activity

Sin duda es el componente más habitual de las aplicaciones para *Android*. Un componente *Activity* refleja una determinada actividad llevada a cabo por una aplicación, y que lleva asociada típicamente una ventana o interfaz de usuario; es importante señalar que no contempla únicamente el aspecto gráfico, sino que éste forma parte del componente *Activity* a través de vistas representadas por clases como *View* y sus derivadas. Este componente se implementa mediante la clase de mismo nombre *Activity*.

La mayoría de las aplicaciones permiten la ejecución de varias acciones a través de la existencia de una o más pantallas. Por ejemplo, piénsese en una aplicación de mensajes de texto. En ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una

segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra tercera puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas deberías estar representada a través de un componente *Activity*, de forma que navegar de una ventana a otra implica lanzar una actividad o dormir otra. *Android* permite controlar por completo el ciclo de vida de los componentes *Activity*.

Muy vinculado a este componente se encuentran los *Intents*. Un *Intent* consiste básicamente en la voluntad de realizar alguna acción, generalmente asociada a unos datos. Lanzando un *Intent*, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar qué aplicación entre las instaladas es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una URL (*Universal Resource Location* - Localizador Universal de Recursos) en algún navegador web, o escribir un correo electrónico desde algún cliente de correo.

Broadcast Intent Receiver

Un componente *Broadcast Intent Receiver* se utiliza para lanzar alguna ejecución dentro de la aplicación actual cuando un determinado evento se produzca (generalmente, abrir un componente *Activity*). Por ejemplo, una llamada entrante o un SMS (*Short Message Service* - Servicio de Mensajes Cortos) recibido. No tiene interfaz de usuario asociada, pero puede utilizar el API *Notification Manager*, mencionada anteriormente, para avisar al usuario del evento producido a través de la barra de estado del dispositivo móvil.

Este componente se implementa a través de una clase de nombre *Broadcast Receiver*. Para que *Broadcast Intent Receiver* funcione, no es necesario que la aplicación en cuestión sea la aplicación activa en el momento de producirse el evento. El sistema lanzará la aplicación si es necesario cuando el evento monitorizado tenga lugar.

Service

Un componente *Service* representa una aplicación ejecutada sin interfaz de usuario, y que generalmente tiene lugar en segundo plano mientras otras aplicaciones (éstas con interfaz) son las que están activas en la pantalla del dispositivo.

Un ejemplo típico de este componente es un reproductor de música. La interfaz del reproductor muestra al usuario las distintas canciones disponibles, así como los típicos botones de reproducción, pausa, volumen, etc. En el momento en el que el usuario reproduce una canción, ésta se escucha mientras se siguen visionando todas las acciones anteriores, e incluso puede ejecutar una aplicación distinta sin que la música deje de sonar. La interfaz de usuario del reproductor sería un componente *Activity*, pero la música en reproducción sería un

componente *Service*, porque se ejecuta en *background*. Este elemento está implementado por la clase de mismo nombre *Service*.

Content Provider

Con el componente *Content Provider*, cualquier aplicación en *Android* puede almacenar datos en un archivo, en una base de datos *SQLite* o en cualquier otro formato que considere. Además, estos datos pueden ser compartidos entre distintas aplicaciones. Una clase que implemente el componente *Content Provider* contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación.

Existe una colección de clases para distintos tipos de gestión de datos en el paquete *android.provider*. Además, cualquier formato adicional que se quiera implementar deberá pertenecer a este paquete y seguir sus estándares de funcionamiento.

3.2.4 Ciclo de Vida de las Aplicaciones *Android*

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

El usuario desconoce este comportamiento de *Android*. Simplemente es consciente de que mediante un simple *click* pasa de una a otra aplicación y puede volver a cualquiera de ellas en el momento que lo desee. No debe preocuparse sobre cuál es la aplicación que realmente está activa, cuánta memoria está consumiendo, ni si existen o no recursos suficientes para abrir una aplicación adicional. Todo eso son tareas propias del Sistema Operativo.

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

Cada uno de los componentes básicos de *Android* tiene un ciclo de vida bien definido; esto implica que el desarrollador puede controlar en cada momento en qué estado se

encuentra dicho componente, pudiendo así programar las acciones que mejor convengan. El componente *Activity*, probablemente el más importante, tiene un ciclo de vida como el mostrado en la figura 3.2.

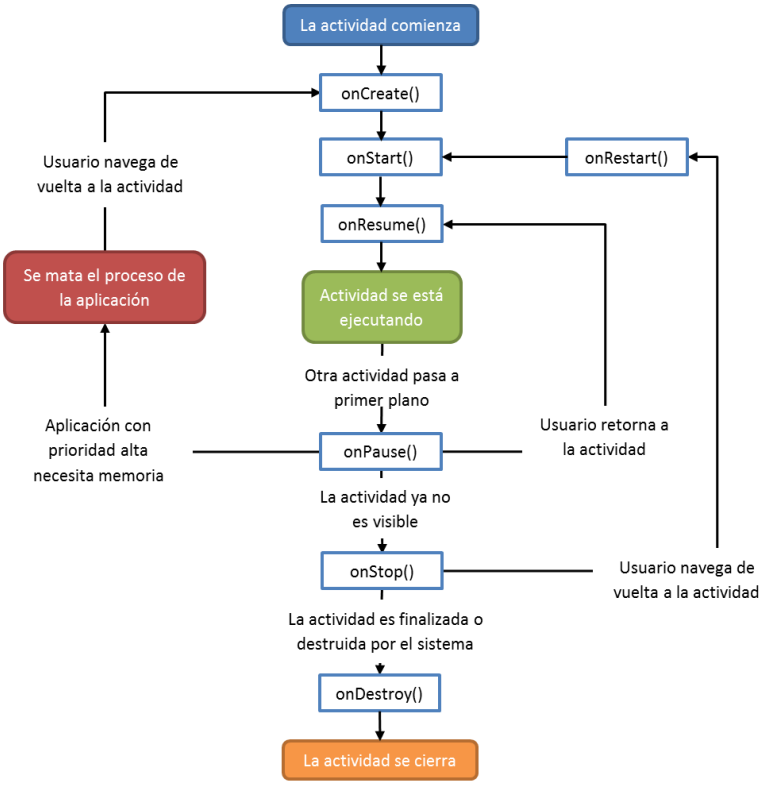

Figura 3.2: Ciclo de vida de un objeto *Activity.*

Se observa en la figura 3.2 que:

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

Tal y como se ve en el diagrama de la figura 3.2, el proceso que mantiene a esta *Activity* puede ser eliminado cuando se encuentra en *onPause()* o en *onStop()*, es decir, cuando no tiene

el foco de la aplicación. *Android* nunca elimina procesos con los que el usuario está interactuando en ese momento. Una vez se elimina el proceso, el usuario desconoce dicha situación y puede incluso volver atrás y querer usarlo de nuevo.

Entonces el proceso se restaura gracias a una copia y vuelve a estar activo como si no hubiera sido eliminado. Además, la *Activity* puede haber estado en segundo plano, invisible, y entonces es despertada pasando por el estado *onRestart()*.

Pero, ¿qué ocurre en realidad cuando no existen recursos suficientes?, obviamente, los recursos son siempre limitados, más aún cuando se está hablando de dispositivos móviles. En el momento en el que *Android* detecta que no hay los recursos necesarios para poder lanzar una nueva aplicación, analiza los procesos existentes en ese momento y elimina los procesos que sean menos prioritarios para poder liberar sus recursos.

Cuando el usuario regresa a una actividad que está dormida, el sistema simplemente la despierta. En este caso, no es necesario recuperar el estado guardado porque el proceso todavía existe y mantiene el mismo estado. Sin embargo, cuando el usuario quiere regresar a una aplicación cuyo proceso ya no existe porque se necesitaba liberar sus recursos, *Android* lo crea de nuevo y utiliza el estado previamente guardado para poder restaurar una copia fresca del mismo. Como se ha explicado, el usuario no percibe esta situación ni conoce si el proceso ha sido eliminado o está dormido.

3.2.5 Seguridad en *Android*

En *Android* cada aplicación se ejecuta en su propio proceso. La mayoría de las medidas de seguridad entre el sistema y las aplicaciones deriva de los estándares de Linux 2.6, cuyo *kernel*, recuérdese, constituye el núcleo principal de *Android*.

Cada proceso en *Android* constituye lo que se llama un cajón de arena o *sandbox*, que proporciona un entorno seguro de ejecución. Por defecto, ninguna aplicación tiene permiso para realizar ninguna operación o comportamiento que pueda impactar negativamente en la ejecución de otras aplicaciones o del sistema mismo. Por ejemplo, acciones como leer o escribir archivos privados del usuario (contactos, teléfonos, etc.), leer o escribir archivos de otras aplicaciones, acceso de red, habilitación de algún recurso *hardware* del dispositivo, etc., no están permitidas. La única forma de poder saltar estas restricciones impuestas por *Android*, es mediante la declaración explícita de un permiso que autorice a llevar a cabo una determinada acción habitualmente prohibida.

Además, en *Android* es obligatorio que cada aplicación esté firmada digitalmente mediante un certificado, cuya clave privada sea la del desarrollador de dicha aplicación. No es necesario vincular a una autoridad de certificado, el único cometido del certificado es crear una relación de confianza entre las aplicaciones. Mediante la firma, la aplicación lleva adjunta su autoría. Para establecer un permiso para una aplicación, es necesario declarar en el manifiesto uno o más elementos <*uses-permission*> donde se especifica el tipo de permiso que se desea habilitar.

Capítulo 4
Metodología y Herramientas

En este capítulo se expondrá la metodología utilizada para planificar y controlar el proceso de elaboración de la aplicación. Además, Se mostrarán y explicarán las diferentes herramientas vinculadas al desarrollo e implementación (tanto en el cliente como en el servidor) de los distintos componentes del sistema realizado.

4.1 Metodología

Para estructurar, planear y controlar el proceso de desarrollo de la aplicación fue utilizada una metodología de Prototipado [8]. Esta se caracteriza por presentar un método de desarrollo iterativo, en el cual se trabaja de cerca con el usuario, diseñando un sistema funcional primario que se aproxime al deseado. Dicho sistema es mostrado al usuario y seguidamente se continúa la elaboración del prototipo basado en las opiniones y recomendaciones dadas por el usuario. La iteración continúa hasta que el sistema se encuentra suficientemente completo tanto para el desarrollador como para el cliente. En esta etapa se afinan los pequeños detalles que hayan quedado pendientes y se entrega el sistema como un producto final [9].

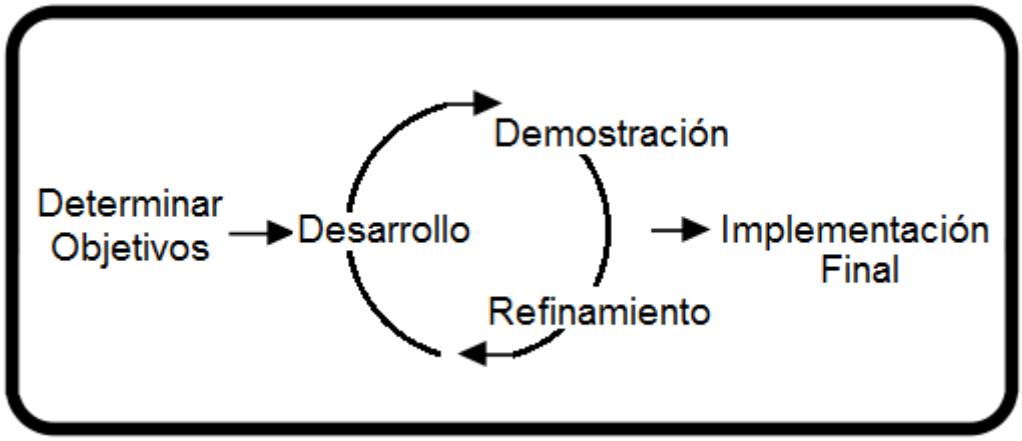

Figura 4.1: Metodología de prototipado.

La metodología de Prototipado permite capturar cada uno de los objetivos que plantea el proyecto de forma concreta, diseñando, desarrollando y corrigiendo cada una de sus fases

de forma puntual, obteniendo además una retroalimentación continua con el cliente o usuario de la aplicación.

Dicha metodología, fue integrada con un paradigma de programación Orientado a Objetos, con el cual, haciendo uso de conceptos como clases, herencia, objetos, métodos y atributos, entre otros, se alcanza una mayor organización, estructuración y reutilización del código.

4.2 Herramientas utilizadas

Un conjunto de programas, librerías y entornos de desarrollo fueron integrados y utilizados tanto para la programación del proyecto como para sus pruebas. A continuación se muestran las principales herramientas utilizadas:

- *Android Developer Tool Kit* (Conjunto de Herramientas de Desarrollo de *Android*) [10], compuesto por distintos componentes que se agregan al entorno de desarrollo integrado (*IDE - Intregated Development Environment*) *Eclipse* y que proporcionan un ambiente de programación para la creación de aplicaciones de *Android*, ofreciendo funciones que agilizan la construcción, prueba, depuración y empaquetamiento de las mismas. Específicamente, la versión de *Android Development Tool Kit* utilizada incluye las siguientes herramientas de desarrollo:
    - Entorno de Desarrollo Integrado *Eclipse* versión 3.0.8.
    - *Plugin Android Developer Tool* versión 21.0.0.
    - *Android SDK Tools* versión 21.0.0
    - Android Platforms-tools versión 16.
    - *Android Platform* API 17.
    - Imágenes del Sistema Operativo *Android* de las versiones utilizadas (2.2, 4.1.2 y 4.2.2) para el emulador.
- Entorno de Desarrollo Integrado *Sublime* versión 2.0.
- API de *Javascript* para *Google Maps* versión 3.
- Librería de *Javascript Jquery* versión 1.9.1.

- Microinstancia *(T1.micro)* de los servicios *Amazon* EC2 (*Amazon Elastic Compute Cloud* - Nube de Cómputo Elástico de *Amazon*). Estos ofrecen un conjunto completo de infraestructura y servicios para aplicaciones que permiten su ejecución remota. La instancia utilizada posee una cantidad de recursos limitada que dependen de la disponibilidad de los mismos. En principio, cada microinstancia de *Amazon* EC2 posee alrededor de 600Mb de memoria, un máximo de 2 unidades informáticas EC2, 8GB de almacenamiento EBS (*Elastic Block Store* - Bloque elástico de almacenamiento) el cual funciona como un Disco Duro y Sistema Operativo Ubuntu 12.04.02 lts.
- Servidor Web *Apache* versión 2.2.22.
- *PHP* 5.3.10.
- Base de Datos *MongoDB* versión 2.2.3.
- Librería *ActiveMongo* para conexión y manejo con base de datos.
- Librería *PHPExcel* para exportación de data en formato .xlsx.

Los equipos usados fueron dos Computadoras personales convencionales con las siguientes características: Procesador *core* i5, 6/8GB de memoria RAM, Disco Duro de 500GB y Sistema Operativo *Windows 7* de 64bits.

Adicionalmente, se utilizó GIT como controlador de versiones del proyecto, GITHUB para el almacenamiento y manejo de repositorios remotos y SSH para el acceso remoto al servidor de *Amazon Web Services*.

Capítulo 5
Diseño e Implementación

En este capítulo se explicará el proceso de desarrollo y diseño de la aplicación y de cada uno de los componentes que la conforman, tanto del lado del cliente como del servidor. Se expondrán los requerimientos capturados, se analizarán los diagramas más importantes, los detalles relevantes y además se explicará la implementación de la solución, haciendo énfasis en las clases y métodos más significativos.

5.1 Requerimientos del Sistema

El requerimiento general del sistema plantea la creación de una aplicación para teléfonos inteligentes, enfocada a capturar datos de ubicación geográfica (latitud, longitud, altitud, entre otros) utilizando el GPS proporcionado por el dispositivo móvil. Además, se propone la construcción de un programa en el servidor que permita recibir los datos capturados y almacenarlos para su posterior visualización y estudio. A partir de esto se obtienen los requerimientos funcionales y no funcionales del sistema.

5.1.1 Requerimientos funcionales

- La aplicación móvil permitirá la obtención o captura de datos de ubicación geográfica. Los datos mínimos necesarios son latitud, longitud y altitud.
- El dispositivo móvil enviará los datos de ubicación geográfica de manera automática al servidor, cada vez que se capturen cierta cantidad de datos establecidos.
- El programa del servidor recibirá los datos de ubicación geográfica enviados por la aplicación y los almacenará en la base de datos.
- El programa del servidor ofrecerá una página Web que permita la visualización en un mapa de los datos de ubicación geográfica capturados y su descarga en formato de hojas de Microsoft Excel.

5.1.2 Requerimientos no funcionales

- Un usuario sin experiencia será capaz de usar la aplicación móvil, por tanto deberá ser fácil e intuitiva.

- Tras un envió fallido de los datos de ubicación geográfica al servidor, estos serán enviados para el próximo envío.
- El teléfono inteligente debe contar con un sistema de posicionamiento global GPS a fin de obtener mayor exactitud en la captura de datos.

5.2 Diseño de la Solución

En base a los requerimientos mencionados en la sección anterior, se considera como punto de inicio de la aplicación la captura de datos de ubicación geográfica. Para este objetivo se plantea el uso de un teléfono inteligente con Sistema Operativo *Android*, debido a su amplia documentación, su licencia de *software* libre y su fácil integración con los diferentes servicios de localización (GPS, redes celulares y redes inalámbricas de servicio de datos).

El segundo punto a desarrollar es la facilidad de uso de la aplicación, la cual debe ser sencilla e intuitiva debido a que podría ser usada por personas poco o nada familiarizadas con la manipulación de teléfonos inteligentes. Para esto se plantea un alto nivel de automatización del sistema, ofreciendo al usuario solo la posibilidad de iniciar y detener el programa, dejando el almacenamiento, envío y vaciado de los datos en manos de la aplicación, facilitando así su uso y disminuyendo al máximo la posibilidad de equivocaciones o errores por parte del factor humano.

En cuanto al componente móvil del sistema, también se tomó en cuenta que debido a la posibilidad de que se utilicen celulares inteligentes tanto de baja como de alta gama, se debía ofrecer la compatibilidad tanto con Sistemas Operativos *Android* antiguos (*Android* versión 2.2) como con los más actuales (*Android* versión 4.1), a fin de no limitar la aplicación solo a un grupo reducido de dispositivos.

Adicional a la aplicación móvil, el sistema debe ofrecer un servicio que permita recibir y almacenar todos los datos obtenidos, por tal motivo se plantea el uso de un programa del lado del servidor. En primera instancia se eligió el uso de una computadora personal con configuraciones especiales para que ejerciera tales funciones, sin embargo, luego de diferentes investigaciones, se encontró un servicio de la empresa *Amazon*, que ofrece una *microinstancia* gratuita de una nube de cómputo manejada por ellos conocida como EC2. Dicha instancia se encuentra virtualizada y posee la cantidad de recursos de almacenamiento y procesamiento suficientes para el sistema a implementar.

Para visualizar y acceder a los datos almacenados en el servidor la opción que se decidió fue la construcción de una página web abierta al público, mediante la cual los usuarios interesados podrán tener acceso tanto a los datos almacenados como a la cantidad de capturas realizadas y los dispositivos utilizados entre otros aspectos.

La página Web ofrecerá dos maneras de tener acceso a los datos recolectados, la primera de ellas es mediante un api de *Google Maps*, el cual permite graficar sobre un mapa cada punto de ubicación geográfica obtenido para cada una de las capturas disponibles, de esta forma el usuario podrá obtener una imagen rápida de la cantidad de puntos, la zona de la captura, entre otros datos significativos. La segunda, es descargar un archivo con formato "*xls*" (Microsoft Excel) que poseerá la data también separada por los recorridos realizados, a fin de mantener cierto orden y no sobrecargar a un solo archivo con todos los datos.

La integración de todos los componentes mencionados anteriormente (la aplicación móvil, el programa en el servidor y la página web), unida a los diferentes métodos para la obtención, visualización y descarga de los datos, son los que conforman el sistema en el cual se centra este Trabajo Especial de Grado.

5.3 Diagrama de Casos de Uso

Una vez descrito el diseño de la solución, se procede al desarrollo del diagrama de casos de uso, el cual sirve para mostrar las funciones de un sistema desde el punto de vista de sus interacciones con el exterior sin entrar ni en la descripción detallada ni en la implementación de sus componentes [11].

5.3.1 Caso de uso de nivel 0

En este nivel se modela el sistema de forma general, abarcando tanto la aplicación móvil como la página web, con sus respectivos actores y el rol que cada uno desempeña (ver figura 5.1).

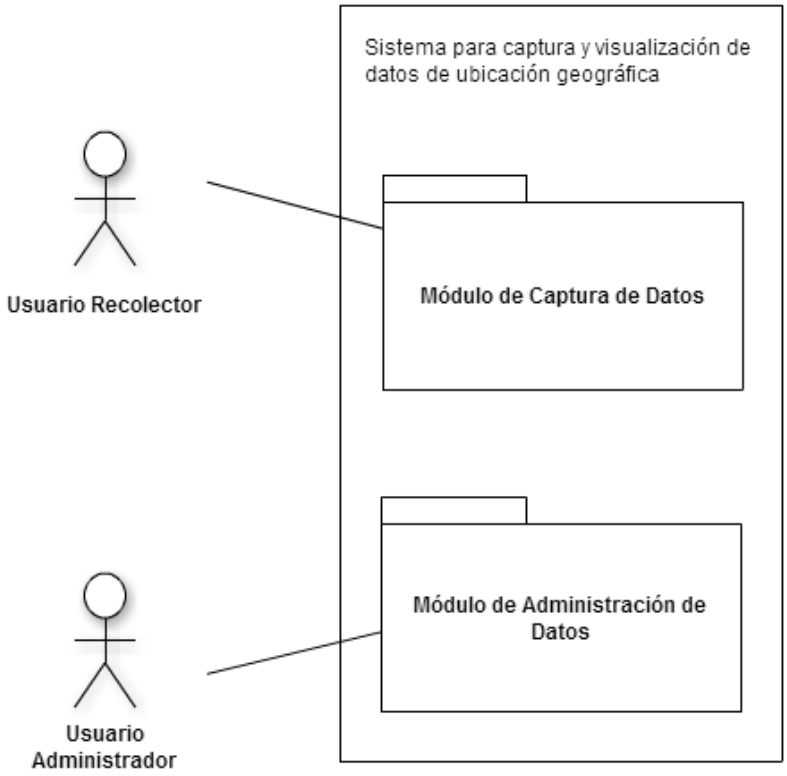

Figura 5.1: Casos de uso Nivel 0.

Usuario Recolector:	Representado por el usuario encargado de activar o detener la aplicación para la captura de datos de ubicación geográfica.
Usuario Administrador:	Actor que representa al personaje que visualiza y descarga los datos capturados por la aplicación.

Tabla 5.1: Usuarios del Sistema.

5.3.2 Caso de uso de nivel 1

En este nivel se refleja con un mayor grado de detalle la interacción de cada usuario con el componente de la aplicación que respectivamente manipula (ver figuras 5.2 y 5.3).

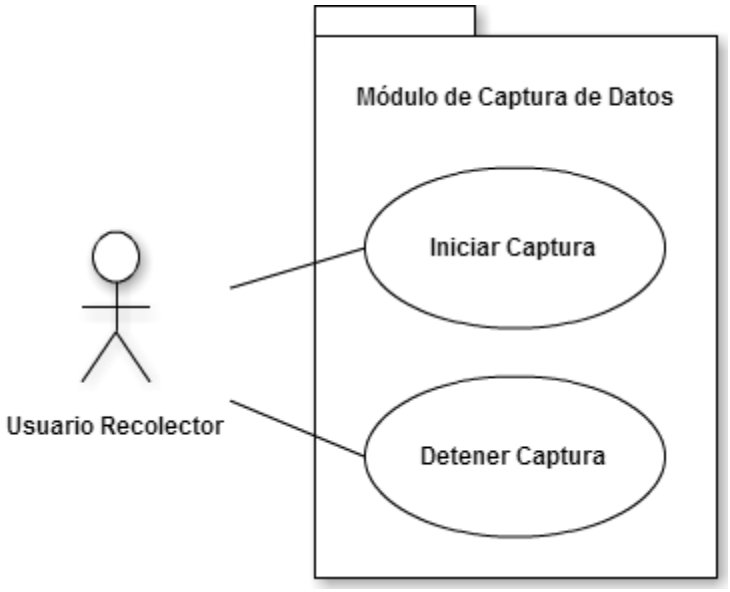

Figura 5.2: Casos de uso Nivel 1, Módulo de captura de datos.

Caso de uso:	Iniciar Captura.
Actor:	Usuario recolector.
Descripción:	Da inicio al funcionamiento de la aplicación para que comience la captura de los datos de ubicación geográfica.
Flujo Básico:	**a)** El usuario pulsa el botón iniciar para comenzar la captura de datos. **b)** Se crea el servicio que se mantendrá a la escucha de cualquier cambio de posición en el dispositivo móvil.
Pre condiciones:	El servicio de GPS debe estar activado en el teléfono inteligente.
Post condiciones:	Se pueden capturar cambios en el posicionamiento del teléfono inteligente.

Tabla 5.2: Caso de Uso Iniciar Captura.

Caso de uso:	Detener Captura.
Actor:	Usuario recolector.
Descripción:	Detiene el funcionamiento de la aplicación, parando la captura de datos al cambiar la posición del teléfono inteligente.
Flujo Básico:	**a)** El usuario pulsa el botón detener, finalizando la captura de datos. **b)** Se destruye el servicio de escucha en el cambio de posición del dispositivo móvil.
Pre condiciones:	La aplicación debe estar iniciada.
Post condiciones:	Se finaliza el servicio de captura de datos en el teléfono inteligente.

Tabla 5.3: Caso de Uso Detener Captura.

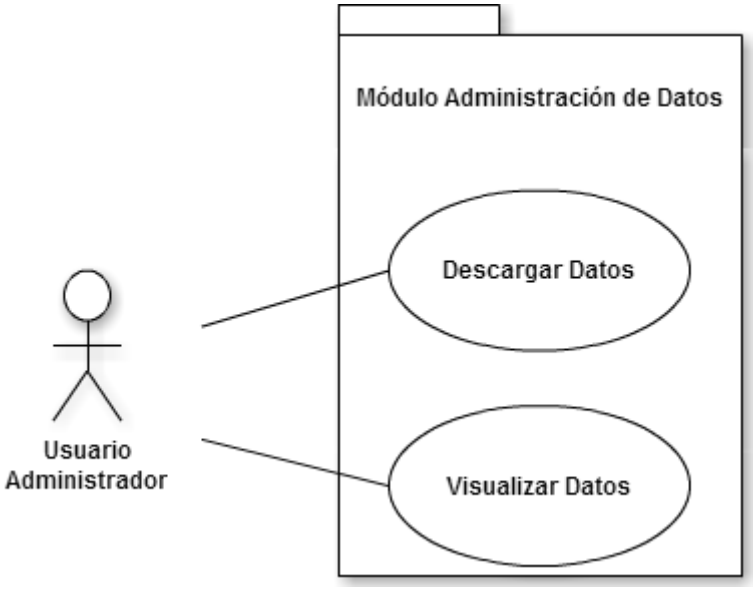

Figura 5.3: Casos de uso Nivel 1, Módulo de Administración de Datos.

Caso de uso:	Descargar Datos.

Actor:	Usuario administrador.
Descripción:	Inicia la descarga de los datos recolectados por la aplicación.
Flujo Básico:	**a)** El usuario accede a la página web de la aplicación y entra en la sección descarga. **b)** Elige el dispositivo móvil del cual quiere descargar los datos. **c)** Selecciona el recorrido realizado por el dispositivo a descargar. **d)** Ubica el rango de datos que quiere descargar. **e)** Presiona el botón descargar.
Pre condiciones:	Deben existir datos para ser descargados.
Post condiciones:	Se obtiene un documento Excel con los datos seleccionados a descargar.

Tabla 5.4: Caso de Uso Descargar Datos.

Caso de uso:	Visualizar Datos.
Actor:	Usuario administrador.
Descripción:	Se visualizan y grafican los puntos capturado por la aplicación en un mapa.
Flujo Básico:	**a)** El usuario accede a la página web de la aplicación y entra en la sección graficar data. **b)** Selecciona el dispositivo móvil del cual quiere graficar los datos. **c)** Especifica el recorrido que desea graficar. **d)** Ubica el rango de datos que quiere graficar. **e)** Presiona el botón graficar.
Pre condiciones:	Deben existir datos para ser graficados.
Post condiciones:	Se obtiene un mapa mostrando los puntos de ubicación geográfica del recorrido seleccionado.

Tabla 5.5: Caso de Uso Visualizar Datos.

5.4 Diagrama de Clases de Aplicación Móvil

La figura 5.4 muestra el diagrama de clases de la aplicación móvil con los métodos y atributos asociados que componen dicho sistema.

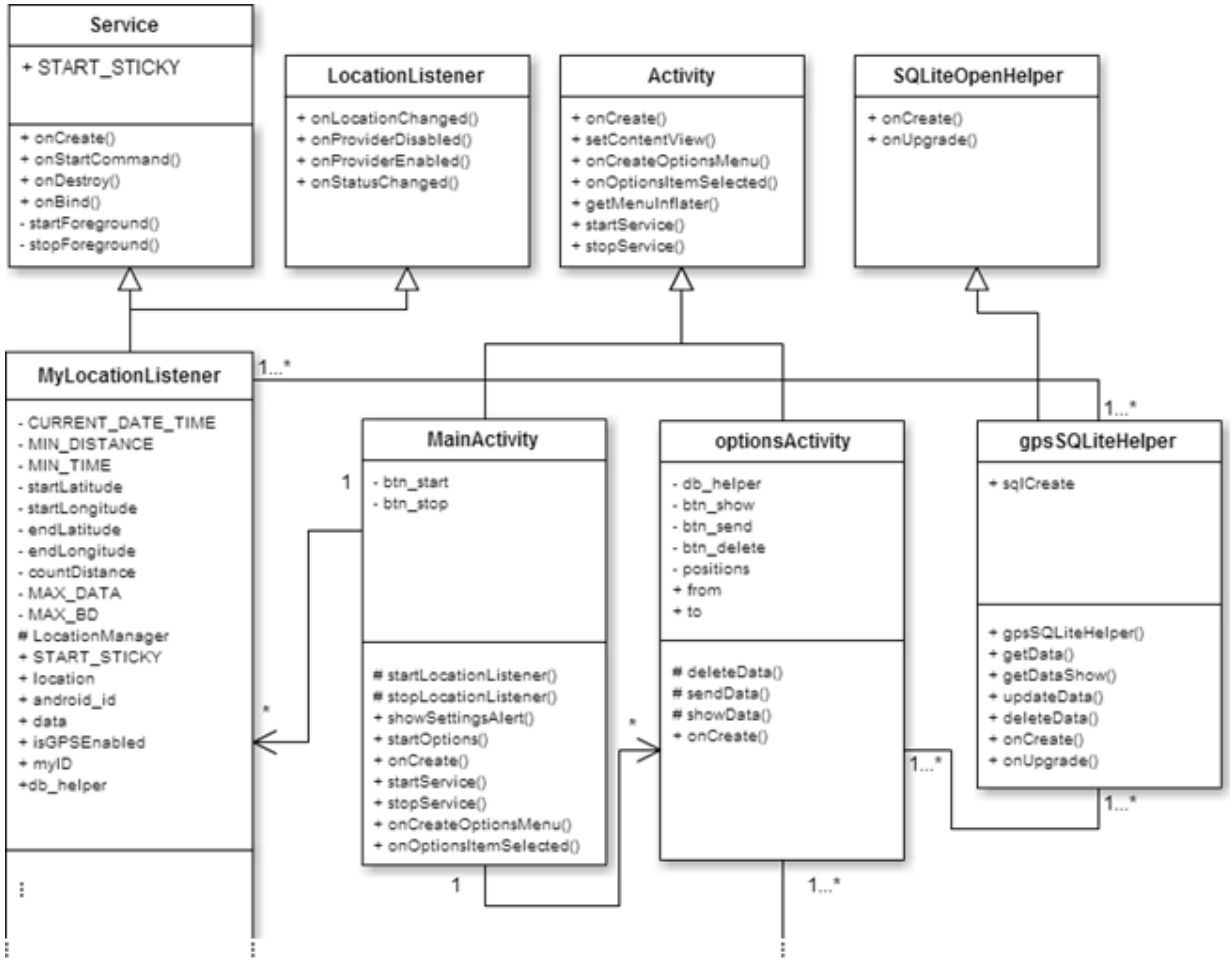

Figura 5.4: Diagrama de clases de la aplicación móvil (parte I).

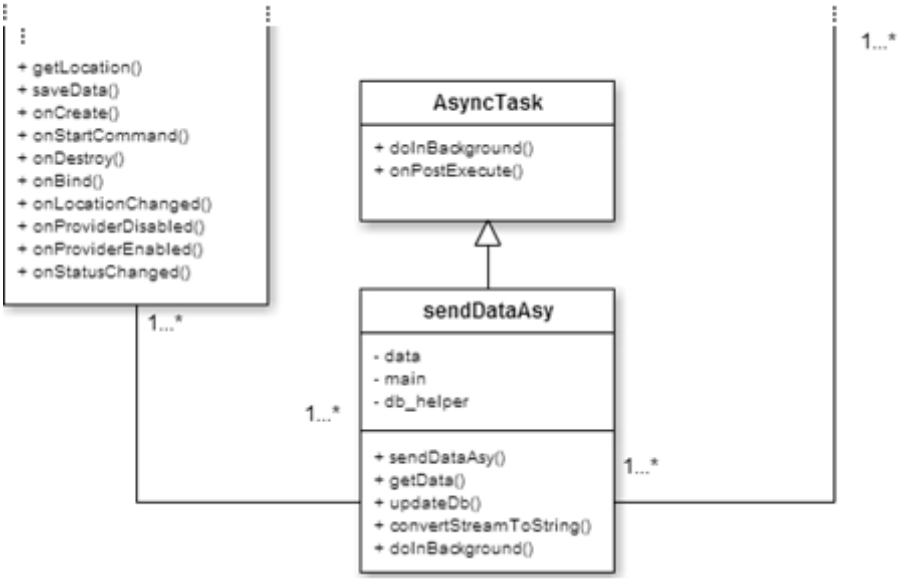

Figura 5.5: Diagrama de clases de la aplicación móvil (Parte II).

5.4.1 Clases Implementadas por el SDK de *Android*

Android provee en su SDK un conjunto de clases que facilitan el desarrollo de aplicaciones para su plataforma, a continuación se explicarán las que fueron utilizadas para la construcción del sistema.

Clase *Activity*

Esta clase es un componente que proporciona una pantalla con las distintas actividades que el usuario puede hacer, como marcar el teléfono, tomar una foto, enviar un correo electrónico o ver un mapa, entre otras [13]. El método que siempre debe implementarse en las clases que hereden de ella es el *onCreate()*, dentro de este se deben inicializar los componentes esenciales y además es referenciado el *setContentView()*, el cual define el diseño de la interfaz de usuario de la actividad.

Clase *Service*

Un servicio es un elemento que puede realizar operaciones de larga duración en segundo plano y no proporciona una interfaz de usuario [14]. La clase *Service* crea la opción de hacer largas operaciones que persisten en ejecución incluso cuando el usuario cambie de

aplicación, sin mostrar ningún tipo de alerta o notificación. Los métodos necesarios que deben implementar las clases hijas son el *onCreate()*, el cual es llamado cuando el sistema crea por primera vez el servicio, *onStartCommand()*, si se quiere que el servicio corra indefinidamente, *onBind()*, para que el usuario se comunique con él y decida su terminación, y por último, *onDestroy()*, que elimina el servicio del sistema y que en la aplicación desarrollada es utilizado como uno de los activadores para enviar los datos al servidor.

Clase *LocationListener*

Esta clase es fundamental en el la captura de los datos, ya que brinda la posibilidad de recibir notificaciones de cambio de posición del dispositivo. En sus métodos más resaltantes encontramos *onLocationChanged()*, el cual debe ser implementado por sus clases hijas para hacer la manipulación o estudio de los datos obtenidos por el servicio de localización escogido.

Clase *SQLiteOpenHelper*

Clase de ayuda para gestionar la creación de bases de datos y gestión de versiones [15]. Esta clase facilita la manipulación de la base de datos ya que interactúa con el API encargado de la gestión de datos, abstrayéndonos de procesos de implementación. Existen algunos métodos que suelen ser sobrescritos por sus clases hijas, dentro de estos se encuentran el *onCreate()*, encargado de la creación de la base de datos y carga de datos (siempre que no haya sido creada previamente), *onUpgrade()* el cual se dispara automáticamente cuando es necesaria una actualización de la base de datos o una conversión de los datos.

Clase *AsynTask*

Clase que facilita el uso de hilos en la aplicación permitiendo realizar acciones en paralelo, implementando operaciones duraderas y que no bloqueen el hilo principal. Algunos de los métodos a desarrollar por sus clases hijas son *doInBackground()*, donde se desarrollará el código principal del hilo. También se encuentra *onPostExecute()*, método que será ejecutado después de finalizarse *doInBackground()*.

5.4.2 Clases Implementadas para la Aplicación

En la construcción de la aplicación fue necesario el desarrollo de un conjunto de clases que conforman la lógica que hace posible el correcto funcionamiento de la aplicación, estas clases heredan de las clases implementadas en el SDK de *Android* pero con características particulares y serán explicadas a continuación.

Clase *mainActivity*

Es la clase principal del proyecto, responsable de iniciar la ejecución y lógica de la aplicación. Permite iniciar la interfaz que es mostrada al usuario junto con sus botones. Además, comienza o detiene el servicio de escucha para la captura de datos a través de la ubicación GPS. Inicia las opciones del menú encargadas de lanzar la actividad para la administración (visualización, eliminación y envío) de los datos obtenidos (ver figura 5.4). Entre sus métodos se encuentran *startLocationListener()*, quien verifica las disponibilidad del GPS y dispara el servicio *startService()* para la escucha del cambio de posición, *stopLocationListener()*, *que* detiene la captura de cambios de posición usando *stopService()*; *showSettingsAlert()*, encargado de mostrar la alerta para activar las opciones de ubicación, *startOptions()*, usado para disparar las opciones de administración, *onCreate()*, encargado de la configuración e inicialización para el uso de la clase, los métodos *onCreateOptionsMenu()* y *onOptionsItemSelected()* usados para el despliegue del menú con la opción de administración.

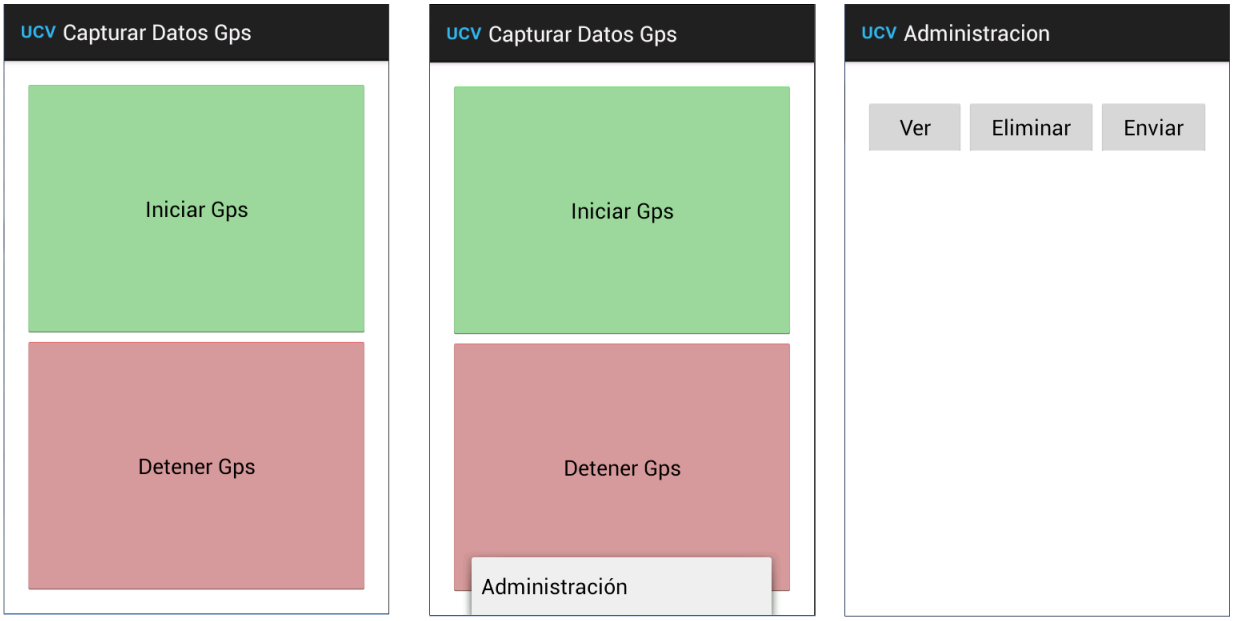

Figura 5.4: Interfaz de la aplicación.

Clase *optionsActivity*

Esta clase permite la visualización de las opciones de administración, ofreciendo la manipulación y visualización de la información almacenada. Entre sus métodos se encuentran *sendData()*, encargado de enviar los datos que fueron capturados y almacenados (o incluso los que quedaron rezagados) en la base de datos, *showData()*, muestra los datos almacenados en

la base de datos con la idea de verificar la información recogida por el sistema de ubicación, *deleteData()*, método desarrollado para la eliminación de los datos enviados al servidor.

Clase *myLocationListener*

Clase fundamental en la captura de datos, ya que es donde se desarrolla la lógica de escucha para detectar los cambios de posición realizados por el dispositivo, esta clase hereda de dos clases importantes, la clase *Service* y *LocationListener*, las cuales permiten que la aplicación se mantenga escuchando por un largo período (así la aplicación no esté siendo usada) los cambios de posición del dispositivo, permitiendo la captura y procesamiento de los datos de manera automática. Los principales métodos utilizados en esta clase son *getLocation()*, que inicializa la escucha del servicio de ubicación y hace la primera solicitud de posicionamiento basándose en el último lugar registrado por el dispositivo, *onLocationChanged()*, método donde se encuentra la lógica en la captura de datos, es llamado automáticamente cuando se detecta un cambio en la posición del dispositivo, en este punto son capturados los datos *latitude, longitude, altitude, time, accuracy, provider, direction y speed,* los cuales son almacenados en memoria y después de cierta cantidad enviados a la base de datos y posteriormente enviados al servidor, automatizando el proceso de captura y envío de información.

Clase *sendDataAsy*

Tiene la responsabilidad de hacer el envío al servidor de la información capturada y almacenada en la base de datos a través de la red disponible. Debido a que este proceso puede ser duradero, la clase *sendDataAsy* hereda de *AsynTask,* obteniendo las características propias de los hilos, logrando así ejecutarse de forma paralela como un hilo auxiliar sin bloquear el hilo principal de la aplicación. Entre los métodos se encuentran *getData()*, que obtiene de la base de datos la información que se envía al servidor, *updateDb()*,usado solamente cuando los datos son enviados y almacenados efectivamente en el servidor, actualizando la información local lista para ser eliminada, *doInBackground()*, método principal de la clase, en este se encuentra la lógica del hilo de ejecución quien hace la conexión con el servidor y realiza el envió de la información.

Clase *gpsSQLiteHelper*

Implementa los métodos para manipulación y conexión de la base de datos en *SQLite*. Entre sus métodos encontramos *getData()* y *getDataShow()*, ambos listan la información capturada, sólo que el primero lista la totalidad de la información, utilizada para almacenar en el servidor y el segundo lista la información mostrada en la aplicación para administración; *updateData()*, actualiza la información cambiando el campo enviado a verdadero para indicar

que dicha fila fue enviada al servidor, *deleteData()*, elimina los datos que tienen el indicador enviado en verdadero para no saturar la base de datos utilizada por la aplicación.

5.4.3 Lógica de Funcionamiento de la Aplicación.

Al momento de iniciar la aplicación es instanciada la clase *mainActivity* quien se encarga de orquestar el resto de las funcionalidades, en este momento hay tres caminos posibles, activar la opción del menú (Administración), iniciar la captura de los datos de posicionamiento o detener la captura.

Si se escoge iniciar captura de los datos, puede suceder que no esté activo el servicio de posicionamiento GPS, por lo que se da la opción al usuario que la active mediante una alerta, de encontrarse activo ya se inicia el servicio proporcionado por la clase *myLocationListener,* que activa una notificación en el barra de acciones del dispositivo avisando al usuario que se está a la escucha de cambios de posición del equipo. Cada vez que se captura una nueva posición esta es almacenada en memoria, luego de diez capturas estas son guardadas en la base de datos, al guardar dos veces en la base de datos esta información es enviada al servidor y eliminada de la base de datos local, automatizando el proceso de obtención y gestión de la información.

Si se elige detener la captura de datos y el servicio existe, entonces es destruido y toda la información capturada hasta ese momento es guardada en la base de datos, quedando en espera para su posterior envío al servidor.

Si es activada la opción del menú (Administración), se muestra una ventana con los diferentes botones para la gestión de los datos capturados, los cuales son *ver*, *listar* y *eliminar*. El primero de ellos permite listar información capturada, el segundo borrar los datos ya enviados al servidor, y el último*,* enviar datos rezagados al servidor.

5.4.4 Implementación Detallada de la Aplicación.

En la clase principal de la aplicación *mainActivity* se instancia la lógica para iniciar y detener los servicios de escucha en el cambio de posición del dispositivo, sin embargo, estas no se encuentran ubicadas ahí. En la figura 5.5 se mostrarán y explicarán detalladamente algunas de las principales secciones de código que se utilizan a partir de la implementación del *mainActivity*.

```
23      @Override
24      public void onCreate(Bundle savedInstanceState) {
25          //Llamado al constructor de la clase Activity
26          //para garantizar el buen funcionamiento
27          super.onCreate(savedInstanceState);
28          //Asignación del layout de la primera ventan
29          setContentView(R.layout.main);
30
31          btn_start = (Button) findViewById(R.id.btn_start);
32          btn_stop = (Button) findViewById(R.id.btn_detener);
33          btn_start.getBackground().setColorFilter(Color.GREEN,PorterDuff.Mode.MULTIPLY);
34          btn_stop.getBackground().setColorFilter(Color.RED,PorterDuff.Mode.MULTIPLY);
35          //Vinculación del boton btn_start para iniciar el servicio
36          btn_start.setOnClickListener(new View.OnClickListener() {
37              public void onClick(View v) {
38                  startLocationListener();
39              }
40          });
41          //Vinculación del boton btn_stop para detener el servicio
42          btn_stop.setOnClickListener(new View.OnClickListener() {
43              public void onClick(View v) {
44                  stopLocationListener();
45              }
46
47          });
48
49      }
```

Figura 5.5: Método *onCreate()* de la clase *mainActivity.*

Como se detalla en la figura 5.5, en el método *onCreate(),* heredado de la clase *Activity,* se vinculan los botones *btn_start* y *btn_stop* con el evento *click* para activar los métodos *startLocationListener()* o *stopLocationListener()* quienes mediante *startSer*vice() o *stopService()* activan o detienen el servicio de escucha (ver figura 5.6).

```
52    protected void startLocationListener() {
53
54        LocationManager locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
55        // Verificando el estado del GPS
56        Boolean isGPSEnabled = locationManager
57                .isProviderEnabled(LocationManager.GPS_PROVIDER);
58        if (!isGPSEnabled) {
59            // Mostrando las opciones para activar el GPS
60            showSettingsAlert();
61        } else {
62            //Iniciando el servicio de escucha proporcionado por MyLocationListener
63            startService(new Intent(MainActivity.this, MyLocationListener.class));
64            btn_start.setEnabled(false);
65        }
66
67    }
68
69    protected void stopLocationListener() {
70        //Deteniendo el servicio
71        stopService(new Intent(MainActivity.this, MyLocationListener.class));
72        btn_start.setEnabled(true);
73    }
```

Figura 5.6: Métodos *startLocationListener()* y *stopLocationListener()* de la clase *mainActivity*.

El servicio activado es provisto por la clase *myLocationListener* que provee las funciones de inicialización de las variables necesarias para empezar la captura de datos en el método *onCreate()* (ver figura 5.7).

```
47    @Override
48    public void onCreate() {
49        // Llamado al constructor de la clase Service
50        // para garantizar el buen funcionamiento
51        super.onCreate();
52        // Muestra mensaje al usuario de que fue exitosa la
53        // creación del servicio
54        Toast.makeText(this, "Servicio creado", Toast.LENGTH_SHORT).show();
55        android_id = Secure.getString(getBaseContext().getContentResolver(),
56                Secure.ANDROID_ID);
57
58        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy/HH:mm:ss");
59        // Marca de tiempo como indicador de captura y poder visualizarlo mejor
60        CURRENT_DATE_TIME = sdf.format(new Date());
61
62        Toast.makeText(this, "Tiempo de captura: " + CURRENT_DATE_TIME,
63                Toast.LENGTH_SHORT).show();
64        // Metodo para activar la escucha de cambios de
65        // de posición
66        getLocation();
67        // Manejador de conexión y operaciones de la base de datos
68        db_helper = new gpsSQLiteHelper(this, "DBgps", null, 1);
69    }
```

Figura 5.7: Método *onCreate()* de la clase *MyLocationListener.*

El método *getLocation()* es el encargado de preparar los servicios de ubicación para la escucha y la variable *db_helper,* encargada de realizar la conexión con la base de datos para realizar las operaciones , sin embargo es en *onStartCommand()* donde realmente se arranca el servicio en segundo plano permitiendo la captura constante de nuevas posiciones (ver figura 5.8).

```
271     @Override
272     public int onStartCommand(Intent intenc, int flags, int idArranque) {
273         Toast.makeText(this, "Servicio arrancado " + idArranque,
274                 Toast.LENGTH_SHORT).show();
275         intenc.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
276                 | Intent.FLAG_ACTIVITY_SINGLE_TOP);
277         PendingIntent pendIntent = PendingIntent
278                 .getActivity(this, 0, intenc, 0);
279 
280         //Constructor de la notificación para avisar en el barra de acción
281         //que se esta capturando posiciones
282         Notification notice = new Notification(R.drawable.ic_stat_ucv,
283                 "Iniciada Captura de Coordenadas", System.currentTimeMillis());
284 
285         //Configurando la notificación
286         notice.setLatestEventInfo(this, "Iniciado", "Capturando datos",
287                 pendIntent);
288 
289         notice.flags |= Notification.FLAG_NO_CLEAR;
290         //Iniciando el servicio en segundo plano para que escuche asi no se use
291         //la aplicación
292         startForeground(myID, notice);
293 
294         return START_STICKY;
295     }
```

Figura 5.8: Método *onStartCommand()* de la clase *myLocationListener*.

Para llevar el proceso de captura y envío de manera automática cada dos veces que se guarda en la base de datos se envía al servidor, este procedimiento se encuentra implementado en el método *saveData()* de la clase *myLocationListener* (ver figura 5.9).

```
215         MAX_BD++;
216         if (MAX_BD >= 2) {
217             MAX_BD = 0;
218             try {
219                 Toast.makeText(this, "Enviando al Servidor", Toast.LENGTH_SHORT)
220                         .show();
221                 //Ruta a donde enviar la informacion
222                 String path = "http://tesis.codesign.me/server.php";
223                 //Creación del hilo que realizara el envio
224                 new sendDataAsy(this).execute(path);
225                 data = new JSONObject();
226             } catch (Exception e) {
227                 e.printStackTrace();
228             }
229         }
```

Figura 5.9: Verificación para el envío al servidor de los datos, en el método *saveData()* de la clase *myLocationListener.*

5.5 Diagrama de Clases de la Página Web

En esta sección se mostrarán las clases implementadas relevantes para el desarrollo de la página web que permite la visualización o descarga de los datos capturados y almacenados en el servidor (ver figura 5.10).

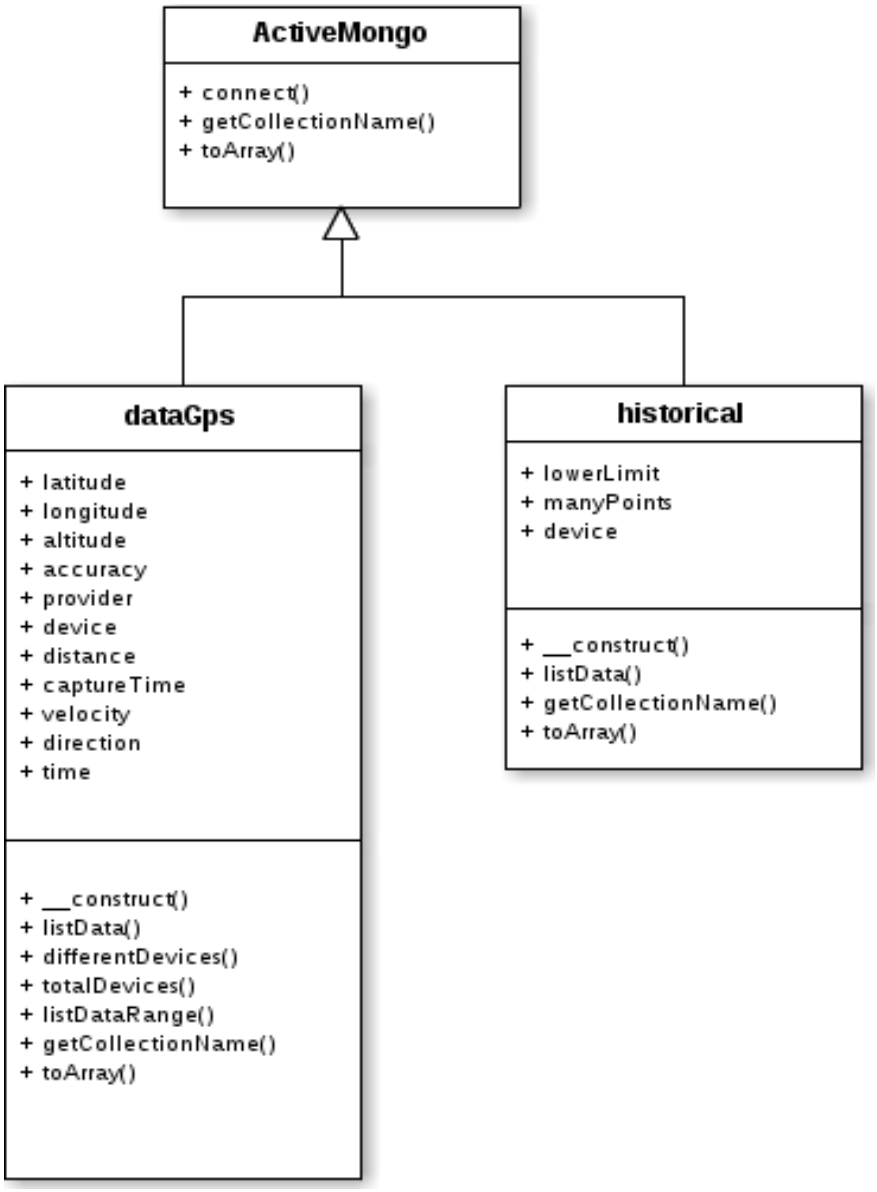

Figura 5.10: Diagrama de clases de página Web.

Clase *ActiveMongo*

Encargada de la implementación de los métodos generales de conexión, búsqueda, inserción, actualización y eliminación de información en la base de datos MongoDB, también transforma o define colecciones (unidad de almacenamiento en MongoDB) en clases, haciendo eficiente y fácil la interacción contra la Base de Datos.

Clase *dataGps*

Encargada de manejar la información guardada en el servidor, la cabecera de la clase *ActiveMongo* facilita la conexión y gestión de datos contra MongoDB. Entre sus métodos encontramos *listData()* y *listDataRange()*, usados para traer los datos según ciertos parámetros recibidos, *differentDevices(),* para ver los diferentes dispositivos con los cuales se capturaron los datos.

Clase *historical*

Clase implementada para llevar el control del historial de descarga de los datos y dar soporte a los usuarios. Mantiene un interfaz sumamente simple, a fin de mantener la sencillez y rapidez (ver figura 5.11).

Figura 5.11: Sección de descarga de los datos.

La realización de la página en esencia fue para el control, visualización y descarga de los datos capturados. El funcionamiento consiste en recibir los datos enviados por la aplicación móvil, guardarlos en la base de datos *NoSQL* (*mongoDB*) y según la funcionalidad requerida graficarlos en un mapa de *Google* (ver figura 5.12) o descargarlos.

Data para graficar --- Descargar DATA

Numero de puntos: 100

Indica el numero de marcas que aparecera en el mapa, **seran graficados los mas recientes capturados**

Inicio de los puntos: 0

Indica el a partir de que pagina de la base de datos se graficaran

Dispositivo: 8c707d8dbcaa1ed0 ----- Tiempo de captura: 09-05-2013/06:19:26

Total de puntos existentes: 980

Indica el telefono o equipo que capturo los puntos

Graficar

Figura 5.12: Sección de visualización de las posiciones.

Capítulo 6
Pruebas, Captura y Análisis de Datos

En este capítulo se analizarán las distintas pruebas, ajustes, configuración de parámetros y capturas de datos realizadas por la aplicación móvil desarrollada para este Trabajo Especial de Grado. Se explicarán los escenarios y ambientes donde las pruebas y recolección de datos fueron aplicadas, y finalmente, se mostrará un análisis de la data recolectada, a fin de representar el alcance de la aplicación en cuanto a precisión y calidad de la información obtenida en cada captura.

6.1 Pruebas de Funcionalidad

El objetivo de las pruebas funcionales es validar el comportamiento del sistema en relación a sus requerimientos y especificaciones. Estas son realizadas principalmente para detectar los posibles errores generados en la fase de programación [12]. Para la aplicación desarrollada fueron realizadas dos tipos de pruebas de funcionalidad, la primera de ellas realizada bajo un ambiente de pruebas virtual controlado, mientras que la segunda estuvo enfocada a un escenario real, realizando la captura a medida que se movilizaba la persona o vehículo con el dispositivo asignado.

6.1.1 Pruebas de Funcionalidad en Ambiente Virtual

El ambiente utilizado para este tipo de pruebas fue proporcionado por el *Android Virtual Device Manager* (manejador de Dispositivos Virtuales de *Android*), el cual contaba con dos configuraciones distintas, una que simulaba *Smartphones* con características de gama baja y otra que representaba *Smartphones* de gama alta (ver figura 6.1) . A continuación se muestran las principales especificaciones para cada uno de los ambientes de prueba virtuales:

- **Máquina Virtual de *Smartphones* de gama baja:**
  - Sistema Operativo *Android* 2.2.
  - 256 MB de Memoria RAM.
  - 128 MB de Memoria Interna.

- **Máquina Virtual de *Smartphones* de gama alta:**
  - Sistema Operativo *Android* 4.2.2.

- 1 GB de Memoria RAM.
- 4GB de Memoria Interna.

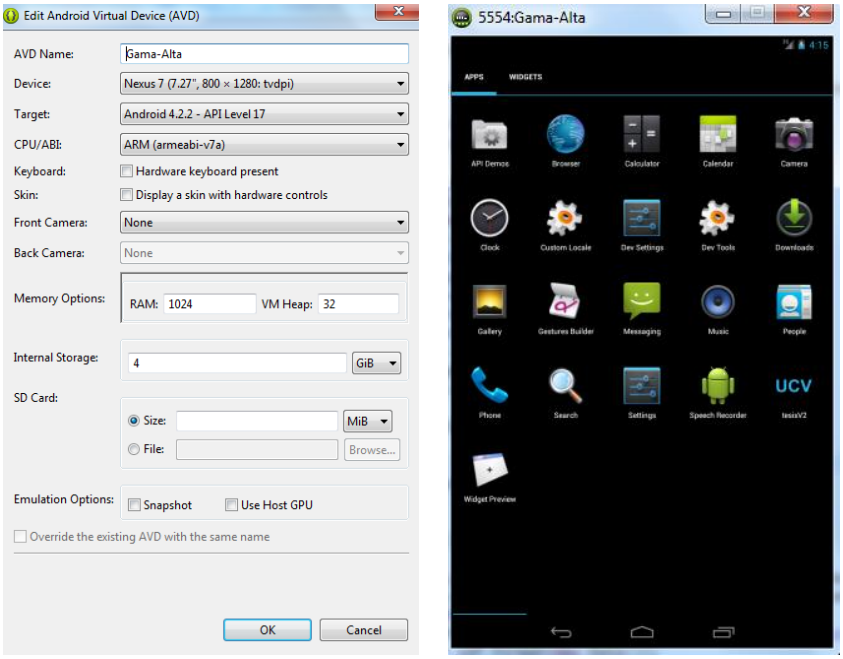

Figura 6.1: Características y pantalla de dispositivo virtual gama alta.

Una vez definidos los ambientes, se pudo dar inicio al ciclo de pruebas, el cual era realizado cada vez que se finalizaba algún funcionamiento significativo del sistema. En general las pruebas de funcionalidad realizadas fueron las siguientes:

- **Instalación:** Debido a que la aplicación se debía ejecutar en un dispositivo con Sistema Operativo *Android*, la primera prueba realizada era que efectivamente permitiera tanto su instalación como su ejecución. Para esta prueba la aplicación era empaquetada en un archivo de tipo *apk*, el cual era instalado y ejecutado desde la máquina virtual.

- **Inicio y finalización del servicio:** Para que el dispositivo permitiera mantener la aplicación corriendo en segundo plano el tiempo que durara la captura de datos, esta debía ser declarada y ejecutada como un servicio, por tanto una de las pruebas realizadas fue que este se iniciara y detuviera correctamente al presionar los botones de "Iniciar" y "Detener" respectivamente.

- **Captura de datos:** Esta prueba consistió principalmente en que todos los datos previamente definidos se capturaran de manera correcta. Los datos a verificar eran longitud, latitud, altitud, precisión, distancia, dirección, hora y fecha de la captura. Una información importante referente a esta prueba es que debido a que el dispositivo virtual se mantiene siempre en el mismo lugar sus coordenadas nunca cambian, por dicha razón, para esta prueba se realizó una configuración especial de la máquina virtual, en donde se suministran diferentes datos de ubicación que son actualizados aleatoriamente a fin de simular el desplazamiento del dispositivo.

- **Almacenamiento de los datos:** Luego de la captura correcta de cada uno de los datos necesarios para cada punto, estos deben ser almacenados en la memoria interna del dispositivo. Esta prueba consistió en la revisión periódica de los datos almacenados en la memoria del dispositivo, a fin de compararlos con los datos capturados y corroborar que se estaban almacenando de forma correcta.

- **Envío de datos al servidor:** Cada cierto tiempo los datos almacenados en el dispositivo eran enviados a un servidor central, que permitía su almacenamiento permanente. Para verificar que los datos se enviaban y recibían de forma correcta, luego de cada envío se comparaban los datos enviados con los datos recibidos por el servidor.

- **Eliminación de datos:** Después de que los datos eran enviados al servidor debían ser eliminados de la memoria del dispositivo a fin de evitar que se sobrecargara la memoria interna del mismo. Esta prueba consistía en revisar la memoria del dispositivo luego de que se realizara la eliminación de datos, a fin de asegurar que todos los datos eran realmente eliminados.

6.1.2 Pruebas de Funcionalidad en Ambiente Real

Este tipo de pruebas se basó principalmente en las mismas explicadas en la sección anterior (Instalación, inicio y finalización del servicio, captura, almacenamiento, envío y eliminación de datos), modificando básicamente el ambiente de prueba, el cual paso de ser un escenario virtual a real. Los diferentes dispositivos utilizados para estas pruebas se listan a continuación:

- *Huawei Ascend Y200*:
    - Procesador 800 MHz Cortex-A5.
    - 256 MB de Memoria RAM.

- 512 MB de Almacenamiento Interno.
- Sistema Operativo *Android* 2.3.

- *Samsung Galaxy S2 I777*:
  - Procesador *Dual-Core* 1.2 GHz *Cortex-A9*.
  - 1 GB de Memoria RAM.
  - 16 GB de Almacenamiento Interno.
  - Sistema Operativo *Android* 4.0.4.

- *Samsung Galaxy S3 I9300*:
  - Procesador *Quad-Core* 1.4 GHz *Cortex-A9*.
  - 1 GB de Memoria RAM.
  - 32 GB de Almacenamiento Interno.
  - Sistema Operativo *Android* 4.1.2.

- *Samsung Galaxy Tab 2*:
  - Procesador *Dual-Core* 1GHz.
  - 1GB de Memoria RAM.
  - 32GB de Almacenamiento Interno.
  - Sistema Operativo *Android* 4.0.3.

En esta ocasión, debido a que se realizaban capturas de datos reales, además de realizar las pruebas definidas, también se mantenía un constante monitoreo en el que se revisaba que los datos obtenidos realmente pertenecían o al menos se mantenían cercanos a las locaciones donde se realizaban las pruebas. En la mayoría de los casos dicha comparación era acertada, demostrando que la aplicación estaba funcionando de la forma esperada.

6.2 Pruebas de Ajuste y Configuración de Parámetros

Estas pruebas consistieron en el ajuste de parámetros como *tiempo mínimo de actualización de posición*, *distancia mínima de actualización de posición* y *proveedor de servicio*, de forma tal que se obtuviera un equilibrio entre la precisión, el intervalo de actualización de los puntos y el uso de recursos del *Smartphone* a utilizar.

6.2.1 Ambiente de Pruebas

Estas pruebas fueron realizadas en campo, en rutas próximas a las áreas de Santa Mónica, Bello Monte, Las Mercedes, Los Ruices, entre otras. Dichas pruebas mantenían tiempos de aproximadamente 30 - 50 minutos y, en la mayoría de los casos, se desempeñaban en horas donde la densidad vehicular era bastante elevada. Como medios de transporte fueron utilizados vehículos propios, aunque en algunas oportunidades las pruebas fueron realizadas a pie. Los principales parámetros que fueron configurados se listan a continuación:

- **Tiempo:** se refiere al tiempo mínimo (representado en microsegundos) que espera la aplicación para realizar una actualización de la posición.

- **Distancia:** se representa como la distancia mínima (representada en metros) que debe desplazarse dispositivo para registrar una actualización de la posición.

- **Proveedor de Servicio:** define el proveedor de servicio que se utilizará durante la captura de datos, puede ser *GPS*, el cual permite una mayor precisión pero genera un consumo mayor de la batería, o *NETWORK*, el cual ofrece menor precisión pero disminuye el consumo de energía.

Los equipos móviles utilizados para la captura de datos fueron los mismos de las pruebas funcionales [ver sección 6.1.2]. Estos varían en rendimiento y características, sin embargo el resultado obtenido de todos era muy similar, lo cual demuestra que la aplicación puede ser ejecutada en la mayoría de los dispositivos con Sistema Operativo *Android* 2.2 en adelante sin presentar mayores variaciones en cuanto a los datos capturados.

6.2.1 Pruebas realizadas

Los resultados de las pruebas son representados visualmente mediante la página web desarrollada para el proyecto, la cual permite la opción de graficar los puntos sobre la superficie de un mapa provisto por el API de *Google Maps* o descargar los datos obtenidos en un archivo con formato *.xlsx* (*Microsoft Excel*), en donde se encuentra detallada la información de cada punto obtenido.

A continuación se mostrarán los resultados más importantes de las pruebas realizadas, las cuales consistían en el cambio del valor de los parámetros recibidos por el procedimiento *locationManager.requestLocationUpdates(),* encargado de registrar la actividad actual que se

actualizará periódicamente dependiendo de tres parámetros principales, el proveedor de servicios, el tiempo mínimo intervalos de actualización de posición, representado en microsegundos, y por último, la distancia mínima entre actualizaciones de posición, representada en metros.

- **Proveedor NETWORK, tiempo mínimo 30000 milisegundos y distancia mínima de 50 metros:** La primera característica que se observó de esta captura de puntos, fue que el Proveedor NETWORK ofrece una precisión que se encuentra por encima de la esperada, mostrando puntos con una precisión de 30 a 50 metros, existiendo algunos puntos que superaban incluso los 100 metros de precisión. Adicionalmente, los puntos se mostraban muy alejados entre sí, lo que hacía muy frecuente la pérdida de información entre los diferentes tramos transitados. Por último, al depender de la red de datos y de la señal celular, el proveedor NETWORK presenta tramos donde no puede obtener data debido a que su servicio se encuentra inhabilitado (ver figura 6.2).

Figura 6.2: Prueba con proveedor NETWORK, tiempo 30000 ms. y distancia mín. de 50 mts.

- **Proveedor GPS, tiempo mínimo 30000 milisegundos y distancia mínima de 50 metros:** Esta prueba evidencia una mejora indiscutible en la precisión de la data obtenida, presentando una precisión que en pocas oportunidades supera los 15 metros y que incluso logra alcanzar en repetidas oportunidades los 5 metros. Sin embargo, los puntos se encuentran aún muy alejados entre sí, permitiendo que se pierda información que podría ser valiosa para su posterior manipulación (ver figura 6.3).

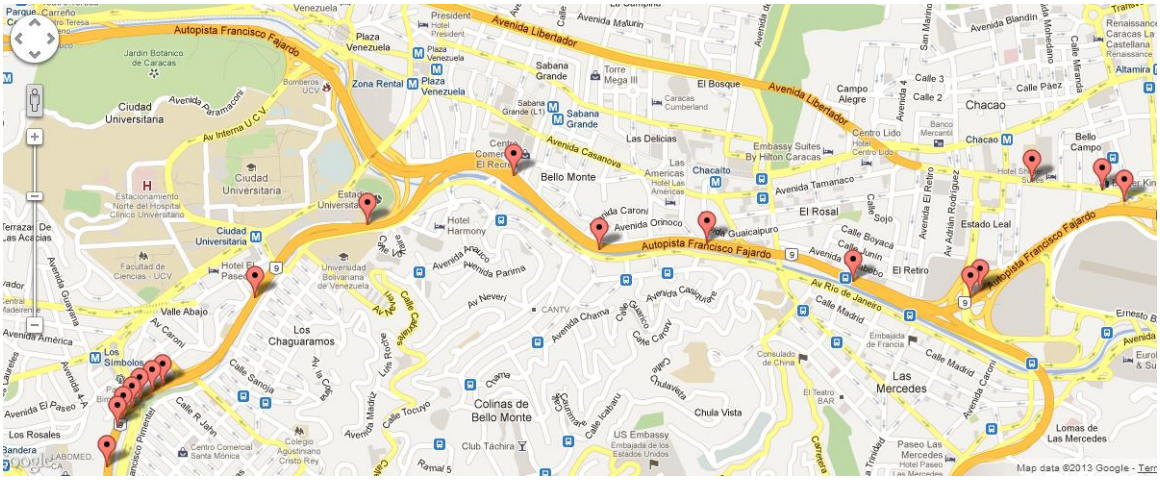

Figura 6.3: Prueba con proveedor GPS, tiempo 30000 ms. y distancia mín. de 50 mts.

- **Proveedor GPS, tiempo mínimo 30000 milisegundos y distancia mínima de 0 metros:** Disminuyendo la distancia a 0 metros se observa que la frecuencia de captura se hace más corta, sin embargo, al ser el tiempo (30 Segundos) el único factor del que depende la misma, se puede observar que en los intervalos donde el vehículo se detiene por algunos minutos la aplicación continúa con la captura de puntos, sobreponiéndolos y creando datos que terminan siendo innecesarios (Ver figura 6.4).

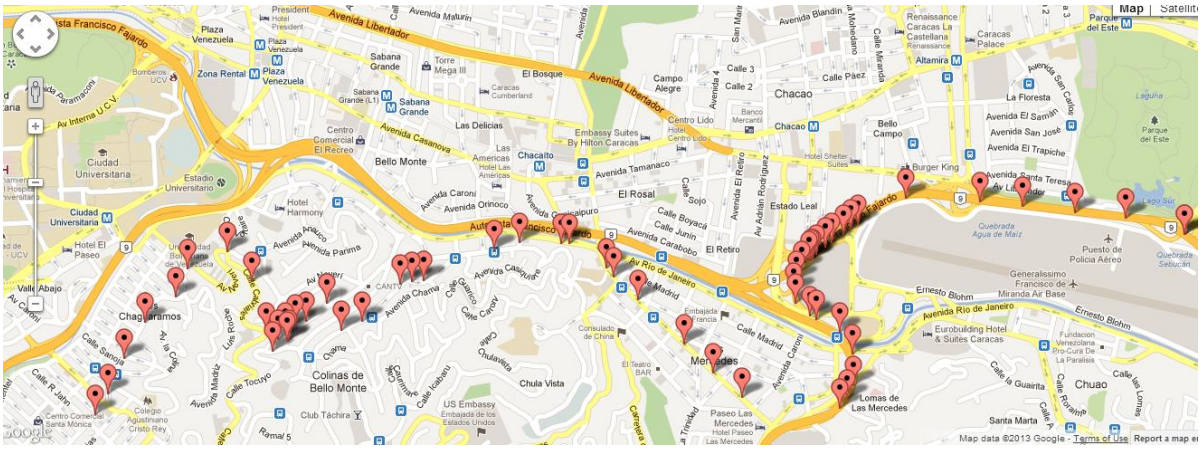

Figura 6.4: Prueba con proveedor GPS, tiempo mín. 30000 ms. y distancia mín. de 0 mts.

- **Proveedor GPS, tiempo mínimo 0 milisegundos y distancia mínima de 5 metros:** Al hacer que la prueba no dependa del tiempo sino sólo de la distancia, se resuelve el problema detectado en la prueba anterior, referente a la sobreposición de puntos cuando el vehículo no esté en movimiento. Sin embargo, al establecer una frecuencia de

captura tan baja (5 metros), se obtiene una cantidad de datos que se vuelve excesiva (ver figura 6.5).

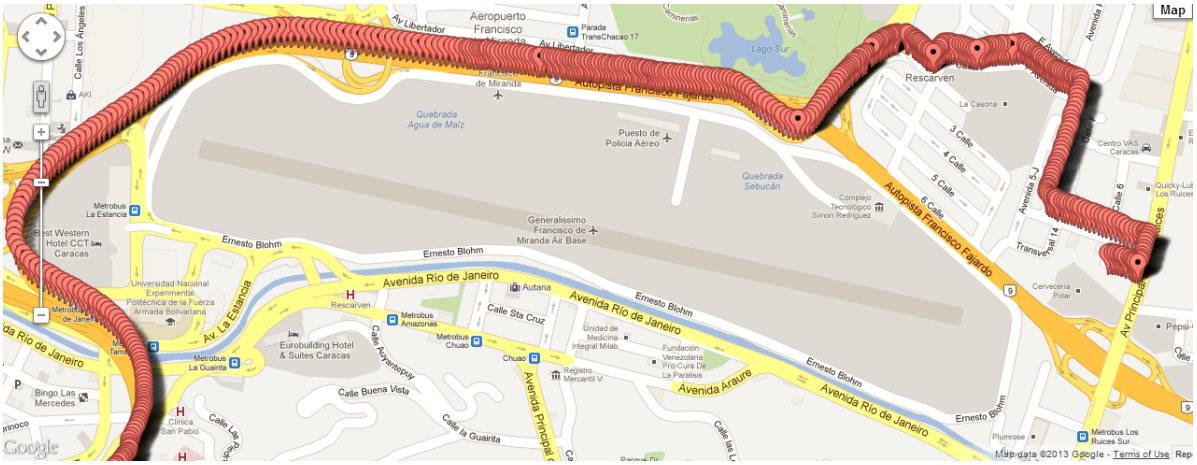

Figura 6.5: Prueba con proveedor GPS, tiempo mín. 0 ms. y distancia mín. de 5 mts.

- **Proveedor GPS, tiempo mínimo 0 milisegundos y distancia mínima de 10 metros:** Esta prueba mantiene la independencia de las capturas con respecto al tiempo. Además, se duplica la frecuencia de captura con respecto a la prueba anterior (de 5 a 10 metros), logrando eliminar los datos de exceso. Por último, el uso de GPS como proveedor de esta prueba, permite mantener una precisión aceptable. Todos estos factores unidos ofrecen un sistema que cumple con los parámetros apropiados para finalizar las pruebas de la aplicación y comenzar con la captura de datos reales (ver figura 6.6).

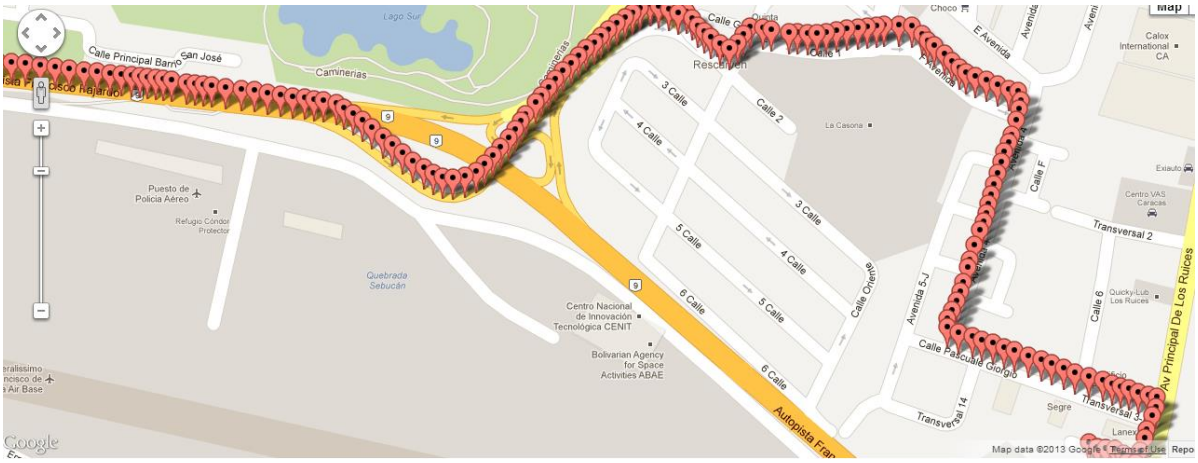

Figura 6.6: Prueba con proveedor GPS, tiempo mín. 0 ms. y distancia mín. de 10 mts.

6.3 Recolección de Datos

Luego de las pruebas realizadas y de establecer los parámetros más indicados para la frecuencia de actualización de posición, se inicia la recolección de datos reales, que como se explicó en los capítulos iniciales de este trabajo, nutrirán un modelo de tráfico vehicular que se construirá posteriormente.

La recolección de datos consiste básicamente en proveer a una unidad de transporte público con un dispositivo en el cual se encuentra instalada la aplicación desarrollada. Dicha aplicación será iniciada cada vez que comience una vuelta de su recorrido diario y será detenida cada vez que dicha vuelta finalice, detallando en cada captura un recorrido completo a la ruta asignada a la unidad de transporte utilizada. El conductor o persona encargada del dispositivo sólo necesitará interactuar con la aplicación para iniciar y finalizar la captura, de resto el programa se encargará automáticamente de recolectar las coordenadas, almacenarlas en la base de datos, enviarlas al servidor y eliminar la información que ya haya sido confirmada como recibida por el servidor.

Para la captura de datos la unidad seleccionada para la realización de la captura de los datos es una camioneta marca *Autogago*, modelo 1993, que mantiene la ruta de Brisas, Carmelitas, Chacaito. El dispositivo móvil utilizado es un *HUAWEI Ascend Y200*, con procesador de 800 Mhz Cortex-A5, 256 MB de Memoria RAM, 512 MB de almacenamiento Interno y S.O. *Android* 2.3. Dicho dispositivo ya había sido probado en diferentes oportunidades durante el período de pruebas realizado previamente.

6.4 Análisis de Datos

Para esta captura en particular se logró obtener la información de dos recorridos (vueltas) completos, lo que arrojó como resultado un total de 7.1 horas de captura, divididas en 3,6 horas para la primera vuelta y 3,5 horas para la segunda vuelta. La cantidad de puntos obtenidos fue 2450 y 2427 para cada vuelta, ofreciendo un total de 4877 puntos.

De los datos recolectados se pueden realizar diferentes análisis, sin embargo la siguiente sección estará enfocada solo a la funcionalidad y desempeño del sistema, ya que el objetivo principal de este trabajo de investigación se basa en la captura de los datos, dejando el procesamiento de los mismos a trabajos futuros que utilizarán métodos más especializados.

En las figuras 6.7 y 6.8 se puede observar gráficamente el trayecto recorrido por la unidad durante la primera y la segunda vuelta respectivamente. Uno de los principales datos

que puede observar a partir de ambas gráficas es que la unidad sigue una ruta muy similar, siendo el sitio de partida/llegada uno de los puntos en el extremo izquierdo del mapa (sector Coco-Frío, Carretera al Junquito) y el punto más alejado el del extremo derecho del mapa (Av. Tamanaco, Chacaito).

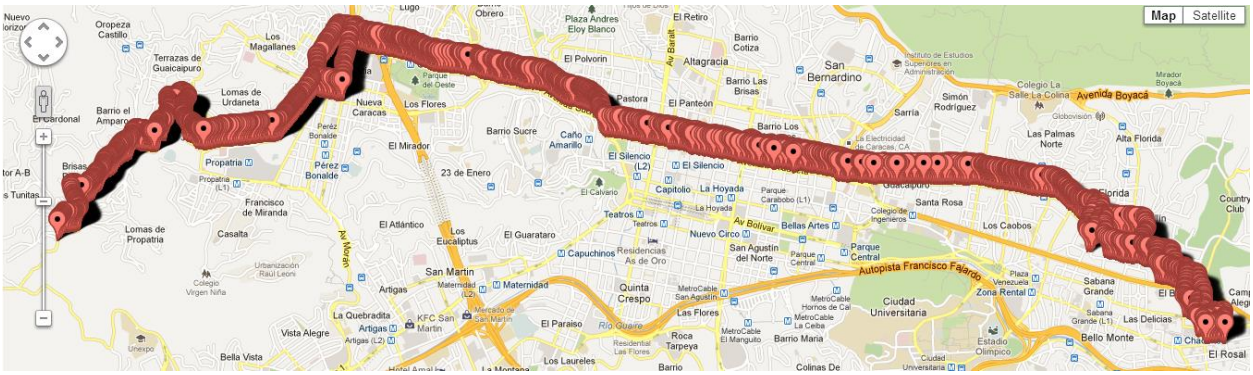

Figura 6.7: Captura de primer recorrido.

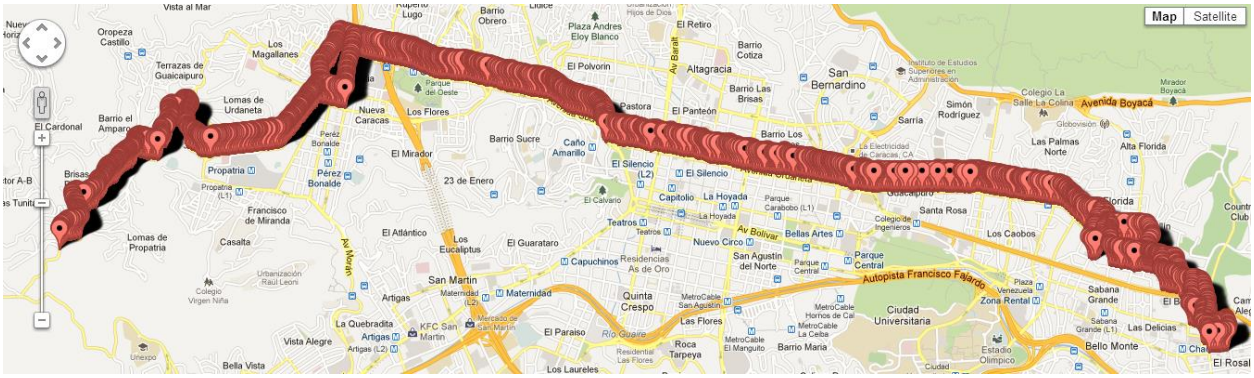

Figura 6.8: Captura de segundo recorrido.

En los siguientes análisis se tomarán en cuenta ambos recorridos de forma grupal y no por separado, a fin de representar la información de manera global, sin diferenciar datos específicos para cada uno de ellos.

6.4.1 Análisis de la Precisión

El primer parámetro a ser evaluado será la precisión, la cual depende del proveedor utilizado (que en este caso es GPS), la visibilidad de los satélites y los obstáculos encontrados (edificios, nubes, túneles, entre otros) en las diferentes rutas recorridas. La precisión en los datos recolectados se encontraba la mayor parte del tiempo entre los 5 y 20 metros, manteniendo un promedio de 10,2 metros. El mínimo valor fue de 5 metros, mientras que el

máximo fue de 169 metros. A continuación se muestra en la figura 6.9 un gráfico detallado de la precisión que se obtuvo en función de la cantidad de puntos obtenidos.

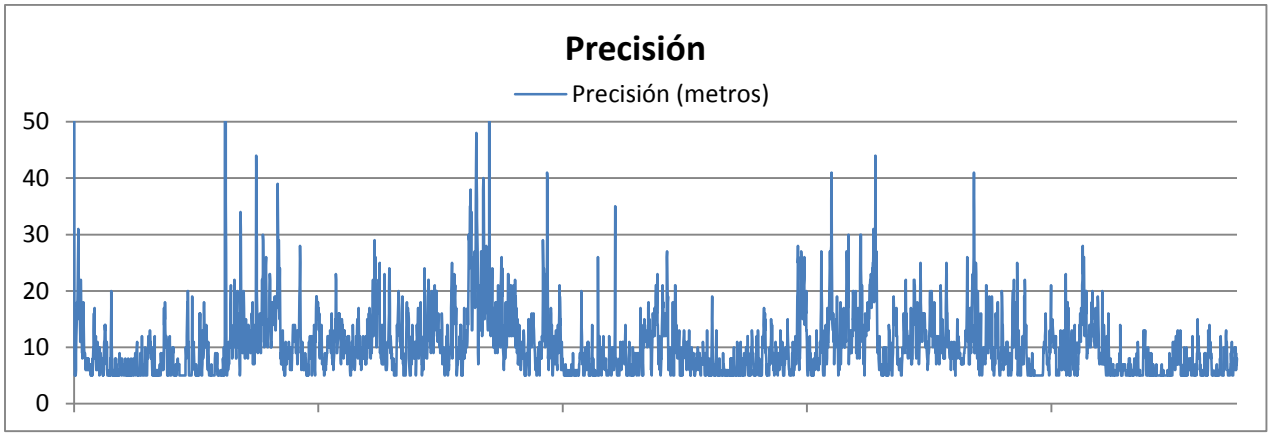

Figura 6.9: Precisión en mts. para cada punto obtenido.

6.4.2 Análisis de la Distancia Recorrida

La obtención de la distancia entre cada punto depende directamente del parámetro preestablecido de *minDistance* (distancia mínima) en la aplicación. Para el caso de esta captura dicho parámetro mantuvo un valor de 10 metros, lo que significa que la posición se actualiza cada vez que se hayan recorrido 10 metros o más. En los datos recolectados la distancia total recorrida fue de 31878 metros, el promedio de distancia entre cada punto obtenido fue de 13,14 metros, la mayor distancia recorrida fue de 210,8 metros y la mínima 0 metros (cuando se iniciaba la captura debido a que no existían puntos anteriores de referencia). En la figura 6.10 se muestra una gráfica representando los valores aquí mencionados.

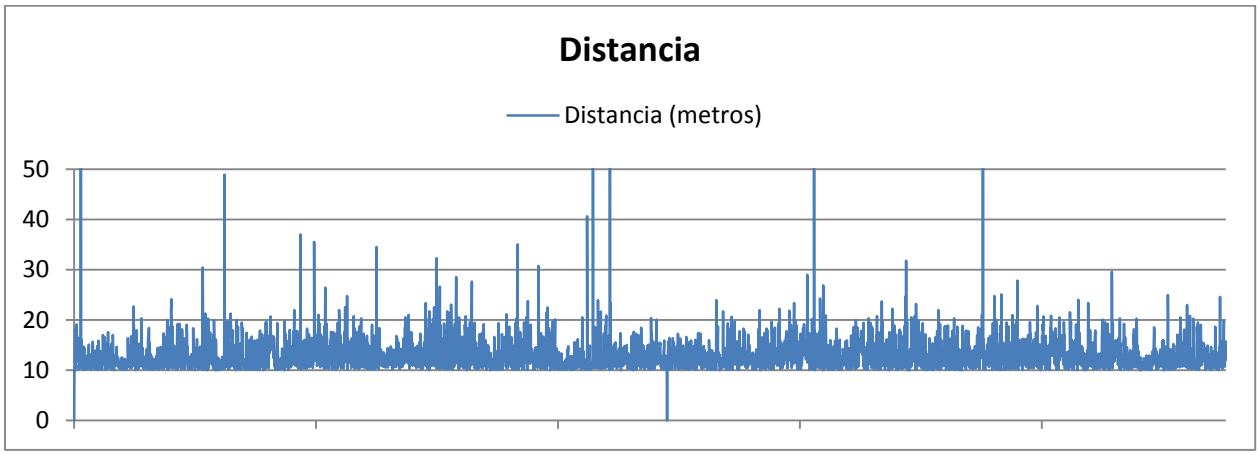

Figura 6.10: Distancia en mts. entre cada punto obtenido.

6.4.3 Análisis de la Velocidad

La velocidad, representada en todo momento de la captura de datos en mts/seg., es calculada con respecto a la distancia y tiempo recorrido desde un punto anterior a un punto actual. Esta mantuvo un promedio de 4,733 mts/seg. Su máximo valor fue de 17,257 mts/seg y su mínimo fue de 0 mts/seg (principalmente cuando se iniciaba la captura). La figura 7.11 muestra una gráfica donde se puede observar el valor de la velocidad con respecto a la cantidad de puntos capturados.

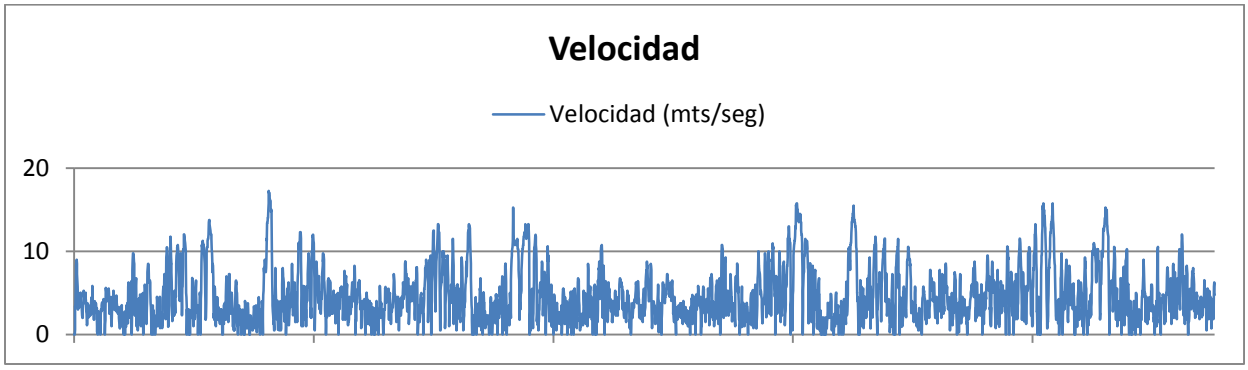

Figura 6.11: Velocidad en mts. entre cada punto obtenido.

Todos los datos e información mostrados en este capítulo demuestran que la aplicación se comportó acorde a lo esperado según los distintos requerimientos y parámetros establecidos, arrojando resultados (posición, precisión, velocidad y distancia) comprendidos dentro de un rango razonable bajo los diferentes ambientes de prueba y de captura de datos, cumpliendo así con una funcionalidad adecuada.

Capítulo 7
Conclusiones y Trabajo Futuros

EL trabajo realizado implicó el desarrollo e implementación de una aplicación para dispositivos móviles con Sistema Operativo *Android* que permite obtener datos de posicionamiento, una página Web que permitiera representar visualmente, descargar la información obtenida y un programa en un servidor capaz de almacenar los datos recolectados y de comunicarse tanto con la aplicación como con la página web. Todo esto con el fin de construir un conjunto de herramientas que permitan obtener datos para el desarrollo de un modelo de tráfico vehicular.

La metodología de prototipado, gracias a su naturaleza iterativa, permitió ir acercando más la aplicación durante cada reunión a los requerimientos del cliente, hasta que finalmente, luego de pasar por diferentes pruebas de funcionalidad y precisión se logró cumplir con cada uno de los resultados esperados, y por consiguiente, con el objetivo de este trabajo especial de grado.

7.1 Contribuciones

El aporte de este Trabajo de Grado fue el desarrollo de un sistema de captura, envío, recepción y visualización de datos de ubicación geográfica utilizando dispositivos con Sistema Operativo *Android* y servicio de GPS. Dichos datos tendrán como principal uso la construcción de un modelo de movilidad de rutas de vehículos de transportes públicos, sin embargo, su uso no se limita a este caso en particular, ya que la aplicación también puede ser empleada para capturas de datos de tránsito de usuarios en vías públicas, datos de ubicación de cualquier tipo de vehículo terrestre o incluso personas, patrones de movilidad para determinar comportamientos, entre otros.

El poder capturar datos de ubicación geográfica de una manera sencilla, automática y con la utilización de dispositivos comunes como teléfonos inteligentes o tabletas, permite facilitar y agilizar la recolección de datos para cualquiera de los casos antes mencionados, abriendo un abanico de oportunidades para trabajos futuros que estudien, modelen e incluso mejoren nuestro entorno.

7.2 Limitaciones

La principal limitación del sistema es que, aunque la aplicación desarrollada puede ejecutarse en cualquier dispositivo móvil con S.O. *Android* mayor a 2.2 y que posea GPS, su funcionamiento y por tanto su interfaz, manejo de datos y comunicación con los demás sistemas están limitados a las especificaciones planteadas en esta investigación. Cualquier modificación de precisión, frecuencia de captura o funcionamiento en general deberá ser realizada directamente en el código de la aplicación, la página web o servidor, y no podrá ser cambiada mediante ningún tipo de opciones o parámetros por parte del usuario.

Otro de la las restricciones, en este caso más relacionada a la puesta en funcionamiento del sistema que a su desarrollo, es que debido a la gran cantidad de puntos necesarios para la realización de un modelo de tráfico vehicular, se debe planificar muy bien la logística para la recolección de los datos, ya que, aunque la aplicación se encarga automáticamente de la gran mayoría del trabajo a realizar, se hace necesaria la presencia humana para activar y desactivar la aplicación cada vez que la unidad de transporte inicia o finaliza un recorrido.

7.3 Trabajos Futuros

En cuanto a trabajos futuros, el sistema desarrollado podrá servir como base para proyectos mayores en donde la captura de datos de posicionamiento y recorrido fuera necesaria, ya sea para la construcción de modelos o para otro requerimiento relacionado. Son muchas las ideas y propuestas que pueden surgir relacionadas a redes vehiculares, prevención de accidentes de tránsito, optimización de rutas, atención al cliente basada en su ubicación, entre otras, y la mayoría de ellas parten por la captura y estudio de recorridos, patrones y demás información de los vehículos y transeúntes.

Referencias

[1] E. Kaplan, (2006), **Understanding GPS: principles and applications.** Norwood Massachusett Artech House, Inc.

[2] P.Correia, (2000), **Guiapractica de GPS.** París, Editions Eyrolles. pp v5 – 21.

[3] F. Zahradnik, 2010, **Smartphone vs. Dedicated Car GPS.** [En línea]. Disponible en: http://gps.about.com/od/gpsproductoverview/a/smartphone-vs-dedicated-gps.htm

[4] Pérez David, 2012. **Sistemas Operativos Avanzados.**

[5] **Condiciones de la Licencia Apache versión 2.0**, [En línea]. Disponible en: http://www.apache.org/licenses /LICENSE-2.0.html

[6] **Página oficial del proyecto Android**, [En línea]. Disponible en: https://developers.google.com/android/

[7] A. Selvig, 2008. **IOS ARCHITECTURE: Mobile Application Development.**

[8] W. Bischofberger, 2011. **Prototyping-Oriented Software Development: Concepts and Tools.** Springer-Verlag New York Incorporated.

[9] W. Suh, 2005, Web Engineering: Principles and Techniques. Pensilvania, Estados Unidos, Idea Group Publishing, pp 82 – 83.

[10] **Activities**, [En línea]. Disponible en: http://developer.android.com

[11] B. Campderrich, 2003, Ingeniería de Software, Eureka Media, Barcelona, España, pp 81- 83.

[12] G. Everett, R. McLeod, 2007, Software Testing: testing across the entire software development life cycle, Jhon Wiley & Sons Inc., Hoboken, New Jersey, pp 99-100.

[13] **Activities**, [En línea]. Disponible en: http://developer.android.com/guide/components/activities.html

[14] **Services**, [En línea]. Disponible en:
http://developer.android.com/guide/components/services.html

[15] **SQLiteOpenHelper**, [En línea]. Disponible en:
http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html