UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Final Máster en Ingeniería de Computadores

Septiembre 2013

Sistema de alerta preventivo contra accidentes ocasionados por badenes

Ejemplo de aplicación distribuida desarrollada para Android

Alumno: **Pedro Rafael Ortego Orozco**
Director: **Enrique Hernández Orallo**

Índice

1. Introducción

Los avances tecnológicos en la fabricación de componentes electrónicos en los últimos años han dado lugar a una revolución, se hacen cada vez más pequeños, más baratos y requieren de menos potencia para funcionar. Las capacidades de cómputo y memoria se han incrementado enormemente. De la mano a los avances en la electrónica, los dispositivos móviles inteligentes se han asentado en el mercado.

Actualmente existe una amplia variedad de dichos dispositivos móviles en el mercado. Por ello ofrecen prestaciones distintas, aunque también comparten muchas características, entre ellas su reducido tamaño y peso lo cual les otorga movilidad, autonomía de procesamiento, de almacenamiento y de alimentación, conectividad inalámbrica e interacción adaptada con el usuario. Sin embargo, la clave de su éxito radica en la capacidad de ofrecer múltiples funciones a sus usuarios.

Estamos entrando en una era donde existen diversos computadores al servicio de una sola persona sin que sea consciente de ello. Mark Weiser, un catedrático y científico ampliamente reconocido por sus contribuciones en el campo de la computación móvil, acuñó el término de computación ubicua en 1988. Ésta puede ser definida como la integración de la informática en el entorno de la persona, de forma que los ordenadores no sean percibidos como objetos diferenciados. Para que esto fuera posible, Weiser escribió sobre dos bases fundamentales, los sistemas distribuidos y la computación móvil.

Es así en que los desarrolladores han llegado a un punto en el que necesitan entender las características y posibilidades que ofrecen los dispositivos móviles para ofrecer aplicaciones que funcionen continuamente y sin problemas en entornos humanos, es decir, que sean conscientes del contexto. El contexto, o información del entorno, se refiere a la información que es parte del

ambiente de operación de la aplicación y que puede ser capturada mediante sensores por la misma. Ejemplos simples del contexto son la localización, la temperatura, la hora, la actividad que realiza una persona, grupo u objetos.

Por tanto, una de las características fundamentales de la computación ubicua es la de capturar experiencias y convertirlas en información útil que esté disponible para usos futuros.

En este proyecto se ha desarrollado una aplicación que ejemplifica los conceptos expuestos anteriormente. A través del uso de sensores en los dispositivos móviles y la computación en la nube que ofrecen compañías como Google, se demostrará la viabilidad de desarrollar aplicaciones que contribuyan a la investigación, al desarrollo científico y tecnológico, y que a la vez sean útiles y mejoren la calidad de vida de la gente que las usa.

1.1. Objetivo

El objetivo de este trabajo es desarrollar una aplicación que sea capaz de prevenir accidentes ocasionados por badenes en calles y autopistas, debido a la velocidad del vehículo antes de pasar por los mismos. Hay dos componentes principales que se han desarrollado para la realización de este proyecto. En primer lugar se encuentra la aplicación cliente sobre la plataforma Android de Google; En segundo el componente servidor, implementado mediante el servicio de cómputo en la nube de Google, el Google App Engine.

La aplicación cliente se utiliza para realizar dos tareas fundamentales, la primera es la captura, análisis y detección de badenes. La información extraída de esta primera tarea es enviada al servidor en la nube. La segunda consiste en prevenir al usuario, con la suficiente antelación, de los badenes que se encuentran a su alrededor.

La aplicación servidor, por su parte, se encarga de guardar la información y localización de badenes enviada por los clientes; asimismo se ocupa de enviar la información de badenes que se encuentren cercanos al usuario en función de la localización que le proporciona el mismo.

Es necesario resaltar que el sistema en conjunto funciona de forma cooperativa, es decir, es necesario que varios dispositivos móviles confirmen la existencia de un badén para que el servidor difunda su ubicación a los clientes que utilicen la aplicación.

1.2. Motivación

Existe evidencia de que los badenes ocasionan accidentes y perjuicios a los pasajeros de un vehículo [1] [2]. En el menor de los casos, cuando un vehículo pasa por un badén a alta velocidad, el riesgo de dañar el vehículo puede ser substancial, y en el peor de los casos podrían causar heridas muy serias a los seres humanos a bordo del mismo. El riesgo se incrementa aún más cuando la conducción se realiza en la noche o cuando existe niebla, lluvia o nieve.

El problema se acentúa todavía más en países en vías de desarrollo donde los badenes no siempre vienen acompañados de la señalización adecuada. Además de la falta de señalización, es frecuente que cuando existan no se encuentren pintados, por lo que en una autopista o carretera sería difícil diferenciarlos del camino. Otro problema es que estos no cumplen algún tipo de estándar o normativa, por lo que pueden tener formas o tamaños irregulares que podrían resultar aún más dañinos si no se tiene la precaución adecuada.

Por supuesto los badenes existen por una razón, y previenen muchos más accidentes de los que llegan a provocar, pero si se logra conservar dichas ventajas y al mismo tiempo reducir las desventajas al mínimo, se cumpliría el objetivo de poner la tecnología al servicio del usuario, de manera que ésta no represente una carga o agobio adicional para el mismo.

1.3. Estructura del Proyecto

Capítulo 1 – Introducción: En esta sección se describe de forma corta y concisa la visión de este trabajo, así como el objetivo que se persigue y la motivación detrás de ello.

Capítulo 2 – Estado del arte: En esta sección se describe el estado tecnológico que rodea a los dispositivos móviles inteligentes, desde el punto de vista del sistema operativo, las tecnologías y herramientas que incorporan, así como las ventajas y desventajas que aporta cada plataforma de desarrollo. También se aborda el estado tecnológico de las plataformas de cómputo en la nube y se comparan algunas de las opciones de desarrollo. Finalmente se describen de forma general algunas de las tecnologías más importantes y que son de especial importancia para la realización del proyecto.

Capítulo 3 – Entorno tecnológico: En esta sección se describen con más detalle las plataformas, herramientas de desarrollo, así como las características del dispositivo utilizado para el desarrollo.

Capítulo 4 – Análisis de la solución: En esta sección se describe el modelo que se ha seguido para obtener la geolocalización pronta y adecuada que requiere el proyecto. Asimismo se exponen las características del sistema de medición detección desarrollado para el proyecto.

Capítulo 5 – Arquitectura del sistema: En esta sección se describe la organización y construcción de los diferentes componentes del sistema, así como el flujo de información que ocurre entre ellos.

Capítulo 6 – Implementación del sistema: En esta sección se describe con detalle el funcionamiento de los componentes individuales del sistema.

Capítulo 7 – Pruebas del sistema: En esta sección se describen las pruebas realizadas y los resultados obtenidos.

Capítulo 8 – Conclusiones: En esta sección se sintetizan los resultados obtenidos y se realiza una reflexión respecto a ellos en comparación con los objetivos marcados inicialmente.

Capítulo 9 – Bibliografía.

2. Estado del Arte

En este capítulo se describe el estado tecnológico que rodea a los dispositivos móviles inteligentes, desde el punto de vista del sistema operativo, las tecnologías y herramientas que incorporan, así como las ventajas y desventajas que aporta cada plataforma de desarrollo con el objetivo de escoger la que mejor se ajuste a las necesidades del proyecto.

A su vez, se aborda el estado tecnológico de las plataformas de cómputo en la nube con el fin de comparar algunas de las opciones disponibles para el desarrollador, y se describen de forma generalizada algunas de las tecnologías más importantes y que son de especial importancia para la realización del proyecto.

Finalmente se describen algunos de los métodos utilizados en otros trabajos para la detección de badenes mediante el uso de dispositivos móviles.

2.1. Dispositivos móviles

Los dispositivos móviles, en particular los llamados teléfonos inteligentes, supusieron una auténtica revolución y un cambio en el paradigma de uso para los usuarios. Anteriormente la utilidad de los mismos se limitaba a realizar llamadas y enviar mensajes de texto, ahora son utilizados como teléfonos, agendas, gestores de contactos, procesadores de texto, reproductores multimedia, cámaras fotográficas y de vídeo, consolas de videojuegos, calculadoras, dispositivos de lectura, GPS y localización, sistemas de radio, televisión, etcétera.

Sin duda alguna esto ha sido posible, no sólo por los avances en hardware, si no por el sistema operativo que estos dispositivos utilizan, así como el entorno de desarrollo que ofrecen. Los sistemas operativos móviles cumplen exactamente la misma función que cumplen en los PC

tradicionales, sin embargo, los sistemas operativos móviles son mucho más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia y a las diferentes maneras de introducir información en ellos.

El sistema operativo para el que se ha decidido desarrollar este proyecto es Android de Google, pero aun así resulta fundamental conocer las diferentes opciones disponibles en el mercado de los dispositivos móviles inteligentes.

2.1.1. Symbian OS

Symbian es un sistema operativo móvil diseñado para teléfonos inteligentes que actualmente es mantenido por la compañía Accenture. Symbian fue desarrollado originalmente por Symbian Ltd., un descendiente de EPOC de Psion y que corre exclusivamente en procesadores ARM. Symbian fue utilizado por muchas de las principales marcas de teléfonos móviles, como Samsung, Motorola, Sony Ericsson, y sobre todo por Nokia, era el sistema operativo para dispositivos móviles más popular en todo el mundo hasta el final de 2010, cuando fue superado por Android.

Debido a la fuerte competencia y ventajas que ofrecen los sistemas operativos de varios de los competidores de Nokia, esta anunció a principios de 2011 que utilizaría al sistema operativo Windows Phone de Microsoft como su plataforma primaria, sustituyendo a Symbian como su principal sistema operativo para teléfonos inteligentes. El 22 de junio 2011 Nokia llegó a un acuerdo con Accenture para un programa de *outsourcing* con el fin de dar servicio y desarrollar software basado en Symbian hasta el año 2016.

El SDK de Symbian está basado en C++ estándar, a través del *framework* Qt. Las aplicaciones hechas en Qt pueden probarse por medio de un simulador que incorpora el mismo. El desarrollo de aplicaciones puede utilizar C++ o QML (Qt Meta Language). La programación puede llevarse a cabo en otros lenguajes y herramientas como Python (llamado Python para S60), Adobe Flash Lite y Java ME.

Symbian hace particular énfasis en la conservación de los recursos; además, toda la programación Symbian está basada en eventos, y la unidad de procesamiento central (CPU) cambia a un modo de bajo consumo de energía cuando las aplicaciones no se ocupan directamente de un evento. Esto se hace a través de una característica de programación llamada “objetos activos”. Asimismo, el enfoque de Symbian para hilos y procesos, es el de la reducción de sobrecarga.

Respecto al uso de sensores, Symbian ofrece para su plataforma el llamado S60 Sensor Framework, un conjunto de APIs que tienen por objeto proporcionar un método consistente de acceso al *hardware* del sensor.

Sólo un número limitado de dispositivos relativamente nuevos puede hacer uso de este *framework*, ya que fue introducido muchos años después del lanzamiento al mercado de los dispositivos Symbian. Por esto existe un alto grado de fragmentación respecto a su uso y de compatibilidad con los dispositivos de esta plataforma. La documentación respecto al *framework* de sensores que aparece en la web de desarrolladores de Nokia, es obsoleta y ha dejado de ser soportada.

El uso de geolocalización en esta plataforma es todavía más confuso, ya que no hay un *framework* como en el caso de los sensores. Los desarrolladores tenían que hacer uso de herramientas externas para proporcionar geolocalización. Fue hasta la tercera versión de Symbian que se lanzó una API con soporte completo y módulos de GPS. Al igual que con el *framework* de sensores, la documentación en la web de desarrolladores de Nokia está obsoleta y desalienta a desarrolladores externos a utilizarla por no garantizar su funcionamiento.

Symbian no es una plataforma apropiada para el desarrollo de este proyecto. La base instalada de dispositivos es baja, ha dejado de ser soportado por Nokia, y además ya no se garantiza que sea compatible con los dispositivos Symbian más recientes. Por otra parte, los desarrolladores terceros ya no pueden subir sus aplicaciones a la *Nokia Store*. Otras plataformas siguen en desarrollo activo, mientras que Symbian está condenado a desaparecer.

2.1.2. BlackBerry 10

BlackBerry 10 es un sistema operativo móvil desarrollado por BlackBerry Limited (anteriormente Research In Motion) para su *BlackBerryline* de dispositivos portátiles de teléfonos inteligentes y tabletas. Está basado en el sistema operativo QNX, que fue adquirido por BlackBerry en abril de 2010. BlackBerry 10 es el sucesor de BlackBerry OS, con el cual comparte numerosas similitudes técnicas, aunque BlackBerry 10 proporciona mejoras sustanciales sobre BlackBerry OS.

Entre las características más novedosas se encuentran una interfaz única, un nuevo teclado inteligente, así como una aplicación de la cámara que permite al usuario ajustar la foto o rostros individuales moviéndose a través de la escala de tiempo para optimizar la calidad de la imagen. La

interfaz de usuario también incluye la capacidad de ejecutar ocho "*Frames* activos" simultáneamente. Estos *Frames* activos son aplicaciones que se están ejecutando en el sistema operativo, pero con un consumo mínimo de recursos. El sistema operativo también cuenta con el *Hub*, una lista accesible desde cualquier parte del sistema operativo donde se pueden encontrar todas las notificaciones como mensajes de correo electrónico, sitios de redes sociales y mensajes de texto. También se ha eliminado el botón de inicio, muy común en los sistemas iOS y Android, por lo que los gestos (movimientos táctiles que realizan acciones predeterminadas en el dispositivo) están integrados en gran medida en el BlackBerry 10, con cuatro gestos principales para facilitar la navegación. La gestión multitarea también está soportada a partir de gestos y algunas aplicaciones también ofrecen *widgets*, de funcionalidad similar a la de Android.

El desarrollo de aplicaciones soporta varios lenguajes de programación, el SDK nativo utiliza C/C++, C++/Qt, y HTML5/Javascript/CSS como SDK para WebWorks. También soporta ActionScript y Adobe Air.

BlackBerry 10 introdujo una capa de ejecución de Android. Esto permite a los desarrolladores re empaquetar y distribuir fácilmente sus aplicaciones diseñadas para trabajar en la versión 2.3 del mismo. Sin embargo, menos del 46% de los dispositivos Android conectados a la tienda Play de Google está ejecutando versiones de Android 2.3 o anteriores. Debido a esto, BlackBerry ha anunciado oficialmente que pronto actualizará esta capa de ejecución para incluir compatibilidad con la versión 4.1. Esta característica añade a Java y el SDK de Android como una plataforma de desarrollo válida para BlackBerry 10.

El SDK nativo de BlackBerry 10 proporciona acceso completo a una gran variedad de sensores a través de las librerías de servicios y sensores de la plataforma. Los tipos de sensores que soporta son el acelerómetro, altímetro, orientación, gravedad, giroscopio, holster, luz, aceleración lineal, temperatura, magnetómetro, proximidad, y temperatura.

Al igual que en el caso de los sensores, el SDK provee acceso completo a librerías muy completas de geolocalización. Éstas ofrecen integración con el servicio de mapas propietario de BlackBerry 10, *BlackBerry Maps*. Los métodos de geolocalización incluyen GPS, WPS, triangulación por antena móvil (*Cell Tower Triangulation*).

Es un sistema operativo en desarrollo activo, con muchas mejoras con respecto a su predecesor. La documentación está actualizada y completa, el soporte a la plataforma es total por parte de

BlackBerry Limited y la capa de compatibilidad con Android es una característica muy interesante. Sin embargo, se ha prescindido de BlackBerry 10 como plataforma de desarrollo por lo reciente de su lanzamiento, la disponibilidad de dispositivos y la experiencia en otras plataformas de desarrollo.

2.1.3. Tizen

Tizen es un proyecto de sistema operativo móvil basado en Linux, patrocinado por Linux Foundation y la Fundación LiMo. Tizen se origina en MeeGo, que a su vez fue una combinación de los sistemas operativos móviles Moblin, creado por Intel y Maemo, creado por Nokia, y está destinado a sustituirlo. También ha sido creado como sucesor de otro sistema operativo de dispositivos móviles, el Bada de Samsung. El desarrollo está dirigido por Intel, Samsung, y algunos ex-desarrolladores de MeeGo.

Las interfaces de desarrollo de Tizen están basadas en HTML5 y otros estándares web optimizadas para su uso en dispositivos móviles, televisores inteligentes y sistemas integrados de información y entretenimiento. El lenguaje de programación utilizado en las aplicaciones es C y C++. Tizen también posee una capa de compatibilidad para correr aplicaciones Android llamado *Application Compatibility Layer* (ACL). Al igual que otros sistemas operativos móviles, las plataformas soportadas por Tizen son ARM y x86.

La primera versión del sistema y su SDK ya ha sido publicada. Tizen está diseñado para ser compatible con las aplicaciones actuales de MeeGo. Se espera que sea más flexible que MeeGo a través de la nueva API basada en HTML5. Los desarrolladores han hecho hincapié en que HTML5 no es la única plataforma disponible y también se han integrado las *Enlightenment Foundation Libraries* en el sistema operativo.

Aunque originalmente fue presentado como un sistema operativo de código abierto, Tizen ha complicado su modelo de licencias. Su SDK está construido sobre componentes de código abierto, pero el SDK completo ha sido publicado bajo una licencia de Samsung de código no abierto.

El sistema operativo en sí mismo tiene muchos componentes de código abierto. Una serie de componentes internos desarrollados por Samsung (por ejemplo la animación del arranque y las aplicaciones de calendario, gestor de tareas y de reproductor de música) son, sin embargo,

publicados bajo la licencia Flora, la cual es incompatible con los requisitos de la *Open Source Initiative*.

Tizen soporta los siguientes sensores de acuerdo a su documentación, acelerómetro, orientación, gravedad, giroscopio, luz ambiental, magnetómetro, proximidad, aceleración lineal y orientación.

Incluye una API de geolocalización bien documentada. Ésta puede realizarse por medio de GPS y WPS.

Es un sistema operativo prometedor, pero su comunidad de desarrolladores todavía es pequeña y su base instalada de dispositivos es insignificante. Su capa de compatibilidad con Android también lo hace un sistema operativo muy interesante, pero se ha decidido utilizar uno distinto debido a la juventud del sistema, lo cual lo hace más proclive a encontrar más problemas y menos soluciones.

2.1.4. Windows Phone 8

Windows Phone 8 es la segunda generación del sistema operativo móvil Windows Phone de Microsoft. Fue lanzado el 29 de octubre de 2012, y al igual que su predecesor, cuenta con la interfaz gráfica conocida como Metro.

Windows Phone 8 sustituye la arquitectura CE utilizada en dispositivos Windows Phone 7 con el núcleo de Windows NT que se encuentra en muchos de los componentes de Windows 8. Los dispositivos actuales con Windows Phone 7.x no se pueden actualizar a Windows Phone 8 y las nuevas aplicaciones compiladas específicamente para Windows Phone 8 no funcionan en los dispositivos con Windows Phone 7.x.

El sistema operativo ha introducido muchas mejoras respecto a Windows Phone 7, entre ellas el sistema de archivos NTFS, controladores, componentes de seguridad y cifrado, medios de comunicación como NFC y soporte gráfico. Windows Phone 8 puede soportar CPUs *multi-core* de hasta 64 núcleos, así como resoluciones de hasta 1280x720 y 1280x768. Además, Windows Phone 8 también incluye soporte de tarjetas MicroSD, para almacenamiento adicional.

El sistema operativo en sí puede realizar multitarea, de hecho las aplicaciones oficiales de Microsoft lo son, pero al igual que su predecesor Windows Phone 7, no se permite que las

aplicaciones de terceros sean verdaderamente multitarea (similar a lo ocurrido con el iOS de Apple).

La plataforma de desarrollo para este sistema operativo es Visual Studio 12. Los lenguajes de programación soportados para la creación de aplicaciones son C++, C# y VB.NET. Es necesario disponer de Windows 8 para poder desarrollar aplicaciones en Windows Phone 8.

Windows Phone 8 soporta múltiples sensores que permiten a las aplicaciones determinar la orientación y el movimiento del dispositivo. La API de movimiento combinado, que combina y procesa información proveniente de todos los sensores, es la forma más sencilla para obtener el movimiento y la información de orientación. Para aplicaciones que lo requieran, Windows Phone también proporciona APIs para recuperar datos de los sensores individuales. Los sensores soportados por esta plataforma son acelerómetro, inclinómetro, giroscopio, brújula, luz, orientación, orientación simple y matriz de rotación.

Respecto a la geolocalización, se puede hacer uso de dos APIs de localización. Una es la *Windows Phone Runtime Location API* y la *.NET Location API*. La primera solamente es compatible con la versión 8 de Windows Phone, mientras que la segunda es compatible con las versiones 8 y 7. Microsoft alienta a usar la primera sobre la segunda por algunas características adicionales así como la sencillez de uso que ofrece, por lo que la segunda solamente debería usarse cuando se desarrolla con ambas plataformas en mente. Las APIs de localización ofrecen integración con el servicio de mapas Bing de Microsoft. Los métodos de geolocalización incluyen GPS, WPS y triangulación por antena móvil.

Es un sistema operativo interesante, en desarrollo activo y tiene el respaldo de una de las corporaciones más grandes del mundo. Sin embargo, es necesario disponer de Windows 8 para desarrollar en ella, la base instalada es mucho menor que la de sus competidores y la compatibilidad entre versiones diferentes del sistema operativo es limitada. También es necesario pagar para tener acceso a la versión con más funcionalidad de Visual Studio 12, así como para instalar la aplicación en el dispositivo.

2.1.5. iOS

El iOS es el sistema operativo móvil de la empresa Apple Inc. Su más reciente versión es la 6, con el lanzamiento de la versión 7 muy próximo a la fecha de redacción de este trabajo. Originalmente fue desarrollado para el iPhone y utilizado posteriormente en dispositivos como el iPod Touch, iPad y el Apple TV. El iOS se deriva de Mac OS X, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo Unix.

La interfaz de usuario táctil de iOS es excelente, tiene una respuesta inmediata, muy fluida y está basada en el concepto de manipulación directa, usando gestos. Los elementos de control consisten de deslizadores, interruptores y botones. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan al sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotarlo en tres dimensiones (un resultado común es cambiar de modo vertical al apaisado u horizontal).

Antes de iOS 4, la multitarea estaba reservada para aplicaciones por defecto del sistema. A partir de iOS 4, se permite el uso de 7 APIs para multitarea, específicamente: audio en segundo plano, voz IP, localización en segundo plano, notificaciones *push*, notificaciones locales, completado de tareas y cambio rápido de aplicaciones.

Las aplicaciones deben ser escritas y compiladas específicamente para la arquitectura ARM, por lo que las aplicaciones desarrolladas para Mac OS X no pueden ser usadas en iOS.

El Xcode es el entorno de desarrollo que Apple ha proporcionado a los desarrolladores para hacer aplicaciones para iOS. Estas aplicaciones, como las de Mac OS X, están escritas en Objective-C. Para poder hacer uso del SDK es necesario disponer de Mac OS. El SDK se puede descargar gratuitamente, pero para publicar el software es necesario registrarse en el programa de desarrollo de Apple, un paso que requiere el pago y la aprobación por parte de Apple. Durante el proceso, se entregan al desarrollador unas claves firmadas que permiten subir una aplicación a la tienda de aplicaciones de Apple.

Las aplicaciones pueden ser distribuidas de tres formas: a través de la App Store de Apple, por parte de una empresa a sus empleados, o sobre una red “Ad-hoc” de hasta 100 dispositivos.

El iOS tiene varios sensores disponibles como en la mayoría de los dispositivos de móviles de última generación, entre ellos el acelerómetro, sensor de proximidad, sensor de luz ambiental,

giroscopio, brújula. Hacer uso de ellos es muy sencillo a través del *framework CoreMotion*, el cual está ampliamente documentado.

El sistema operativo tiene un *framework* de localización llamado *CoreLocation*, muy sencillo de utilizar. Los métodos de geolocalización incluyen GPS, WPS y triangulación por antena móvil.

Es un sistema operativo para móviles muy amigable, sencillo y su interfaz proporciona una respuesta muy satisfactoria. Dado que es un ecosistema cerrado, no existen los problemas de compatibilidad y fragmentación que tiene Android. Tiene una base considerable de usuarios, aunque menor a la de Android, siendo la desproporción aún mayor en España. Al no disponer de Mac OS para utilizar el SDK y a la experiencia en el desarrollo en Android se ha descartado su uso.

2.1.6. Android

Android es un sistema operativo basado en Linux y diseñado principalmente para dispositivos móviles con pantalla táctil. Fue desarrollado inicialmente por Android, Inc., una compañía respaldada económicamente por Google y que más tarde compró en el año 2005. El sistema operativo Android fue presentado en 2007 al mismo tiempo en el que se formó la fundación Open Handset Alliance, un consorcio de compañías de *hardware, software* y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles.

Tiene una gran comunidad de desarrolladores así como de aplicaciones disponibles a través de tiendas oficiales como la de Google y la de Samsung. Existe la posibilidad de obtener *software* externamente a las tiendas oficiales, aunque ésta es la fuente más común de *malware* para esta plataforma.

La plataforma de *hardware* principal de Android es la arquitectura ARM, aunque existe soporte para arquitecturas x86. Un ejemplo de esto último es Google TV, que utiliza una versión especial de Android.

Las aplicaciones se desarrollan habitualmente en Java a través del Android SDK, pero existen otras herramientas de desarrollo. Entre ellas el kit de desarrollo nativo (NDK) para aplicaciones o extensiones en C/C++ y también el Google App Inventor, un entorno visual diseñado para programadores novatos. De igual forma, es posible utilizar librerías de Qt.

Android, al contrario que otros sistemas operativos para dispositivos móviles como iOS o Windows Phone, ha sido desarrollado de forma abierta y se puede acceder tanto al código fuente como a la lista de incidencias donde se pueden ver problemas aún no resueltos y reportar problemas nuevos.

Android ofrece acceso a una amplia variedad de sensores a través de su API. La disponibilidad y precisión de éstos depende de los dispositivos ya que existe una gama muy diversa de los mismos y de la versión del sistema operativo que tengan instalada. Hasta su última versión, la API de Android incluye los siguientes sensores: acelerómetro, temperatura ambiental, gravedad, giroscopio, luz, aceleración lineal, campo magnético, orientación, presión, proximidad y humedad relativa.

El sistema operativo también posee una API de localización muy sencilla y versátil. Se pueden integrar los servicios de localización con la propia API de mapas de Google, *Google Maps*. Los métodos de geolocalización incluyen GPS, WPS y triangulación por antena móvil.

Es el sistema operativo con la mayor cuota de mercado. Es un sistema abierto, con una gran cantidad de desarrolladores y documentación muy completa. El SDK es gratuito y no es necesario pagar para probar la aplicación en el dispositivo. Android también permite la multitarea para aplicaciones desarrolladas por terceros, con el inconveniente del posible consumo más rápido de la batería. Por otro lado, la fragmentación es un gran problema en Android, existen muchas versiones del sistema operativo, para múltiples plataformas, lo cual genera problemas de compatibilidad y requiere de esfuerzo extra por parte del desarrollador para asegurarse que su aplicación es compatible con la mayor parte de los dispositivos disponibles, pero se ha escogido como sistema operativo para este proyecto debido a la facilidad que proporciona para desarrollar, así como la base de dispositivos instalada.

2.2. Computación en la Nube

La computación en la nube, también conocida como *Cloud Computing*, es un paradigma que permite ofrecer servicios de computación por medio de internet. Toma su metáfora de internet, ya que la nube representa todo aquello que hace que la red funcione y en el que la computación es ofrecida como un servicio. El objetivo principal es el ahorro de costes (*hardware*, mantenimiento, etc.).

Un ejemplo sencillo sería el de una empresa de pequeño tamaño que desarrolla una aplicación cliente que requiere capacidad de cómputo y almacenamiento de datos. A dicha empresa no le interesa, ni le conviene arriesgar en la adquisición de servidores dedicados, ya que si la aplicación resulta menos exitosa de lo esperado, las pérdidas serían estrepitosas. Sería preferible para la empresa alquilar los servicios que necesita a un proveedor y que éste se encargue del mantenimiento del mismo. Las características generales de un servicio de computación en la nube son los siguientes:

- **Auto Servicio bajo demanda** – Un consumidor puede proveerse de forma unilateral de recursos sin necesitar interactuar directamente con el proveedor de servicio.

- **Acceso a través de internet** – Las capacidades se proporcionan a través de la red con unos mínimos requerimientos del cliente.

- **Elasticidad** – El consumidor puede aumentar o disminuir dinámicamente el número de recursos en cualquier momento, percibiendo una ilusión de capacidad infinita.

- **Servicio mediante pago por uso** – Los recursos utilizados se contabilizan de forma independiente (almacenamiento, computación, ancho de banda, etc.) y precisa.

- **Alta Configuración** – Los recursos utilizados deben de tener un alto grado de configuración con el objetivo de poder adaptarse a las necesidades de los diferentes usuarios.

A pesar de la gran cantidad de beneficios que ofrece, la computación en la nube también tiene algunas desventajas. Una de las principales es que se dejan aspectos clave de un negocio en manos de terceros, por lo que la dependencia y éxito del mismo está directamente ligado al proveedor. Este debe ofrecer condiciones óptimas de disponibilidad, así como de seguridad de los datos alojados en sus servidores. Otra desventaja es la dependencia al acceso a internet y la imposibilidad de proporcionar un servicio al cliente si hay cortes.

2.2.1. Modelos de despliegue

Existen diferentes modelos de despliegue de acuerdo a la exclusividad de la arquitectura utilizada:

Nube Privada (*Private Cloud*)

Infraestructura de uso exclusivo por una institución. Es una infraestructura bajo demanda gestionada para un solo cliente que controla las aplicaciones que deben ejecutarse y el lugar para hacerlo. El cliente es propietario del servidor, de la red y el disco, además que puede decidir qué usuarios están autorizados a utilizar la infraestructura.

Nube Pública (*Public Cloud*)

Infraestructura disponible para todo el mundo. Es mantenida y gestionada por terceras personas no vinculadas con la organización. En este esquema tanto los datos como los procesos de diferentes clientes se mezclan en los servidores, sistemas de almacenamiento y otras infraestructuras de la nube. Los usuarios finales de la nube no conocen los recursos destinados a otros clientes que puedan estar ejecutándose en el mismo servidor. Las aplicaciones, almacenamiento y otros recursos están disponibles al público a través el proveedor de servicios que es propietario de toda la infraestructura en sus centros de datos; el acceso a los servicios solo se ofrece de manera remota, normalmente a través de Internet.

Nube de Comunidad (*Community Cloud*)

Infraestructura compartida entre organizaciones de una comunidad en específico con preocupaciones o prioridades similares (seguridad, jurisdicción, etc.), ya sea administrada internamente o por un tercero y que sea alojada interna o externamente. Los costes se reparten entre menos usuarios que una nube pública, pero más que una nube privada.

Nube Híbrida (*Hybrid Cloud*)

Composición entre varios de los modelos anteriores. El cliente es propietario de algunas partes y comparte otras de manera controlada. Dicha composición expande opciones de implementación, permitiendo a las organizaciones de TI utilizar los recursos de computación en la nube pública para

satisfacer necesidades temporales, pero añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes.

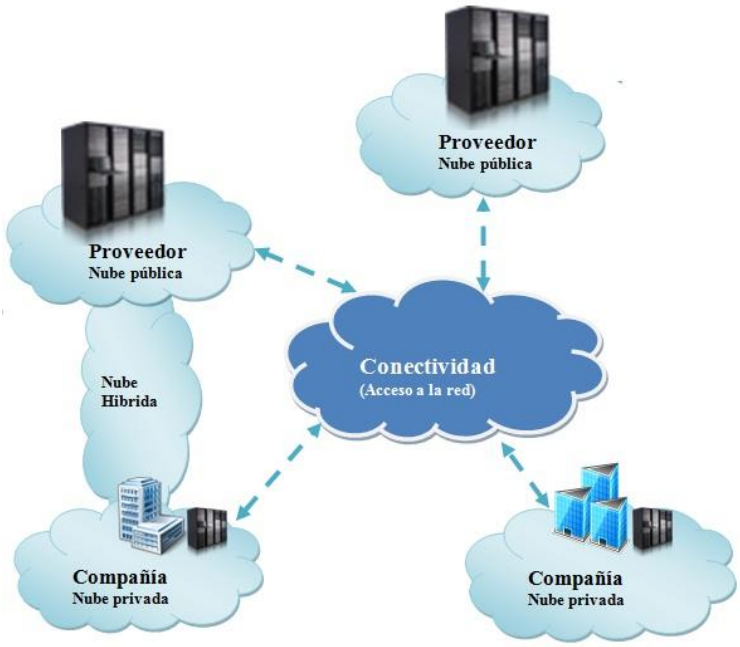


Figura 2.1 Modelos de despliegue

2.2.2. Modelos de Servicio

También existen diferentes modelos de computación en la nube de acuerdo al tipo de servicio ofrecido:

Software como Servicio (SaaS)

Este modelo se caracteriza por ofrecer una aplicación bajo demanda al usuario final. Una sola instancia del software se ejecuta en la infraestructura del proveedor y sirve a múltiples clientes. Existen varios ejemplos de tipos de aplicaciones ofrecidas bajo este esquema, algunas de ellas son las CRM (*Customer Resource Management*), contabilidad, gestión de contenido web, correo y chat. Uno de los ejemplos más reconocidos es Salesforce.

Plataforma como Servicio (PaaS)

En la plataforma como servicio se proporciona un entorno de desarrollo y ejecución. Podría definirse como la encapsulación de una serie de módulos o complementos que proporcionan,

normalmente, una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.). Las ofertas de PaaS pueden dar servicio a todas las fases del ciclo de desarrollo y pruebas del software, o pueden estar especializadas en cualquier área en particular, tal como la administración del contenido.

Algunos ejemplos comerciales incluyen a Google App Engine, que sirve aplicaciones de la infraestructura Google, y también Windows Azure de Microsoft.

Infraestructura como Servicio (IaaS)

Mientras las dos anteriores ofrecen aplicaciones a los clientes, la infraestructura como servicio ofrece *hardware*. Es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. La infraestructura como servicio permite el alquiler de recursos como memoria, ciclos de CPU, almacenamiento, red, etc. Bajo este esquema servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran (por ejemplo a través de la tecnología de virtualización) para manejar tipos específicos de cargas de trabajo, desde procesamiento en lotes, hasta aumento de servidor/almacenamiento durante las horas pico.

El ejemplo comercial mejor conocido es Amazon Web Services, cuyos servicios EC2 y S3 ofrecen cómputo y servicios de almacenamiento esenciales (respectivamente).

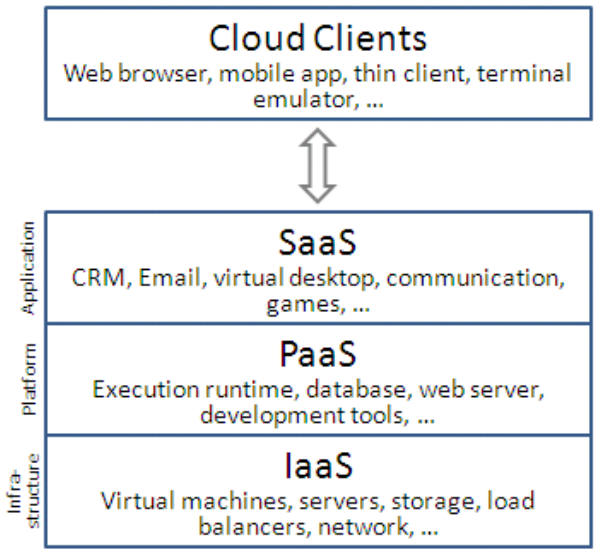


Figura 2.2 Modelos de servicio

2.2.3. Proveedores de Cómputo en la Nube

Existe una cantidad inmensa de proveedores de cómputo en la nube. El proyecto en si necesita de un servidor web, capaz de servir las peticiones de múltiples usuarios así como de guardar en una base de datos la información de los badenes encontrados en el camino. Por tanto el modelo que se empleará es el de una plataforma como servicio. El proveedor seleccionado es Google App Engine, pero resulta conveniente mostrar algunas de las opciones disponibles en el mercado.

Windows Azure

Lanzada en el 2010, Windows Azure es una plataforma como servicio alojada en los centros de datos de Microsoft. En esta plataforma, el servicio de Windows Azure es el encargado de proporcionar el alojamiento de las aplicaciones y el almacenamiento no relacional. Dichas aplicaciones deben funcionar sobre Windows Server 2008 R2 y pueden estar desarrolladas en .NET, PHP, C++, Ruby, Java. Además de proporcionar el servicio de ejecución, Windows Azure dispone de diferentes mecanismos de almacenamiento de datos como bases de datos SQL y blobs.

Windows Azure utiliza un sistema operativo especializado para ejecutar su "capa de fábrica", un clúster en los centros de datos de Microsoft que gestiona el cómputo y almacenamiento de recursos de los equipos así como la disposición de recursos (o un subconjunto de ellos) a las aplicaciones que se ejecutan sobre Windows Azure.

Windows Azure puede describirse como una "capa de nubes" ubicada en la parte superior de una serie de sistemas de Windows Server, que utilizan Windows Server 2008 y una versión personalizada de *Hyper-V*, conocida también como *Windows Azure Hypervisor*, que sirve para proporcionar virtualización de los servicios. La escalabilidad y la fiabilidad son controlados por el llamado *Fabric Controller* de Windows Azure, el cual se encarga de que los servicios y el entorno general de aplicación no se pierdan si alguno de los servidores dejara de funcionar en el centro de datos. También realiza la gestión de los recursos como memoria y balanceo de carga de la aplicación del usuario.

Una de las principales desventajas de Windows Azure es que hay que ajustarse a la plataforma durante el desarrollo. También es más restrictiva y la curva de aprendizaje es mayor. Tiene un costo, sin importar la cantidad de tráfico o de datos almacenados, si bien ofrece un modelo de prueba de 3 meses el cual resulta insuficiente para el desarrollo del proyecto.

Amazon Web Services Elastic Beanstalk

Elastic Beanstalk es la solución de plataforma como servicio que ofrece Amazon Web Services con el fin de permitir a los usuarios crear aplicaciones y que a su vez puedan hacer uso del conjunto de servicios ofrecidos por Amazon Web Services. Entre ellos se encuentran *Amazon Elastic Cloud Compute* (Amazon EC2), *Amazon Simple Storage Service* (Amazon S3), *Amazon Simple Notification Service* (Amazon SNS), *Elastic Load Balancing* y *Auto Scaling* con el objetivo de ofrecer una infraestructura de alta fiabilidad, escalable y rentable de la que puedan depender sus clientes.

Elastic Beanstalk se compila mediante pilas de software conocido, como Apache HTTP Server para Node.js, PHP y Python, Passenger para Ruby, IIS 7.5 para .NET y Apache Tomcat para Java. No se cargan tarifas adicionales por Elastic Beanstalk; solo paga por los recursos de AWS que necesite para almacenar y ejecutar las aplicaciones.

Elastic Beanstalk ofrece una serie de herramientas para facilitar la creación de versiones de la aplicación, que incluye *AWS Management Console*, la implementación de Git y la interfaz de línea de comandos, *AWS Toolkit for Visual Studio* y *AWS Toolkit for Eclipse*.

Un entorno que utilice la configuración predeterminada ejecutará una única instancia muy pequeña de Amazon EC2 (servidor de la aplicación) y *Elastic Load Balancer*. La instancia EC2 se configura para auto escalado, por lo que se agregarán instancias adicionales para gestionar los picos de la carga de trabajo o del tráfico (las instancias progresivas se descartarán si el tráfico disminuye). Asimismo, el equilibrador de carga distribuye el tráfico entrante por varias instancias de Amazon EC2. Cada instancia de Amazon EC2 se crea a partir de imágenes de la máquina de Amazon (AMI), que contienen toda la información necesaria para crear una nueva instancia de un servidor. Elastic Beanstalk utiliza la AMI de Amazon Linux o la AMI de Windows Server 2008 R2 de modo predeterminado. Estas AMI contienen todo el software para actuar tanto como un servidor web como un servidor de la aplicación (p. ej., Linux, Apache y PHP).

Es una plataforma interesante, gratuita para pequeñas cantidades de tráfico y disponible para el desarrollador en varios SDK. Elastic Beanstalk es completamente personalizable y muy flexible. Permite también controlar la forma en la que escala la aplicación. La desventaja es que la curva de aprendizaje es más larga, y el tiempo de desarrollo por lo general también es más largo.

Google App Engine

Google App Engine es una plataforma de servicios de cómputo en la nube para el desarrollo y alojamiento de aplicaciones *web* en los centros de datos gestionados por Google. Google App Engine ofrece escalabilidad automática de aplicaciones *web*; conforme aumenta el número de solicitudes para una aplicación, App Engine asigna automáticamente más recursos a la misma.

Google App Engine es gratuita hasta que los recursos consumidos alcanzan un cierto nivel. Las tasas se cobran por conceptos de almacenamiento adicional, ancho de banda, o las horas de instancia que requiere la aplicación.

Actualmente, los lenguajes de programación soportados son Python, Java (y, por extensión, de otros idiomas JVM como Groovy, JRuby, Scala, Clojure y PHP a través de una versión especial de Quercus), Go y PHP, aunque estos dos últimos en un estado experimental. Google planea implementar más lenguajes en el futuro, ya que Google App Engine ha sido creado para ser independiente del lenguaje.

Google App Engine soporta varios *frameworks* y estándares Java. Resulta esencial la tecnología Java Servlet 2.5 mediante el empleo del servidor web Jetty de código abierto, junto con tecnologías como JSP. Las tecnologías de almacenamiento de datos como JDO (Java Data Objects) son relativamente desconocidas para los desarrolladores, pero son sencillas de implementar mediante la API de persistencia de Java y están bien documentadas.

Google App Engine goza de alta fiabilidad y soporte, con reportes de hasta el 99.95% de SLA (*Service Level Agreement*) para todas las aplicaciones de almacén de datos de alta replicación. La plataforma está diseñada de tal manera que se pueden mantener múltiples cortes de centros de datos sin ningún tiempo de inactividad.

Google ofrece soporte de sus ingenieros si se goza de una cuenta Premium. Existe soporte gratuito por medio de los llamados *App Engine Groups*, aunque no se garantiza la asistencia por parte del *staff* de Google.

En comparación con otras plataformas como servicio, Google ofrece un entorno más amigable para desarrollar aplicaciones escalables; asimismo brinda un modelo de uso adecuado para este proyecto, además de disponer de soporte igualmente gratuito. La curva de aprendizaje es menor

que otras plataformas, la facilidad de implementación es mayor y más rápida. Sus principales desventajas son que se debe adaptar la aplicación a las APIs que ofrece GAE, el desarrollo está limitado por el momento a Java y Python, por lo que es poco flexible y la gama de herramientas (como bases de datos) está limitada.

2.3. Geolocalización en Android

La geolocalización se ha vuelto una característica casi indispensable en los dispositivos móviles actuales. Conocer la localización del usuario permite a una aplicación ser más inteligente y ofrecer mejor información y prestaciones al usuario.

La geolocalización puede ser definida como la identificación de la ubicación geográfica en el mundo real de un objeto, en el caso de este proyecto, un dispositivo móvil. El concepto de geolocalización está estrechamente relacionado con el uso de sistemas de posicionamiento, pero se distingue de ellos por un mayor énfasis en la determinación de una ubicación significativa, como la dirección de un lugar o el nombre de una calle, en lugar de sólo un conjunto de coordenadas geográficas que para el usuario final no significan mucho.

En el caso de Android, cuando se desarrolla una aplicación con reconocimiento de ubicación, se pueden utilizar tanto el GPS como el proveedor de localización por red de Android. Por lo general, el GPS es más exacto, aunque sólo funciona al aire libre, consume mucho más rápido la batería y el dispositivo puede tardar en inicializarse. Por otra parte, el proveedor de localización por red, determina la ubicación del usuario mediante la triangulación de señales de antenas de telefonía (*cell tower triangulation*) y las señales WiFi, proporcionando información de una manera que funciona en interiores como en exteriores, se inicializa mucho más rápido y utiliza menos energía de la batería. No se tiene que decidir entre uno u otro método, si la aplicación lo requiere se pueden utilizar ambos métodos de geolocalización.

Independientemente del método elegido, existen varias razones por las que la lectura de la ubicación puede contener errores y ser inexacta. Los retos para determinar la ubicación del usuario incluyen:

- **Múltiples fuentes de localización** – Tanto el GPS, como el Cell-ID y WiFi proporcionan pistas respecto a la localización del usuario. Determinar cuál usar es

una cuestión de poner en una balanza a la precisión, la velocidad y eficiencia de la batería.

- **Movimiento del usuario** – La ubicación del usuario cambia constantemente, se debe de tomar en cuenta esto y recalcular su ubicación cada cierto tiempo.

- **Precisión variable** – Las estimaciones de localización de cada una de las fuentes utilizadas no son consistentes en su precisión. Una ubicación obtenida 10 segundos antes de una fuente puede ser mucho más exacta que la localización más reciente de otra fuente.

Hay muchos casos de aplicación de la geolocalización, siendo dos de los más comunes el etiquetado de contenido creado por el usuario a partir de una ubicación y el de ayudarlos a decidir hacia dónde dirigirse.

En el primer caso, se podría pensar en usuarios que comparten sus experiencias locales, como por ejemplo visualizar contenido aumentado a partir de la ubicación actual. En la figura 2.3 se muestra un modelo de cómo esta interacción puede ocurrir respecto a los servicios de localización de Android.

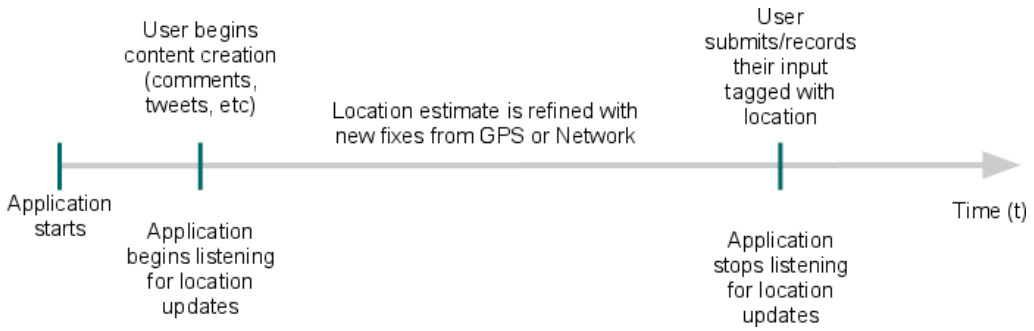


Figura 2.3 Geolocalización y etiquetado de contenido

En el segundo caso, se podría pensar en una aplicación que trata de proporcionar a sus usuarios un conjunto de opciones sobre dónde ir. Por ejemplo, proporcionar una lista de restaurantes de la zona, tiendas y lugares de entretenimiento, y el orden de las recomendaciones cambie dependiendo de la ubicación del usuario.

Para dar cabida a dicho flujo, se podría optar por reorganizar las recomendaciones cuando se obtiene una nueva estimación óptima o por dejar de escuchar las actualizaciones si el orden de las recomendaciones se ha estabilizado. En la figura 2.4 se visualiza un modelo de cómo esta interacción puede ocurrir respecto a los servicios de localización de Android.

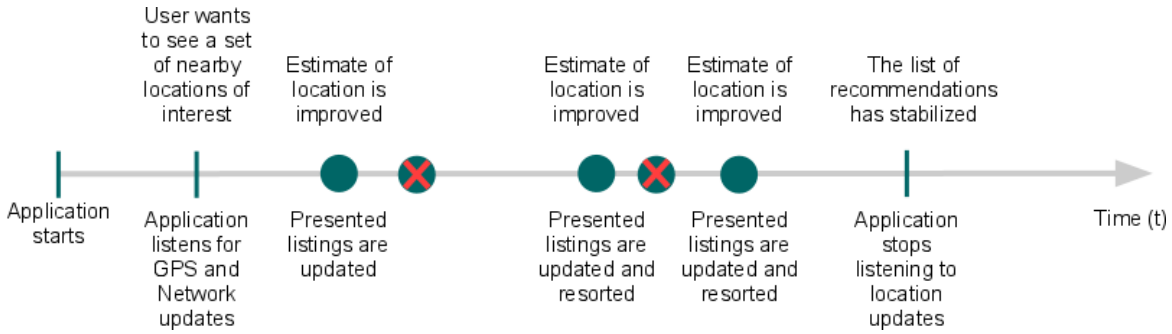


Figura 2.4 Geolocalización y listado de contenido

En el caso de la aplicación que ocupa a este proyecto, realmente se hace uso de modelos muy parecidos a los de los presentados. La detección de badenes funciona más como el primer caso, con la diferencia que el contenido creado o etiquetado son los badenes por los que ha pasado.

La prevención de accidentes funciona de forma más bien parecida al segundo caso, pero en lugar de mostrar al usuario un conjunto de lugares cercanos donde parar, le son mostrados los badenes en la zona con el objetivo de pasarlos con precaución.

2.4. Sensores en los dispositivos móviles

Los sensores en los dispositivos móviles han abierto un abanico de posibilidades en la creación de aplicaciones.

La mayoría de dispositivos Android incorporan en mayor o menor medida sensores de movimiento, orientación y de otras condiciones ambientales. Estos sensores son capaces de brindar datos de gran precisión y son útiles si se desea medir y/o controlar la posición tridimensional del dispositivo, o bien si se desea monitorizar los cambios del medioambiente. Se pueden enumerar multitud de aplicaciones prácticas, por ejemplo, un juego puede rastrear las lecturas de sensor de gravedad de un dispositivo para inferir los gestos y movimientos de usuario complejas, como la inclinación, el movimiento, rotación o giro. Del mismo modo, una aplicación de tiempo podría

utilizar el sensor de temperatura y humedad para calcular e informar el punto de rocío, o una aplicación de viajes podría usar el sensor de campo geomagnético y acelerómetro para funcionar como una brújula.

Todos los sensores se manejan de forma homogénea, todos devuelven valores escalares y todos tienen cierto consumo, resolución y periodo de muestreo. Android divide los tipos de sensores en tres categorías:

- **Sensores de movimiento.** Estos sensores miden las fuerzas de aceleración y fuerzas rotacionales a lo largo de tres ejes. Esta categoría incluye los acelerómetros, sensores de gravedad, giroscopios y sensores de rotación del vector.

- **Sensores ambientales.** Estos sensores miden diferentes parámetros ambientales, tales como temperatura del aire ambiente y la presión, la iluminación, y la humedad. Esta categoría incluye barómetros, fotómetros y termómetros.

- **Sensores de posición.** Estos sensores miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

Para todos los dispositivos móviles, los sensores utilizan un estándar de 3-ejes del sistema de coordenadas para expresar los valores de datos, aunque la forma de etiquetar las coordenadas puede variar para diferentes dispositivos. En el caso de Android, el sistema de coordenadas se define en relación a la orientación y a la pantalla del dispositivo. Cuando el dispositivo se mantiene en su orientación predeterminada, el eje X es horizontal y apunta hacia a la derecha, el eje Y es vertical y apunta hacia arriba, y los puntos del eje Z hacia el exterior de la superficie de la pantalla. Este sistema de coordenadas es utilizado por todos los sensores de movimiento y posición. Se puede apreciar fácilmente en la figura 2.5.

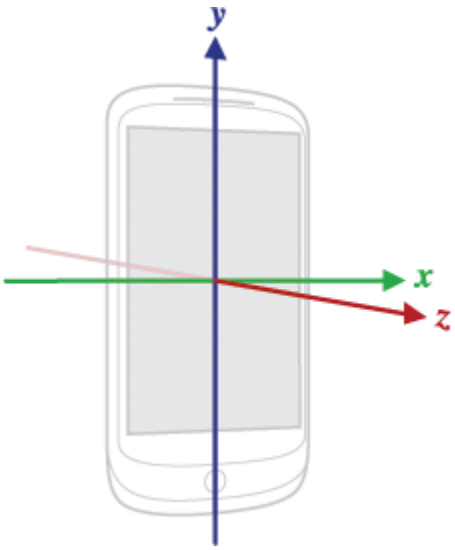


Figura 2.5 Sistema de coordenadas utilizado en dispositivos Android

2.4.1. Tipos de sensores

La disponibilidad de los sensores depende de los dispositivos, y además de la versión de sistema. A continuación se presentan los que por lo general se pueden encontrar en los dispositivos Android.

Acelerómetro

Este tipo de sensor detecta las fuerzas de aceleración a las que se ve sometida una masa en los 3 ejes del sistema de coordenadas (x, y, z), incluyendo la fuerza de gravedad. La fuerza es alterada por los cambios en la velocidad de la masa. La magnitud del valor que devuelve está en metros/segundo^2.

El inconveniente que puede presentar este sensor es que si se gira a velocidad constante sobre el eje z, entonces no se apreciarán cambios en este sensor.

Se utiliza sobre todo para detección de movimiento, para medir los cambios en la velocidad de un objeto y para medir la inclinación del dispositivo con respecto al centro de masa de la tierra. Las lecturas de este sensor son utilizadas en este proyecto para la detección de los badenes.

Giroscopio

Este tipo de sensor detecta la rotación a la que es sometido un dispositivo en los tres ejes (x, y, z). Funciona mediante el principio de conservación de la cantidad de movimiento en un cuerpo que puede girar libremente sobre sí mismo. La magnitud del valor que devuelve el sensor está en radianes/segundo.

En la práctica se puede utilizar para mejorar las prestaciones de un sensor de orientación (porque lo hará más sensible a la rotación) o para medir la velocidad de giro en una aplicación de navegación dentro de un automóvil.

Magnetómetro

Este tipo de sensor detecta la fuerza y dirección del campo magnético en los tres ejes (x, y, z). La magnitud del valor devuelto está dada en μT (micro-Tesla).

Este sensor funciona por medio de materiales magneto-resistivos, su resistencia cambia de acuerdo al ángulo de incidencia de un campo magnético. El inconveniente que tiene este sensor es que si gira tomando como eje una línea de campo magnético, entonces no se apreciará un cambio.

Es frecuente el uso de este sensor para crear aplicaciones como brújulas. También sirven para detectar metales y corrientes eléctricas. En el proyecto se utiliza este sensor para asistir en la orientación del usuario.

Orientación

Este tipo de sensor detecta la orientación que tiene el dispositivo. Esto se puede lograr combinando las lecturas del acelerómetro y el magnetómetro. El acelerómetro permite obtener un vector al centro de masa de la tierra, a su vez que el magnetómetro obtiene un vector al polo norte magnético. Combinando estas dos informaciones se puede conocer a donde apunta el dispositivo en sus 3 ejes. La magnitud devuelta por el dispositivo está en grados, con valores posibles desde 0 a 359.

La realidad aumentada es una de las aplicaciones prácticas de dicho sensor.

Gravedad

Relacionado con el acelerómetro, pero este sensor mide solamente la fuerza de gravedad que es aplicada sobre dispositivo en los tres ejes (x, y, z). Cuando el dispositivo está en reposo, las lecturas del acelerómetro, así como de la gravedad deberán de ser idénticas. La magnitud de la medición también está dada en metros/segundo^2.

Se utiliza normalmente para detección de movimiento.

Aceleración Lineal

Este tipo de sensor detecta las fuerzas de aceleración a las que se ve sometida una masa en los 3 ejes del sistema de coordenadas (x, y, z), pero al contrario del acelerómetro se excluye la fuerza de gravedad. La magnitud de la medición también está dada en metros/segundo^2.

Se utiliza en aplicaciones que necesiten monitorizar movimiento sin tomar en cuenta a la fuerza de gravedad.

Presión

Este sensor detecta la presión atmosférica y funciona mediante la piezorresistencia. La magnitud de la lectura está en hPa (milibarias).

Las aplicaciones más comunes son altímetros y barómetros digitales.

Proximidad

Este tipo de sensor detecta si hay un objeto a menos de 5 cm del dispositivo. Este funciona emitiendo una luz infrarroja que mide con el sensor de luz. La magnitud de la medición devuelta por el sensor está en cm (centímetros).

La utilidad práctica más común es la de apagar la pantalla cuando se habla por medio del dispositivo. También es usado frecuentemente en juegos.

Temperatura Ambiental

Este sensor detecta la temperatura ambiental. No muchos dispositivos incorporan este sensor en la actualidad. La magnitud que devuelve el sensor está en grados Celsius.

En la práctica se utiliza para monitorizar la temperatura en el entorno del dispositivo.

Temperatura

Este sensor detecta la temperatura interna del dispositivo. Funciona mediante transistor integrado, aprovechando la dependencia de la unión NP con la temperatura. La magnitud que devuelve el sensor está en grados Celsius.

En la práctica se utiliza para prevenir averías por sobrecalentamiento

Luz Ambiental

Este sensor detecta la luz ambiental a la que es sometido el dispositivo. Funciona a través de la fotorresistencia. La magnitud que devuelve en sensor está en unidades lux.

Su aplicación más común es la de ajustar el brillo de la pantalla en el dispositivo.

Humedad Relativa

Este tipo de sensor mide la humedad relativa del aire y del ambiente. El sensor realiza esta medición a partir del cálculo del punto de rocío así como de la humedad absoluta. El dispositivo devuelve el porcentaje de humedad (%).

Es usado naturalmente en aplicaciones ambientales o meteorológicas.

2.5. Métodos de detección de badenes mediante móviles

Ya se han realizado trabajos y artículos donde se detectan badenes e incluso agujeros en el pavimento. En esta sección se describen algunos de los sistemas desarrollados.

Uno de los trabajos más interesantes es el *Speed-Breaker Early Warning System* [3] hecho por investigadores de la India para sistemas Android, en el cual se utilizó principalmente el

acelerómetro como método de detección de badenes mediante la creación de vectores de amplitud a partir de las lecturas en los tres ejes del acelerómetro. Este vector de amplitud es obtenido mediante la siguiente fórmula (en donde x, y, z representan la lectura del acelerómetro en cada uno de los ejes):

$$a_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

La mayor aportación de este trabajo es que fue diseñado para ser usado en cualquier *Smartphone* ya que hace uso exclusivo del acelerómetro como sensor detector de badenes. Mediante el uso del vector de amplitud, se garantiza que el dispositivo móvil puede ser utilizado en cualquier posición, sin embargo, no aclara cómo previenen la detección de falsos positivos cuando el teléfono es sometido a otros factores como el frenado del vehículo. Otro problema de este trabajo es que a pesar que distingue entre badenes y agujeros en el pavimento, no aclara de ninguna forma cómo realiza esa distinción, solamente que los vectores de amplitud fueron identificados manualmente para diferentes casos.

Otro trabajo interesante es el de *Nericell: Rich Monitoring of Road and Traffic Conditions using Mobiles Smartphones* [4] desarrollado también por investigadores indios de Microsoft. En este trabajo se detectan agujeros, badenes, frenado y sonidos de las bocinas mediante el uso varios sensores, como el acelerómetro, el micrófono, radio GSM y sensor GPS. Este trabajo también está enfocado en la aplicación de los sensores dejando de lado un poco la parte del servidor donde son recogidos todos los datos. Al contrario del *Speed-Breaker Early Warning System*, este trabajo no escatima en el uso de una gran variedad de sensores, aunque lo hace de forma eficiente para no consumir la batería del dispositivo de forma excesiva. También hace uso de algoritmos de reorientación mucho más complejos para poder utilizar el dispositivo en cualquier posición, como los ángulos de Euler. La mayor aportación es la de detectar condiciones de tráfico caóticas mediante el uso del micrófono, una idea muy original, aunque podría resultar muy imprecisa para determinar que en verdad existe una cantidad de tráfico considerable.

Otro trabajo con una temática similar es el llamado *Real Time Pothole Detection using Android Smartphones with Accelerometers* [5] y desarrollado por investigadores de Latvia también para la plataforma Android. En el artículo se pretenden detectar agujeros en el camino mediante el uso del acelerómetro. Una gran diferencia con respecto a los trabajos mencionados anteriormente es que aquí se omite el problema de la reorientación del dispositivo y se coloca en una posición fija, sin

duda una gran desventaja frente a los trabajos mencionados anteriormente, sobre todo si se pretende que sea utilizado por usuarios finales de la aplicación. Al igual que el *Speed-Breaker Early Warning System,* este sistema hace uso extensivo del acelerómetro para detectar los agujeros en el pavimento; asimismo está diseñado con dispositivos móviles de recursos limitados como en el caso anteriormente mencionado.

La mayor aportación de este trabajo es que distingue los resultados obtenidos dependiendo de la tasa de muestreo del dispositivo utilizado, aunque uno de sus problemas es que no explica como distingue entre agujeros del pavimento y badenes comunes. Por otra parte, al predeterminar la posición del dispositivo es fácil descartar cambios rápidos de la aceleración del vehículo, como el frenado.

Un trabajo que también está enfocado en detectar condiciones de tráfico es *Wolverine: Traffic and Road Condition Estimation using Smartphone Sensors* [6]. Es muy interesante ya que está enfocado especialmente en detectar eventos de frenado del vehículo como un indicador de congestión de tráfico, así como la detección de badenes para determinar el tipo de vía por la que se circula. Los métodos de entrada principales son el acelerómetro y el magnetómetro. A diferencia de *Nericell*, la reorientación del móvil se realiza a través del magnetómetro.

La mayor contribución de este trabajo es que usa técnicas de aprendizaje de máquina para detectar las condiciones de tráfico, aunque los experimentos estuvieron limitados al uso exclusivo de un coche como medio de transporte.

Quizás el mayor problema de este trabajo sea que el magnetómetro no es un sensor que se encuentre presente en todos los dispositivos móviles; asimismo es un sensor susceptible a interferencia magnética por lo que podría ser menos confiable que otros métodos de reorientación propuestos en otros trabajos.

Se aprecia que existen múltiples trabajos relacionados al tema que ocupa este proyecto, y aunque las formas de abordar la problemática son distintas, se tiene que todas las soluciones propuestas utilizan por lo menos el acelerómetro del dispositivo, y también es común el uso del GPS para determinar la posición del vehículo.

3. Entorno tecnológico

Este capítulo describe en primer lugar el modelo de sistema distribuido utilizado para el proyecto, con sus respectivas ventajas y desventajas. Después se escribe con detalle acerca de la plataforma de desarrollo utilizada para la aplicación cliente, Android, que como se verá consta de una arquitectura basada en capas, así como los distintos componentes y peculiaridades del ciclo de vida de una aplicación desarrollada en esta plataforma. Se exponen también las características del dispositivo móvil con el cual se ha trabajado.

El capítulo a su vez, aborda sobre la arquitectura y componentes que forman al Google App Engine. Además se describen las herramientas de desarrollo que utilizan tanto cliente como servidor, que en algunos casos resultan ser convenientemente comunes, como el IDE Eclipse.

3.1. Sistemas Distribuidos

Un sistema distribuido se define como un conjunto de computadores débilmente acoplados en los que se ejecutan aplicaciones compuestas por procesos que intercambian información entre ellos o invocan servicios que realizan otros componentes. En general, es un sistema heterogéneo, compuesto de varias arquitecturas, sistemas operativos y protocolos diferentes y en el cual los procesos gozan de un alto nivel de autonomía.

Para este proyecto en concreto, se sigue el modelo cliente-servidor, el cual permite compartir recursos en red mediante la repartición de tareas entre los proveedores de recursos o servicios (servidores) y los demandantes (clientes). En esta arquitectura la capacidad de proceso se distribuye entre los clientes y los servidores, lo cual ofrece importantes ventajas, como la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita el

diseño y entendimiento del sistema. Este esquema también ofrece alta escalabilidad y facilidad de mantenimiento.

Cabe aclarar que la separación entre cliente y servidor es una separación lógica, esto significa que el servidor no es ejecutado necesariamente sobre sólo una máquina, ni tampoco significa que consta de un solo programa que ofrece un único servicio.

La principal desventaja de este modelo se encuentra en la congestión ocasionada cuando un gran número de clientes realiza peticiones al servidor, y éste no puede darse abasto para tantas por lo que la respuesta tarda en llegar o inclusive no llegar del todo. Esto último no será una desventaja para este proyecto, ya que la computación en la nube a través de Google App Engine ofrece la escalabilidad necesaria.

3.2. La Plataforma Cliente: Android

Como se ha mencionado anteriormente, Android es una plataforma *software* abierta, completa y libre creada por la Open Handset Alliance. Está basada en Linux y se distribuye bajo la licencia Apache 2.0. Actualmente se utiliza principalmente en dispositivos móviles aunque su propósito no se limita únicamente a este mercado.

No es una plataforma *hardware* aunque da completo acceso al mismo para promover la innovación y dar más libertad a los desarrolladores.

El código fuente está disponible y se suministra a todo aquel con la capacidad para producir una imagen (http://source.android.com).

Existen tres componentes de software necesarios para el desarrollo de aplicaciones en Android. El primero es el kit de desarrollo de Java (Java Development Kit ó JDK) estándar, el segundo es el kit de desarrollo de Android (Android SDK) y el tercero el entorno de desarrollo, que en este caso, es Eclipse.

3.2.1. Arquitectura Android

En este apartado se explicará la estructura de este sistema operativo. En Android, la arquitectura está formada por varias capas para facilitar al desarrollador la creación de aplicaciones. Esta distribución permite acceder a las capas más bajas mediante el uso de librerías, de esta forma, el

desarrollador no tiene que realizar programación a bajo nivel para que la aplicación haga uso de los componentes de *hardware* del dispositivo.

Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, a esta clase de arquitectura se le conoce también como *stack* o pila. A continuación se presenta el diagrama de la arquitectura de Android, obtenido del sitio oficial de desarrolladores:

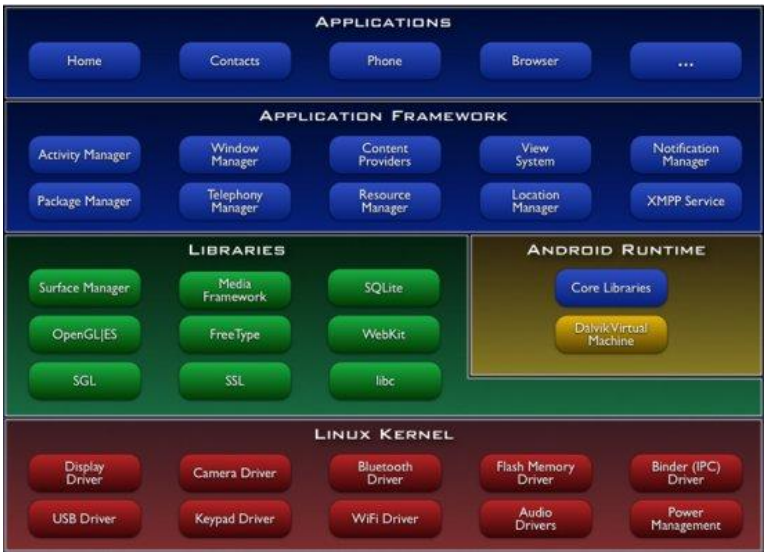


Figura 3.1 Arquitectura Android

Capa Linux Kernel

El núcleo actúa como una capa de abstracción entre el *hardware* y el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. De esta manera se evita la dificultad de conocer las características exactas de cada dispositivo. Todos los componentes de *hardware* del teléfono disponen de un controlador (o *driver*) dentro del kernel que permite utilizarlo por medio del *software*.

El kernel también es el encargado de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación, etc.

Cabe destacar que el kernel elimina algunas de las características comunes de Linux, por ejemplo no se soporta la GlibC, sino una variante llamada Bionic. Tampoco hay un sistema de ventanas nativo.

La gestión de energía es propia de Android, pero está basada en el sistema de gestión de energía de Linux. Las aplicaciones informan al kernel de sus necesidades a través de una librería que se ejecuta en el espacio de usuario y se utilizan mecanismos de cerrojo para servir a dichas aplicaciones.

Cuenta con un controlador de comunicaciones IPC entre aplicaciones y servicios, también llamado Binder que facilita el intercambio de objetos entre procesos, gestionándolos como direcciones en el espacio de memoria de los procesos.

La razón por la que utiliza Linux es porque su kernel está probado. La fiabilidad es más importante que las prestaciones en el ámbito móvil. Los dispositivos están diseñados para ofrecer (por encima de todo) un buen servicio de telefonía móvil. El sistema desacopla el *hardware* del *software* a través de una HAL (*Hardware Abstraction Layer*). Cuando aparecen en el mercado nuevos accesorios, éstos sólo requieren de nuevos controladores para poder ser utilizados.

Capa Librerías Nativas

La siguiente capa situada justo arriba del kernel está compuesta por las librerías nativas de Android. Están escritas en C o C++ y compiladas para el hardware específico del teléfono. Estas normalmente están hechas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las librerías es el de proporcionar la funcionalidad adecuada a las aplicaciones para tareas frecuentes, evitando tener que codificarlas continuamente a la vez de garantizar que se lleven a cabo eficientemente.

Entre las librerías habituales se encuentran OpenGL (motor gráfico), Bibliotecas multimedia (formatos de audio, imagen y video), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

Capa Entorno de ejecución Android

El entorno de ejecución de Android también está compuesto por librerías, por lo que no se considera una capa en sí mismo. Estas librerías contienen las funcionalidades habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, que se encuentra optimizada para tener un requerimiento bajo de memoria y está diseñada para

permitir la ejecución simultánea de varias instancias de la máquina virtual. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. Se debe aclarar que Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el *bytecode* Java. Java se usa únicamente como lenguaje de programación, por lo que los ejecutables que se generan con el SDK de Android tienen la extensión ".dex" que es específico para Dalvik, y por ello no se puede ejecutar Java en Android ni viceversa.

Capa Framework de Aplicaciones

Esta capa se forma por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik. En el diagrama se encuentran las siguientes:

1. *Activity Manager*. Se encarga de administrar la pila de actividades de cualquier aplicación así como su ciclo de vida.

2. *Windows Manager*. Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.

3. *Content Provider*. Crea una capa que encapsula los datos que se compartirán entre aplicaciones para tener control sobre cómo se accede a la información.

4. *Views*. Las vistas son elementos que ayudan a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.

5. *Notification Manager*. Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Esta librería también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono en caso de tenerlos.

6. *Package Manager*. Permite obtener información sobre los paquetes instalados en el dispositivo, además de gestionar la instalación de nuevos paquetes. El paquete se refiere a la forma en que se distribuyen las aplicaciones en Android, éste contiene el archivo .apk, que a su vez incluye los archivos .dex con to-dos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.

7. *Telephony Manager*. Con esta librería se pueden realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

8. *Resource Manager*. Con esta librería se pueden gestionar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o *layouts*.

9. *Location Manager*. Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles.

10. *Sensor Manager*. Permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.

11. *Cámara*. Con esta librería se puede hacer uso de la(s) cámara(s) del dispositivo para tomar fotografías o para grabar vídeo.

12. *Multimedia*. Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

Capa de Aplicaciones

En la última capa se encuentran las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

3.2.2. Componentes y ciclo de vida de una aplicación

Las aplicaciones en Android constan de varios componentes que son definidos en el fichero de manifiesto o *Android Manifest.* Estos tipos componentes pueden ser *Activity*, *Service*, *Broadcast Receiver* y *Content Provider*.

Una *Activity* es un componente que suele corresponderse con una interfaz gráfica desde la que pueden realizarse un conjunto de acciones. Un conjunto de *Activities* crea la interfaz gráfica de usuario de la aplicación, aunque cada *Activity* es independiente del resto. El intercambio de información entre las *Activities* se hace mediante *Intents*. Un *Intent* podría verse como una representación abstracta de una operación a realizar, son definidos con base en una acción, la información necesaria para llevar a cabo dicha acción y de quién debe llevarla a cabo.

Los componentes de tipo *Service* realizan operaciones de larga duración y no poseen interfaz gráfica de usuario. Éstos se ejecutan en segundo plano, pero bajo el hilo principal del proceso que los alberga. Los *Services* deben utilizarse solamente cuando se requiera realizar una tarea de larga duración que no requiera de la interacción del usuario con la aplicación.

Los *Broadcast Receiver* son componentes que tienen el propósito de recibir y reaccionar a eventos globales, como la recepción de un mensaje o una llamada.

Los *Content Provider* son componentes que le permiten el almacenamiento y la recuperación de datos entre múltiples aplicaciones. Un ejemplo de esto podría ser una lista de contactos. Por tanto no deberían utilizarse si no se planea reusarlo para múltiples aplicaciones.

Cuando se crea una aplicación Android, es de vital importancia tener en cuenta su ciclo de vida. Toda aplicación Android se ejecuta en su propio proceso Linux y posee su propia instancia de la máquina virtual Dalvik. Cuando una aplicación necesita de la intervención de uno de sus componentes, entonces Android se asegura de que el proceso asociado a la aplicación esté en ejecución, arrancándolo en caso de necesidad. Android también verifica que la instancia adecuada del componente esté disponible, y en caso contrario la crea.

El proceso Linux que encapsula a las aplicaciones es creado cuando el código de la misma debe ejecutarse y permanece en ejecución hasta ya no es necesario o el sistema necesita de los recursos que otras aplicaciones ocupan.

El ciclo de vida de una aplicación Android no se controla desde la misma aplicación, sino desde la plataforma. Esto lo hace con base en las partes de la aplicación que el sistema sabe que están ejecutándose, la cantidad de memoria disponible en el sistema y la importancia de dichas partes para el usuario. El ciclo de vida de los componentes consta de un principio y un fin, mientras tanto su vida transita entre estados activos e inactivos, y en el caso de las *Activities* como visible e invisible.

Las *Activities* en el sistema se gestionan a través de una pila de *Activities*. Cuando una nueva *Activity* es iniciada, se pone en lo más alto de la pila y es por tanto la *Activity* que se ejecuta en ese momento. Toda *Activity* previa permanece abajo en la pila, y no saldrá de ese plano de nuevo hasta que la *Activity* más reciente salga de la pila.

En la figura 3.2 es presentado un diagrama con los posibles estados de una *Activity*. Los óvalos representan los diferentes estados en los que se puede encontrar una *Activity*. Los rectángulos representan métodos que pueden ser programados y en los que es posible programar tareas que deban realizarse cuando una *Activity* se mueva entre estados. Esto debe realizarse si es que se quiere contar con una aplicación funcional, que no siga consumiendo recursos del sistema cuando ya no es necesario y para guardar el estado de una *Activity* cuando se regrese a ella.

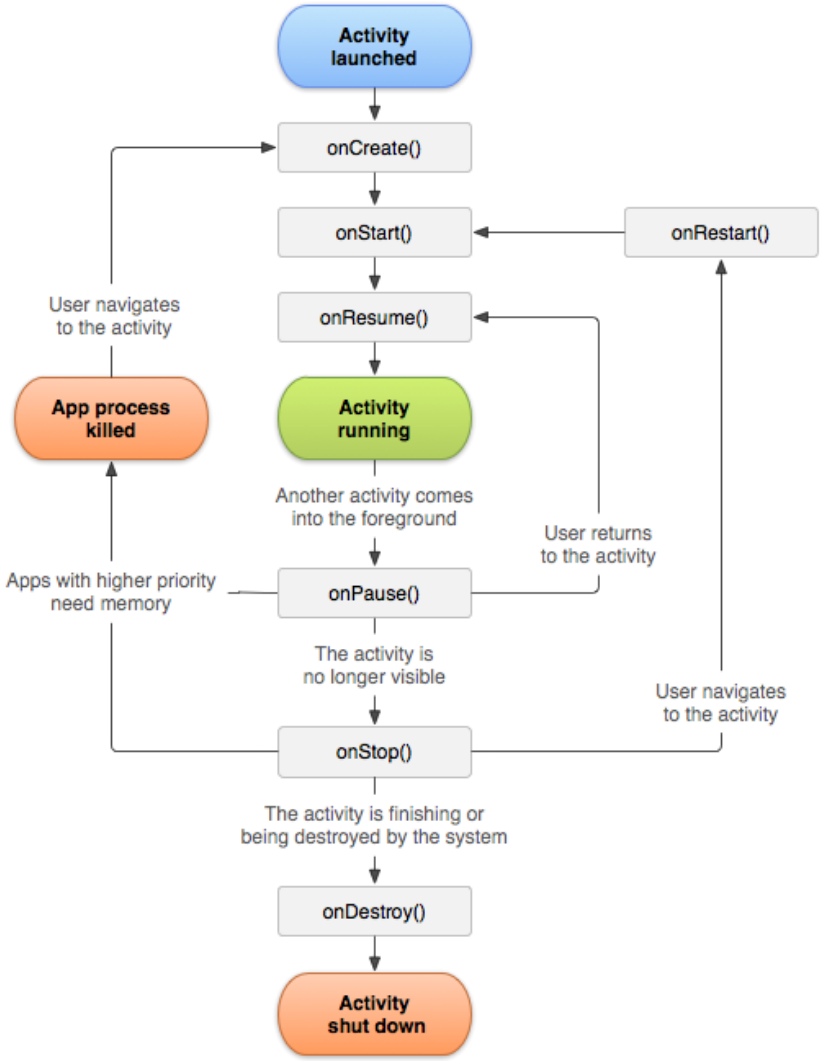


Figura 3.2 Ciclo de vida de una *Activity*

3.2.3. Java Development Kit JDK

El Java Development Kit, también conocido como JDK, es un software con las herramientas de desarrollo necesarias para la creación de programas en Java. Fue desarrollada por Sun Microsystems, ahora propiedad de Oracle. El JDK está disponible para una gran cantidad de sistemas operativos, incluido el utilizado para el desarrollo de este proyecto, Windows 7. Se le considera el kit de desarrollo de *software* más utilizado en la actualidad.

La versión más actual de este kit de desarrollo es la versión 7, pero la versión utilizada para la creación de este proyecto es la 6 ya que se trata de una versión más estable y fiable.

3.2.4. Android SDK

El SDK de Android, proporciona las librerías API y las herramientas de desarrollo necesarias para crear, probar y depurar aplicaciones para Android.

El kit de desarrollo consta de un depurador de código, librerías, simulador de dispositivos, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux, Max OS X 10.4.9 o posterior, y Windows XP o posterior. La plataforma integral de desarrollo soportada oficialmente es Eclipse junto con el complemento ADT (Android Development Tools *plugin*). Además, por medio del kit de desarrollo pueden controlarse dispositivos Android que estén conectados al ordenador. En el caso de este proyecto se utiliza la plataforma Windows 7, así como el IDE de Eclipse con ADT integrado.

El Android Debug Bridge (ADB) es un conjunto de herramientas incluidas en el paquete de Android SDK que permiten depurar las aplicaciones en tiempo real. El ADB se accede normalmente a través de la interfaz de línea de comandos.

Las actualizaciones del SDK se realizan mediante el Android SDK Manager. El SDK también soporta versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y realizarse pruebas de compatibilidad.

Adicionalmente, existe el Android NDK (Native Development Kit), el cual permite implementar partes de la aplicación desarrollada en código C y otros lenguajes nativos. No será necesario instalar el NDK ya que las APIs del SDK proporcionan la funcionalidad adecuada para este proyecto.

3.2.5. Eclipse

Eclipse es un entorno de desarrollo integrado, mantenido por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Eclipse soporta el uso de diversos lenguajes de programación para el desarrollo de proyectos, aunque el lenguaje más utilizado con diferencia es Java.

Como se mencionó anteriormente, la plataforma oficial de desarrollo soportada para proyectos Android es Eclipse, por lo que la mera instalación del *plugin* ADT otorga al desarrollador la capacidad de comenzar proyectos Android y hacer uso del SDK.

Adicionalmente, para el desarrollo de la plataforma servidor, Google proporciona un *plugin* para Eclipse con el conjunto de herramientas de desarrollo necesarias para construir, optimizar y desplegar aplicaciones en Google App Engine. De esta forma se tiene un entorno de desarrollo que sirve para la creación de todos los componentes de este proyecto.

3.2.6. Samsung Galaxy ACE

El dispositivo sobre el cual se ha trabajado para el desarrollo la aplicación, así como todas las pruebas de este proyecto es un Samsung Galaxy ACE. Es un teléfono inteligente de gama media desarrollado por Samsung y que cuenta con el sistema operativo Android en su versión 2.3.4 "Gingerbread", que hasta el 8 de Julio de 2013, es la distribución instalada en la mayoría de dispositivos Android con un 34.1%.

En la figura 3.3 puede apreciarse la cuota del mercado Android por distribución:

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.4%
2.2	Froyo	8	3.1%
2.3.3 - 2.3.7	Gingerbread	10	34.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.3%
4.1.x	Jelly Bean	16	32.3%
4.2.x		17	5.6%

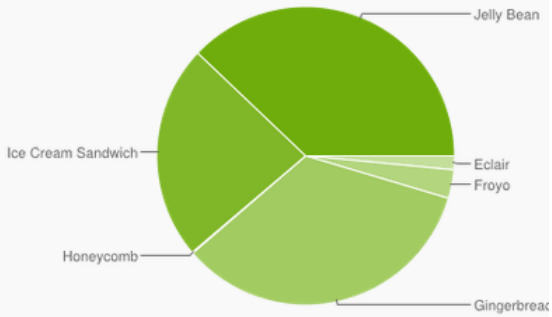


Figura 3.3 Cuota del Mercado Android por versión del sistema operativo

Es importante señalar que por el uso de la versión 2.0 de la API de mapas de Google, la cual limita la versión de Android sobre la cual puede funcionar. Por tanto esta aplicación requiere de una versión mínima instalada en el cliente de Android 2.2.

Es un dispositivo 3.5G, ofrece quad-band GSM y soporte a dos bandas HSDPA (900/2100) a 7,2 Mbit/s. La pantalla es una TFT LCD del tipo táctil capacitiva de 3.5 pulgadas y tiene una resolución

HVGA de 320x480. Cuenta con un procesador Qualcomm MSM7227 de 800 MHz y un GPU Adreno 200. Asimismo tiene 384 MB de memoria RAM. Respecto a la conectividad, también soporta Bluetooth v2.1 y WiFi 802.11 b/g/n.

Figura 3.4 Samsung Galaxy ACE

Para poder desarrollar aplicaciones en cualquier dispositivo real con Android, resulta imprescindible activar la opción de "Depuración USB" dentro de las opciones de las aplicaciones y desarrollo del aparato, de otra forma el dispositivo no será reconocido cuando se le intente cargar software.

Figura 3.5 Activación de opción depuración de USB

Es un dispositivo muy funcional y responsivo, si bien existe la limitación de que al realizar este proyecto no se contaba con un plan de datos, por lo que la comunicación con el servidor de Google App Engine no puede realizarse en tiempo real, sino de forma asíncrona mediante WiFi.

3.3. La Plataforma Servidor: Google App Engine

3.3.1. Arquitectura Google App Engine

Como ya se ha mencionado, Google App Engine o GAE, es una plataforma de servicios de cómputo en la nube para el desarrollo y alojamiento de aplicaciones web en los centros de datos gestionados por Google. Las aplicaciones desarrolladas permiten el uso de un dominio propio o uno proporcionado por Google. Los lenguajes de programación soportados al día de hoy en GAE son Java y Python.

La plataforma GAE está provista de una capacidad total de escalabilidad automática. Conforme el número de peticiones de la aplicación aumenta, ésta es virtualizada en múltiples servidores, de manera que se escala de manera automática y sin que el desarrollador tenga que hacer alguna gestión al respecto.

La arquitectura de GAE está compuesta por los siguientes elementos: el entorno de ejecución, el almacén de datos y los servicios.

Entorno de ejecución GAE

El entorno de ejecución tiene tres componentes principales, los *Frontends*, los servidores de archivos estáticos y los servidores de aplicaciones. Los *Frontends* se encargan de recibir la petición del cliente y distribuirla por medio del balanceo de carga a los servidores ejecutando la aplicación.

Los servidores de archivos estáticos sirven ficheros que son mantenidos de manera fija durante el funcionamiento normal del servicio (como ficheros HTML ó imágenes).

Los servidores de aplicaciones atienden las peticiones de los clientes en sí. Éstos ejecutan sus servicios en modo *sandbox* y poseen un tiempo de respuesta máximo de 30 segundos. Los servidores de aplicaciones se ejecutan en entornos Java y Python y hacen uso del almacén de datos para llevar a cabo su labor.

El almacén de datos

Existen dos tipos de servicio de almacenamiento de datos, el *Datastore* o almacén de datos y el *Memcache*. El primero es el servicio de almacenamiento permanente para aplicaciones web, en el

sentido más estricto, no se trata de una base de datos relacional/jerárquica, sino que las aplicaciones almacenan la información en entidades a su vez compuestas por una o más propiedades. Sin embargo, se pueden realizar consultas y modificar los datos almacenados por medio de transacciones.

Memcache se trata de un servicio de almacenamiento a corto plazo. Es muy conveniente de utilizar en el caso de consultas constantes ya que resulta muy rápido en comparación con las consultas realizadas al *Datastore*.

Servicios

Por último se encuentran los servicios, App Engine proporciona una amplia variedad de servicios para realizar operaciones comunes mientras se gestiona una aplicación, por ejemplo peticiones HTTP a otros servidores en internet, servicios de correo electrónico o de mensajería instantánea.

3.3.2. El entorno Java en GAE

El SDK Java de GAE está disponible como una descarga independiente o como se mencionó anteriormente, un *plugin* para Eclipse. La creación de aplicaciones para el entorno de ejecución GAE es realizado mediante las herramientas comunes de desarrollo y API estándar de Java. Las aplicaciones interactúan con el entorno por medio de los *Servlet* Java. Un *Servlet* es una clase Java que puede recibir peticiones (por lo general HTTP) y generar una salida (normalmente HTML, WML o XML).

También se pueden utilizar tecnologías Web comunes como *Java Server Pages* (JSP), los cuales son mucho más ligeros que los *Servlets*, aunque no más potentes. Cabe mencionar que el SDK soporta el uso de aplicaciones que utilicen ya sea la versión 6 ó 7 de Java.

El entorno de desarrollo incluye la plataforma Java SE Runtime Environment (JRE) versión 6 junto con todas sus librerías. Las restricciones del entorno *sandbox* son implementadas en la JVM. Una aplicación puede utilizar cualquier característica o *bytecode* JVM, siempre que no rompa con las restricciones del *sandbox*. Por ejemplo, un *bytecode* que intente abrir un *socket* o escribir en un archivo producirá una excepción en tiempo de ejecución.

Las aplicaciones tienen acceso a los servicios del App Engine a través de los APIs estándar de Java. Respecto al almacén de datos, el SDK incluye implementaciones de la interfaz para Java Data Objects (JDO) y Java Persistence API (JPA). Asimismo, las aplicaciones pueden utilizar la API JavaMail para enviar mensajes de correo electrónico por medio del servicio de correo de Google App Engine. La HTTP API java.net accede al servicio URL FETCH del App Engine.

3.3.3. El almacén de datos y Java Data Objects

A diferencia de las bases de datos relacionales tradicionales, el almacén de datos de App Engine utiliza una arquitectura distribuida para escalar conjuntos de datos de gran tamaño automáticamente. Aunque la interfaz de almacén de datos tiene muchas de las características de las bases de datos tradicionales, se diferencia de ellos en la forma en que describe las relaciones entre los objetos de datos. Las entidades del mismo tipo pueden tener diferentes propiedades y diferentes entidades pueden tener propiedades con el mismo nombre, pero con diferentes tipos de valor.

Es importante decir que Google App Engine impone ciertas limitaciones cuando se trata de una aplicación gratuita, éstas son mostradas en la figura 3.6, aunque afortunadamente son suficientes para este proyecto.

Limite	Cantidad
Tamaño máximo de entidad	1 megabyte
Tamaño máximo de transacción	10 megabytes
Número máximo de entradas de índice para una entidad	20000
Número máximo de bytes para índices compuestos por una entidad	2 megabytes

Figura 3.6 Limitaciones de almacenamiento en GAE

EL SDK de Google App Engine soporta dos estándares de almacenamiento de datos, Java Data Objects (JDO) y Java Persistent API (JPA). El estándar utilizado en este proyecto es JDO, el cual está basado en la plataforma DataNucleus Access, la implementación de referencia de código abierto para JDO 2.3.

Cada objeto guardado por JDO se convierte en una entidad en el almacén de datos de App Engine. El tipo de entidad se deriva del nombre simple de la clase. Cada campo persistente de la clase Java representa una propiedad de la entidad, por tanto el nombre de la propiedad es igual al nombre del campo.

JDO permite que las interfaces realicen tareas de anotación de objetos Java, la recuperación se hace mediante consultas y la actualización mediante transacciones. Mediante las consultas se obtienen entidades del almacén de datos que reúnen ciertas condiciones. Este lenguaje de consultas se denomina JDOQL.

3.3.4. Consola de administración

Es una herramienta web muy útil que le otorga al desarrollador acceso público y control completo de la aplicación desplegada en GAE. Entre otras cosas, la consola de administración puede ser utilizada para:

- Realizar la configuración básica (cambiar el título de las aplicaciones, establecer la caducidad de las cookies, establecer opciones de autenticación, etc.).
- Configurar las opciones de gestión de costes y rendimiento.
- Ver servicios configurados.
- Establecer un nuevo dominio host.
- Desactivar escritura del almacén de datos.
- Deshabilitar o eliminar la aplicación.
- Administrar el almacén de datos, realizar copias de seguridad, restaurar, copiar y borrar datos.
- Dividir el tráfico entre las diferentes versiones de la aplicación.
- Ver peticiones y registros de errores, así como analizar el tráfico.
- Administrar colas de tareas, lo que permite hacer una pausa, depuración y eliminación.
- Ver estadísticas *Memcache*, vaciar la caché, ver y editar sus valores.

4. Análisis de la Solución

En este capítulo se describe con detalle el modelo de funcionamiento de la geolocalización y de la detección de badenes por medio del acelerómetro. Es necesario establecer un modelo bajo el cual se obtenga una ubicación geográfica rápida y lo más precisa posible ya que de nada sirve detectar un badén si no se puede determinar posteriormente la ubicación del mismo. También es necesario notificar con antelación al usuario de la cercanía del badén, así que la velocidad y precisión de la ubicación juegan un papel igual de importante en este aspecto.

El otro elemento analizado en este capítulo es el modelo de obtención de lecturas del acelerómetro para determinar que se ha pasado por un badén.

4.1. Optimización de la geolocalización

Ya se ha mencionado que existen diferentes formas de obtener la localización en un dispositivo móvil. En Android existen tres proveedores de localización, GPS (GPS, AGPS), red (AGPS, Cell-ID, WiFi MACID) y el proveedor pasivo (Cell-ID, WiFi MACID).

Existen dos modos principales en la aplicación cliente, el de captura y el de prevención. Para ambos modos es esencial obtener la posición del dispositivo, por un lado para enviar los datos con la mayor precisión posible al servidor, y por otro asegurar que los badenes solicitados al servidor sean los que realmente rodean al usuario para así como prevenir con prontitud de la aproximación de los mismos.

Dados estos requerimientos se han considerado dos tipos de proveedores de localización, el de GPS y el de red. No se utiliza el método pasivo ya que este funciona recibiendo las actualizaciones

de ubicación que otras aplicaciones reciben en el mismo dispositivo, así que se puede prescindir de este método.

Ambos modos principales de la aplicación comparten un mismo flujo de acciones a partir de los cuales se obtiene la ubicación geográfica. Dicho flujo se representa en la figura 4.1.

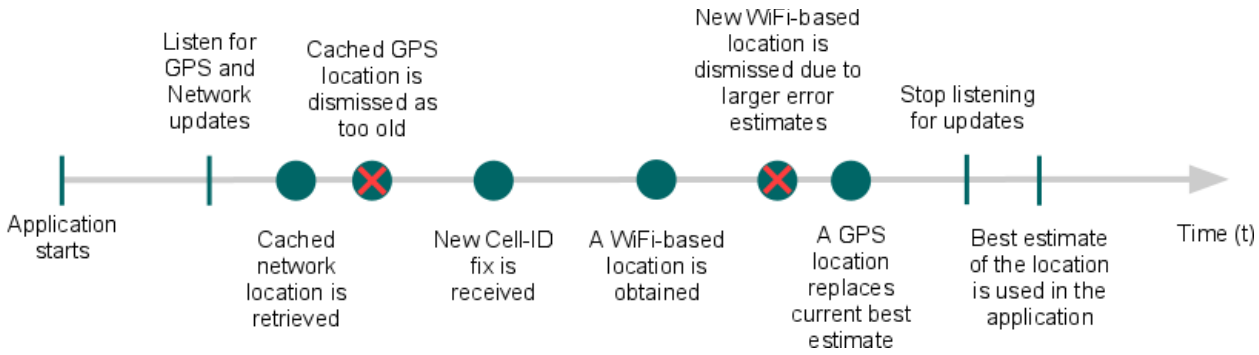


Figura 4.1 Flujo de acciones para obtener ubicación geográfica

Básicamente después de que se inicializa la aplicación o *Activity*, se obtiene la última ubicación conocida del dispositivo. Luego se recibe la ubicación por red y después se comienzan a recibir actualizaciones del proveedor GPS. La clave se encuentra en determinar cuál es la mejor ubicación de las que se dispone. Por último, cuando la aplicación termina, se dejan de recibir actualizaciones de ubicación y así dejar de consumir recursos.

Una vez expuesto el flujo de acciones que sigue la aplicación en sus modos principales, hay que definir ciertas cosas y realizar algunas acciones para llevarlas a cabo. En primer lugar hay que otorgarle a la aplicación los permisos adecuados para poder acceder a los servicios de geolocalización. Éstos son establecidos en el fichero *Manifiest* de la aplicación Android. Los permisos especifican si la aplicación hace uso de geolocalización de grano fino o grueso. Como se ha establecido que la aplicación necesita el mayor grado de precisión posible, se añade el permiso de grano fino o ACCESS_FINE_LOCATION, el cual hace uso tanto del proveedor de red como de GPS.

Una vez que se han otorgado los permisos pertinentes a la aplicación, se ha procedido a definir un *Location Manager*, una clase que permite acceder a los servicios de localización del sistema, los cuales facilitan la adquisición de actualizaciones periódicas de la ubicación geográfica del dispositivo. Seguido a este acto, se define un *Location Listener*, el cual recibe las notificaciones del *Location Manager* cuando la ubicación ha cambiado. Este *Listener* debe incorporar cuatro métodos distintos y que la aplicación utilizará.

- **onLocationChanged**: Se utiliza para establecer la ubicación del dispositivo y actualizar los datos que aparecen en pantalla. Desde este método se llama a otro llamado *getBetterLocation*, que implementa el algoritmo descrito en la figura 4.2 para determinar si una nueva ubicación es más fiable que la anterior. Es con este método que también se establece si el dispositivo ha estado en movimiento mediante la comprobación de una distancia mínima de 1 metro entre cada actualización para considerar que existe desplazamiento. De igual manera, si la última actualización toma más de treinta segundos en llegar, la aplicación pregunta si el dispositivo se ha movido por lo menos 10 metros entre la última y la nueva actualización, en caso negativo se considera que no está en movimiento.
- **onStatusChanged**: Este método sirve para establecer qué hacer en el caso que el dispositivo abandone el área de servicio. Android distingue entre tres posibles cambios de estado, "Fuera de servicio", "Temporalmente Fuera de Servicio", "Disponible". Si la aplicación se encuentra en cualquiera de los estados, guarda en el log y posteriormente notifica.
- **onProviderEnabled**: Para especificar que acción tomar si se habilita un proveedor en mitad de la aplicación. En el caso de esta aplicación se notifica y guarda en el *log* que se ha habilitado ya sea el proveedor de red o GPS.
- **onProviderDisabled**: Para especificar que acción tomar si se deshabilita un proveedor en mitad de la aplicación. En el caso de esta aplicación se crea un hace un *Intent* con el cual el usuario podrá activar al proveedor de red o GPS si así lo desea.

Como se ha mencionado ya, la geolocalización no es una tarea sencilla, la ubicación obtenida puede contener errores y ser imprecisa sin importar de cuál de las dos fuentes se reciba. Como se observa en la figura 4.1, es necesario determinar criterios para obtener la mejor ubicación posible de acuerdo a las necesidades de esta aplicación. Los criterios que se han definido son el tiempo o antigüedad de la ubicación recibida y el radio de precisión de la misma.

Para ello se ha implementado un método llamado *getBetterLocation*, el cual sigue el siguiente diagrama para determinar si una nueva ubicación geográfica es mejor que la anterior.

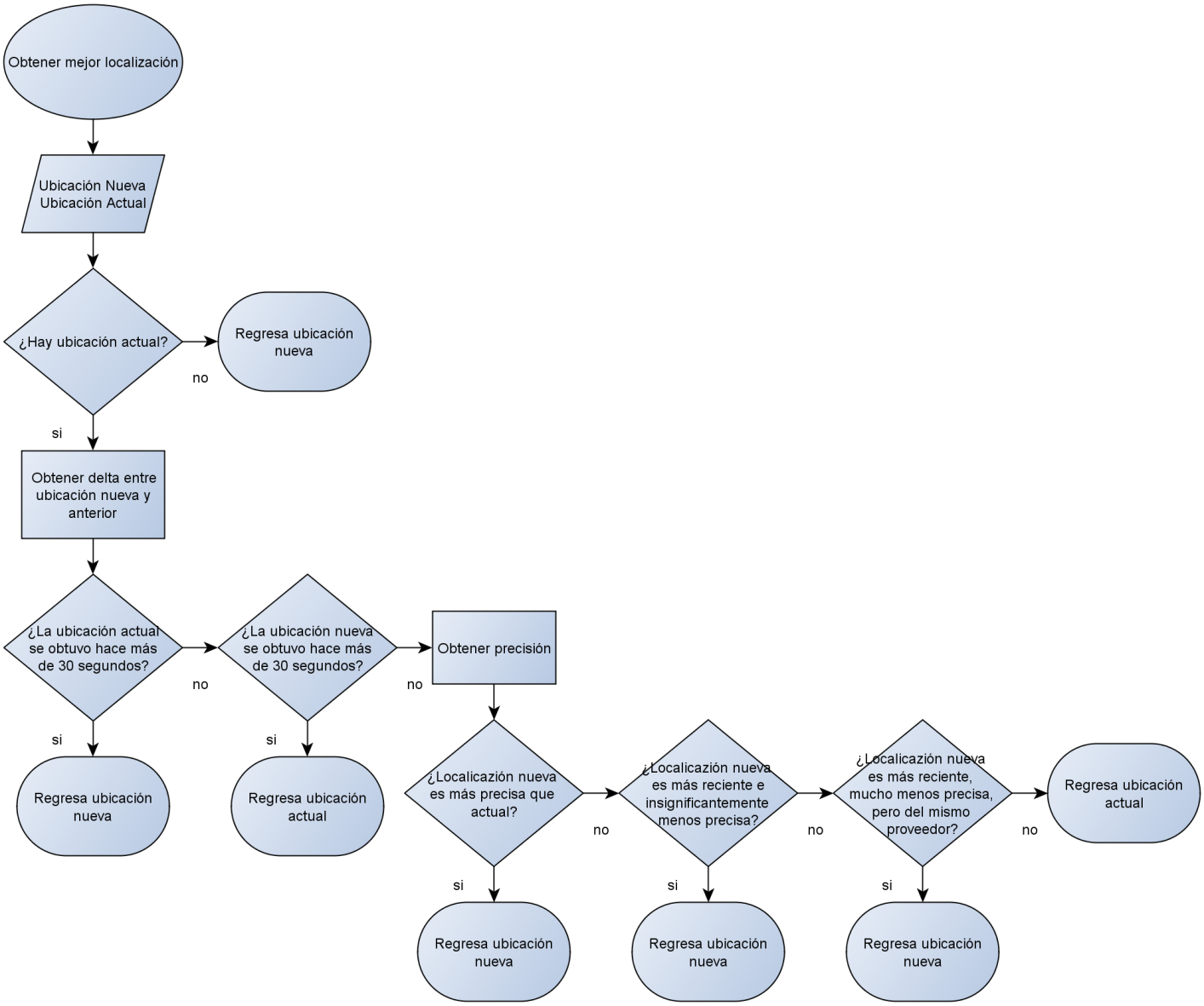


Figura 4.2 Diagrama de flujo para determinar la mejor ubicación recibida

A grandes rasgos, primero comprueba si existe o no una nueva ubicación, si no se posee ubicación alguna se devuelve la nueva. Para saber si es significativamente nueva se ha definido una constante con un valor en tiempo de 30 segundos, si han pasado más de 30 segundos desde la última ubicación entonces se toma la nueva. Si la nueva localización es más vieja que eso, entonces se descarta ya que es probable que sea menos precisa.

En caso de que ambas ubicaciones tengan una antigüedad mayor a treinta segundos, se toma como prioritaria la precisión de la localización. Android proporciona un método para calcular la

precisión en metros de una ubicación recibida a través del *Listener*, es llamado *getAccuracy*. Android define a esta precisión como un radio de exactitud con un 68% de confianza. En otras palabras, si se dibuja un círculo con un centro basado en la latitud y longitud de este lugar, y un radio igual a la precisión, entonces hay una probabilidad del 68% de que la verdadera ubicación se encuentre dentro del círculo. Se obtiene un delta entre la precisión de ambas ubicaciones, si es negativa, la ubicación es más precisa, si es positiva es trivialmente menos precisa y si es mayor a 200 es mucho menos precisa.

Cabe destacar que se espera que la aplicación sea utilizada primordialmente dentro de un coche, por lo que el consumo de energía no ha sido considerado un factor determinante a la hora de desarrollar el modelo. Esto también ha permitido indicarle al *Location Manager* que la solicitud de ubicación se haga lo más pronto, pero razonablemente posible.

Existen dos parámetros mediante los cuales se indica el intervalo de obtención de la nueva ubicación. El primero es el tiempo mínimo, el intervalo de petición de una nueva ubicación nunca será menor al tiempo mínimo especificado, pero podría ser mucho mayor dependiendo del proveedor. En el caso de que el proveedor sea la red WiFi se observó que este intervalo no era menor de 30 segundos en ningún caso. Por otra parte, al proveedor GPS podía solicitársele la nueva ubicación cada segundo.

El otro parámetro es la distancia mínima en metros, de esta forma el proveedor solamente envía la actualización de la ubicación cuando esta haya variado por lo menos en la distancia y el tiempo mínimo especificado, por lo que el parámetro primario resulta ser el tiempo mínimo y luego la distancia.

En el caso de la detección se estableció un tiempo mínimo para actualización de 1 segundo y la distancia mínima en 1 metro. En el modo de prevención se le dio un poco más de margen al tiempo de actualización y ha quedado en 3 segundos, esto es porque se necesita visualizar el mapa en el dispositivo y la aplicación podría entorpecerse si la actualización es demasiado seguida. La distancia mínima es igual a la del modo de detección, es decir de 1 metro para poder actualizar.

4.2. Detección de badenes mediante sensores

Como ya se ha descrito en la sección 2.5, existen estudios previos que utilizan diferentes tipos de sensores [3] [4] [5] [6], pero para este proyecto se ha elegido trabajar exclusivamente con el acelerómetro en lo que respecta a la detección de badenes, ya que este sensor es común en la abrumadora mayoría de los dispositivos móviles inteligentes actuales. Como se explicó con anterioridad, el acelerómetro detecta las fuerzas de aceleración a las que se ve sometido el dispositivo en los 3 ejes del sistema de coordenadas, incluyendo la fuerza de gravedad, por lo que la aceleración del dispositivo se ve alterada por los cambios en la velocidad de la masa.

Cuando el dispositivo se encuentre en reposo, la lectura del acelerómetro cuyo eje se ve afectado por la aceleración de la gravedad, tendría la tendencia a ser equivalente a la misma gravedad (dependiendo de la posición en la que se encuentre el dispositivo). La aproximación de este trabajo es el de encontrar una variación significativa en la lectura del acelerómetro para determinar si es que se está circulando sobre un badén o no. Para ello se han realizado experimentos con el fin de analizar las características que tienen las lecturas cuando se pasa sobre uno.

También se busca que el dispositivo funcione en prácticamente cualquier posición, por lo que la fuerza de gravedad no será substraída de las lecturas del acelerómetro para así determinar la rotación del dispositivo.

En primer lugar se instancia la clase *Sensor Manager*, la cual se usa para acceder a los servicios de sensor. También se implementa en la *Activity* la interfaz *Sensor Event Listener*, la cual proporciona los métodos que hay que sobrescribir cuando se obtiene una nueva lectura del sensor y de cuando cambia la precisión. En el caso de este proyecto solamente se sobrescribe el método de lectura del sensor ya que no es necesario cambiar la precisión.

Cuando se registra el sensor acelerómetro en el *Sensor Manager*, es posible darle pistas al sistema sobre la tasa de muestreo. Se habla de dar pistas porque aunque se especifique el tiempo de espera antes de una nueva muestra, no forzosamente se toma una nueva lectura ya que puede que no haya cambiado el valor del sensor.

Existen cuatro tasas de muestreo que Android tiene por defecto, *Normal, UI, Game y Fastest*. Cada una corresponde a la tasa que la documentación recomienda para diferentes tipos de aplicación,

siendo la primera la más lenta. En este proyecto se ha escogido la más lenta *Normal*, que en promedio equivale a cinco muestras por segundo, suficiente para el propósito de los experimentos.

Una vez definidos estos elementos en la aplicación, el dispositivo móvil puede tomar las lecturas del acelerómetro, que serán tres, correspondientes a los ejes del dispositivo.

Como ya se ha mencionado, se busca utilizar el dispositivo en cualquier posición, por lo que se determina la orientación del dispositivo con base en la lectura del eje que es afectado por la fuerza de gravedad como se aprecia en la figura 4.3.

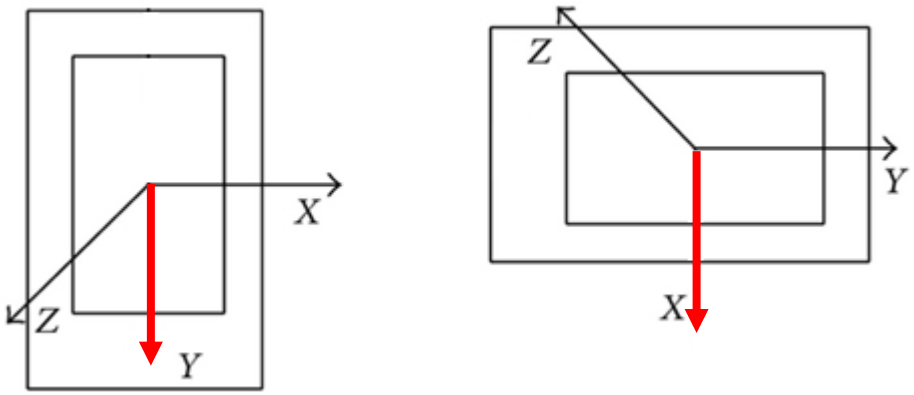


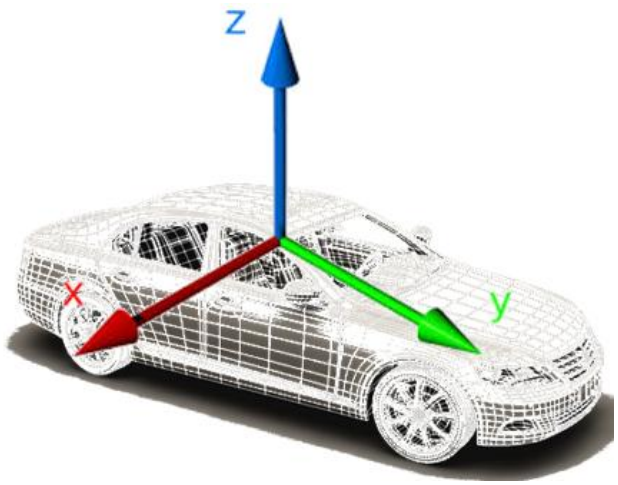
Figura 4.3 Determinación de orientación de dispositivo a partir del eje afectado por la gravedad

Además, la lectura del eje que sea afectado por la gravedad también será la que se utilice para realizar un análisis y decidir si se ha pasado por un badén. Las lecturas sobre los otros ejes se descartan ya que corresponden a fuerzas a las que el dispositivo es sometido y que no son de interés para el objetivo de este proyecto. De tomarlas en cuenta, es muy probable ocasionar falsos positivos cuando el coche se ve sometido a cambios bruscos de aceleración como pueden ser el frenado o la circulación por rotondas.

4.2.1. Recolección de datos

Antes de implementar el sistema, es necesario realizar una recolección de lecturas del eje afectado tanto por la gravedad como por la circulación sobre los badenes, para posteriormente identificar las características de las mismas con el objetivo de sintetizar un algoritmo de detección. Para ello se condujo por el campus de la Universidad Politécnica de Valencia, el cual es un terreno regular y uniforme, con una cantidad muy alta de badenes.

El dispositivo móvil fue colocado en el tablero del coche de forma horizontal y con la pantalla apuntando hacia arriba con el fin de manipular la aplicación más fácilmente de ser necesario. La correspondencia de las coordenadas del dispositivo con el automóvil de la forma en que ha sido desplegado puede observarse en la figura 4.4.



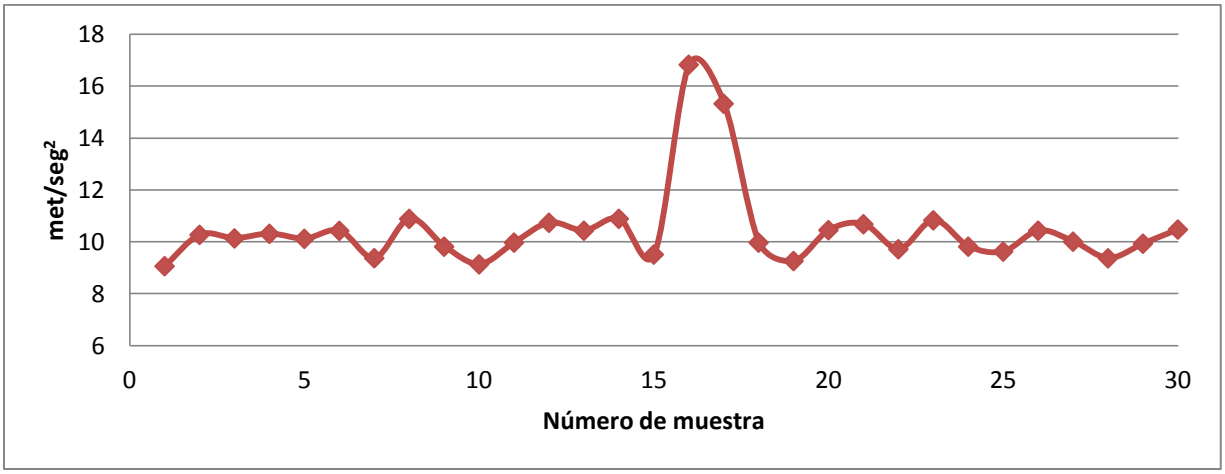
4.4 Correspondencia de coordenadas del dispositivo con el automóvil.

Una vez considerados todos estos elementos y desplegado el dispositivo en el automóvil, se procedió a circular a lo largo del campus, capturando las lecturas del acelerómetro, el tiempo exacto y las coordenadas geográficas en todo momento para posteriormente hacer la correspondencia entre las lecturas y el badén por el que se pasó. Después de haber concluido el recorrido se identificaron dos tipos de badenes, angostos y anchos, cuyas formas son representadas en la figura 4.5.

4.5 Tipos de badenes. Angosto y ancho respectivamente.

Naturalmente, pasar por los diferentes tipos de badenes a diferentes velocidades e incluso pasar por ellos en diagonal provoca características distintas en las lecturas tomadas. A continuación se presentan las particularidades de las muestras donde se encontraba un badén.

En la siguiente figura se muestra un intervalo de lectura de seis segundos sobre el eje Z del dispositivo cuando se pasó por un badén angosto. Debido a que se obtuvieron cinco muestras por segundo, un intervalo de seis segundos corresponde a treinta lecturas. La primera gráfica, correspondiente a la figura 4.6 (a), representa el valor en metros/segundo^2 de cada una de las treinta lecturas, mientras que la segunda gráfica, la figura 4.6 (b), muestra la desviación estándar de las cinco lecturas tomadas cada segundo.



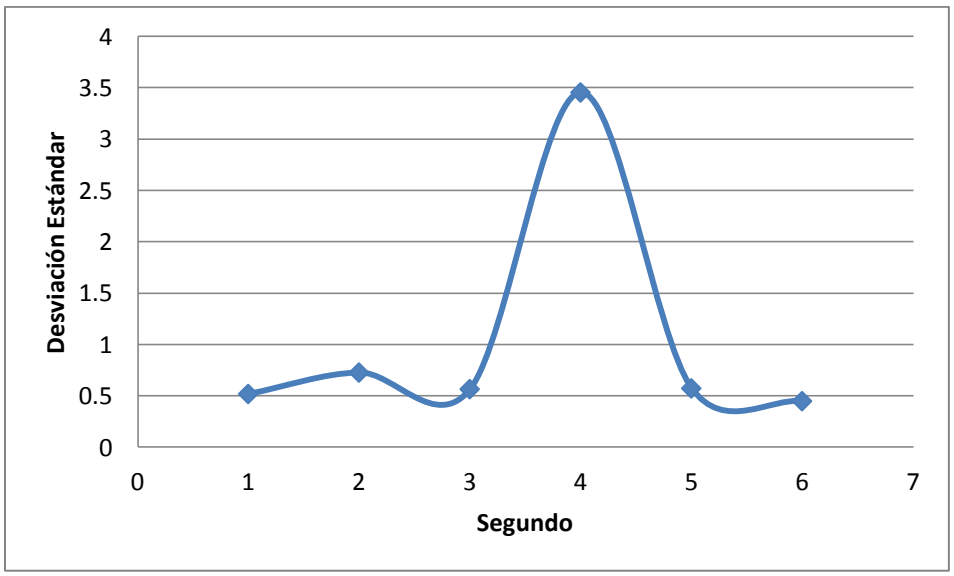
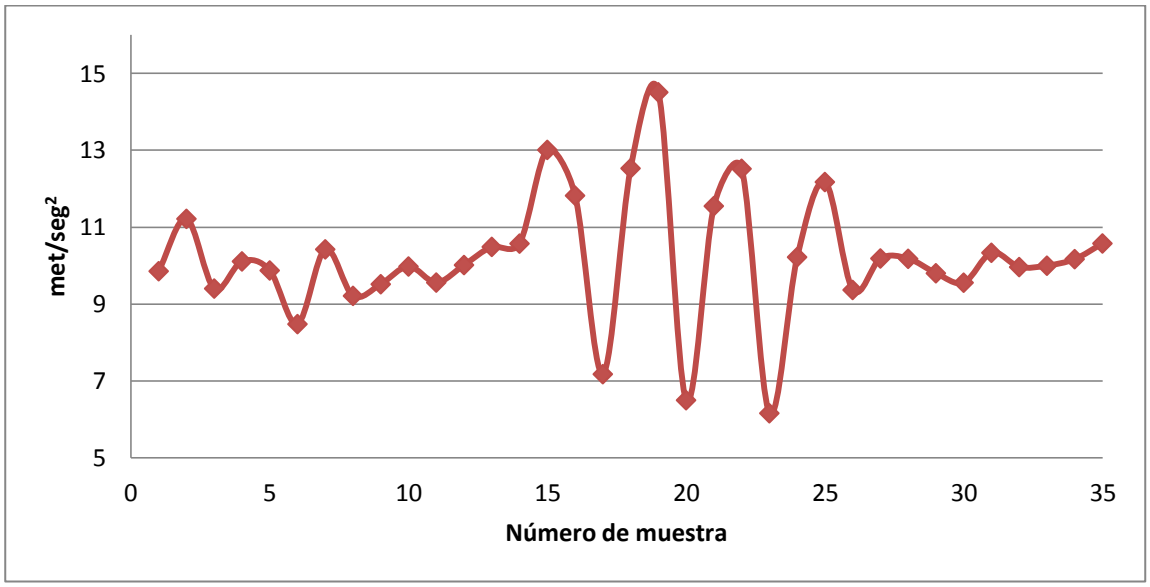


Figura 4.6 (a) Lecturas de acelerómetro badén angosto. (b) Desviación estándar por segundo

Se aprecia que entre las lecturas 15 y 20 de la figura 4.6 (a) existen fluctuaciones muy grandes con respecto al resto. Si se observa la gráfica de la figura 4.6 (b), se nota que la desviación estándar en el segundo 4 de la muestra tiene un valor cercano a 3.5, mientras que en el segundo anterior y posterior la desviación estándar ha rondado el valor de 0.57. Sin embargo, los conductores pueden

pasar por el mismo badén y obtener características distintas debido a su estilo de conducción como se puede notar en la figura 4.7 (a).



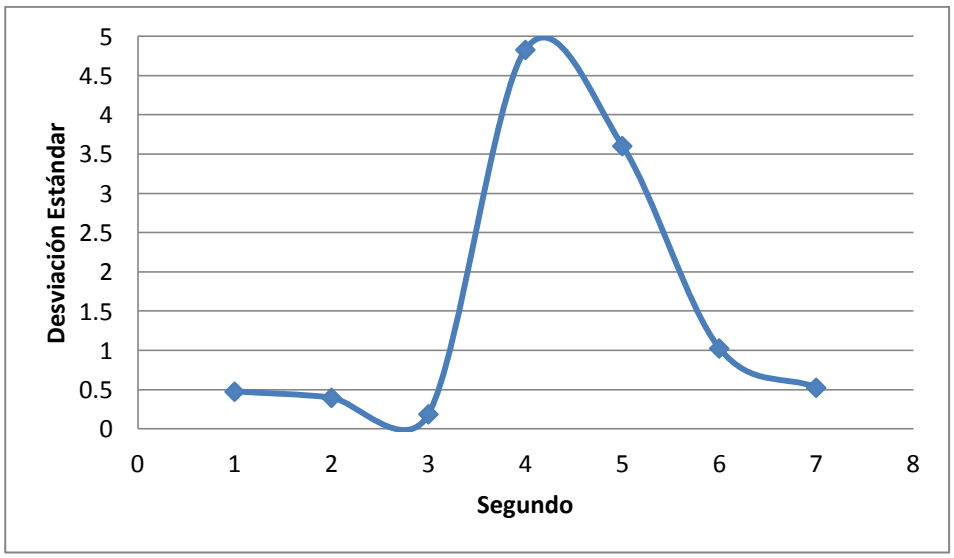


Figura 4.7 (a) Lecturas de acelerómetro badén angosto pasado más lentamente. (b) Desviación estándar por segundo

Tomando en cuenta que la tasa de muestreo aproximada es de cinco lecturas por segundo, ahora se puede observar que las fluctuaciones se dieron durante un periodo aproximado de dos segundos (muestras 15-25). Si se repite el procedimiento anterior y se calcula la desviación estándar de las lecturas de cada segundo se obtiene la gráfica 4.7 (b). En ella se observa que la desviación estándar es mucho mayor en los segundos 4 y 5 de la muestra, mientras que en los segundos 3 y 6 esta

desviación disminuye drásticamente. Se deduce por las fluctuaciones más prolongadas, que el conductor se tomó un segundo adicional en pasar este badén en comparación con el primer caso, es decir a una velocidad menor. Cabe destacar que el 27% de los badenes pasados en esta prueba tuvo características similares a las de esta lectura.

Para variar, ahora se muestran las lecturas correspondientes a la circulación sobre un badén ancho en una franja de tiempo de cuatro segundos.

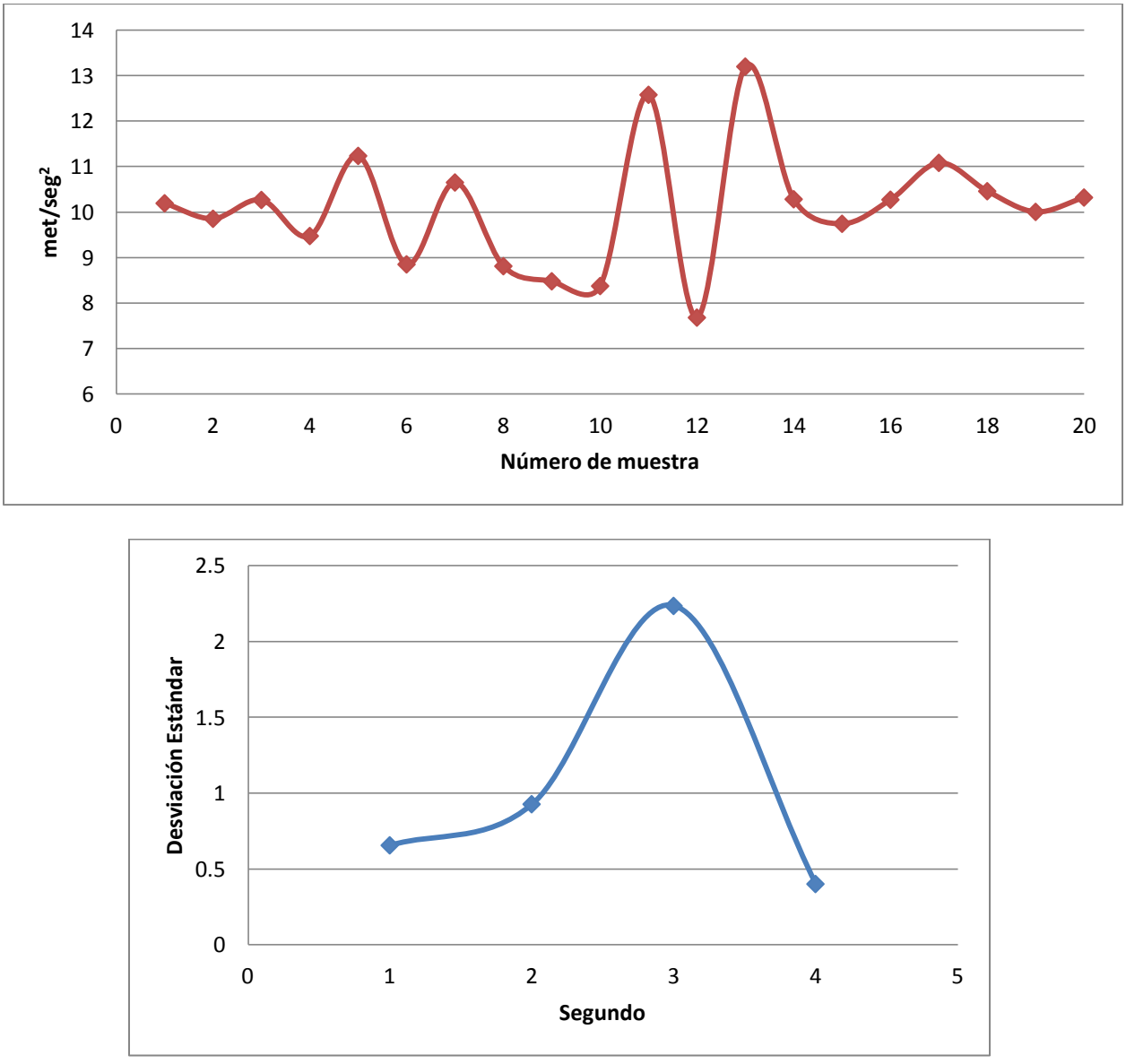


Figura 4.8 (a) Lecturas de acelerómetro badén ancho. (b) Desviación estándar por segundo

Se aprecia en la figura 4.8 (b) que la desviación estándar en el segundo 3 fue mayor a la de los segundos 2 y 4, un caso idéntico al del badén angosto del primer ejemplo en la figura 4.6.

De manera análoga a la circulación del badén angosto, se muestra a continuación una franja de cinco segundos, o 25 lecturas, donde el tiempo tomado para pasar sobre el badén ancho fue de dos segundos.

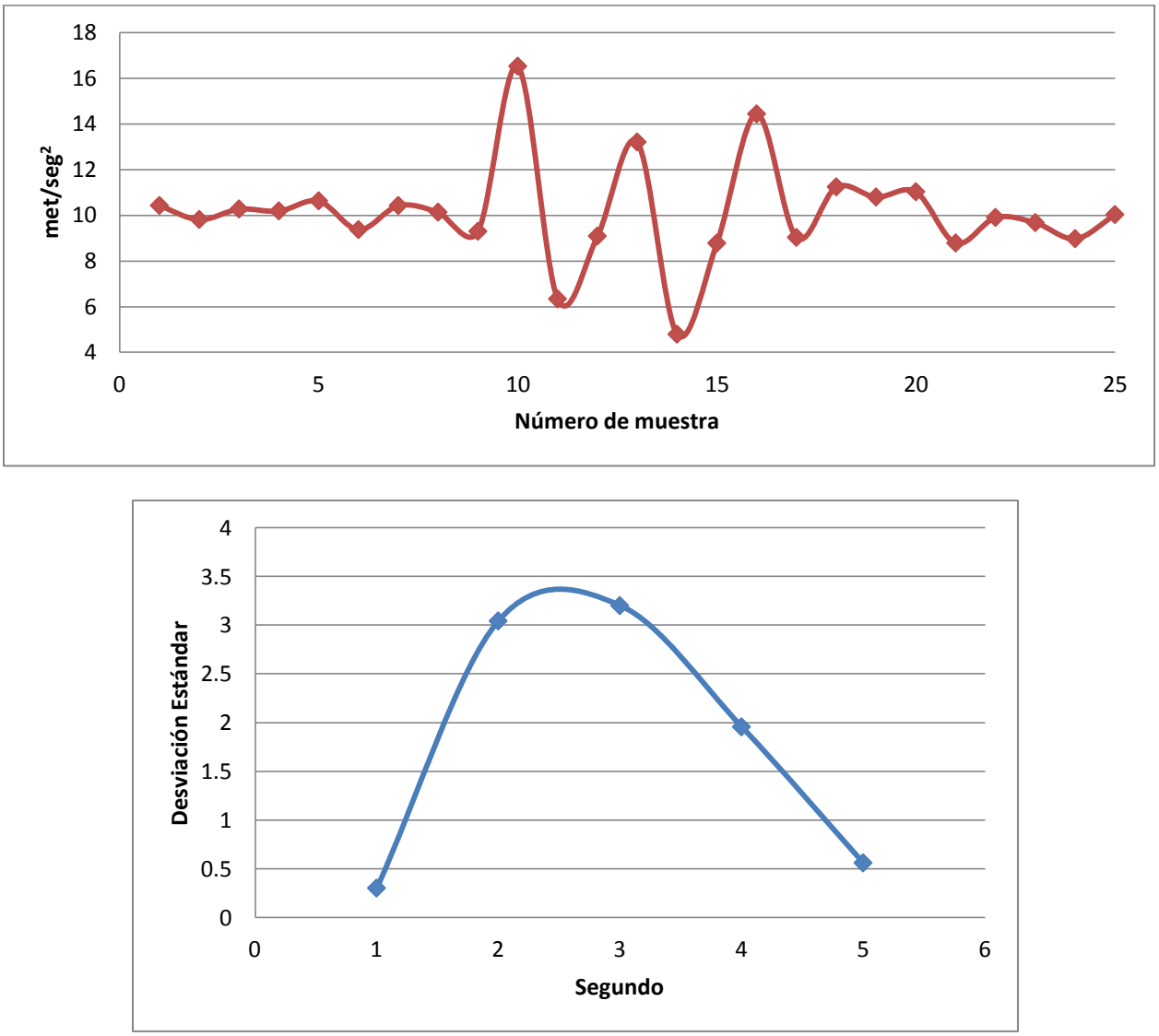


Figura 4.9 (a) Lecturas de acelerómetro badén ancho pasado más lentamente. (b) Desviación estándar por segundo

Justo como con el badén corto de la figura 4.7, este badén ancho presenta dos desviaciones estándar muy por encima del valor de las demás, en los segundos 2 y 3 de la figura 4.9 (b). La única diferencia que se puede apreciar con respecto al badén corto es que son valores ligeramente más

bajos, que podría ser explicado por el hecho de que los badenes anchos provocaron una sacudida menor a la de los badenes angostos.

Por último, la siguiente franja de tiempo resulta interesante, ya que se pasó por un badén ancho, pero en lugar de hacerlo de frente se circuló sobre él de forma diagonal, obteniendo la siguiente colección de lecturas y su correspondiente desviación estándar por segundo.

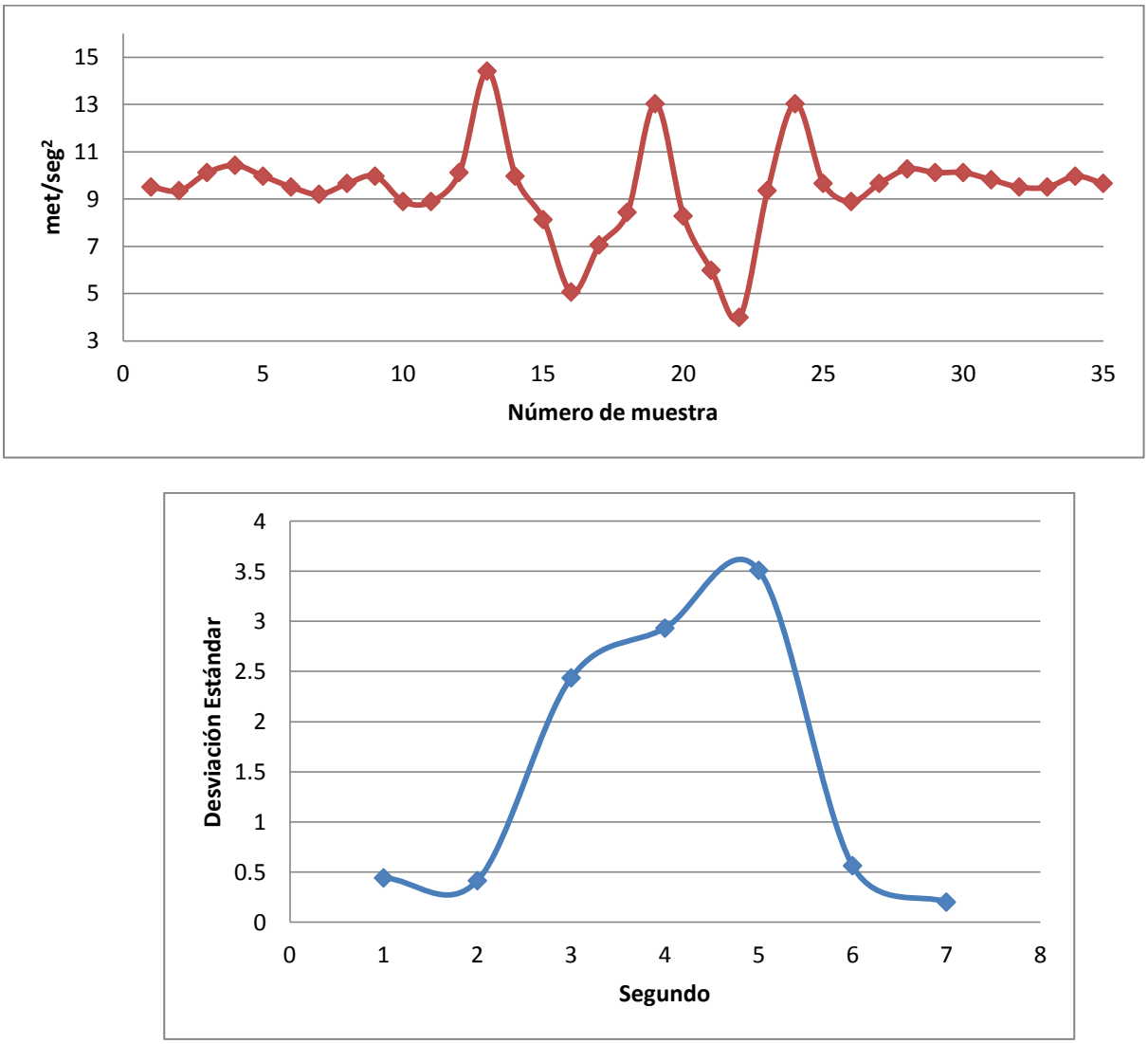


Figura 4.10 (a) Lecturas de acelerómetro badén ancho pasado en diagonal. (b) Desviación estándar por segundo

Se aprecia en la figura 4.10 (b), que en este caso la desviación estándar tuvo valores muy elevados en una ventana de tiempo de tres segundos, por lo que es otro caso a considerar. No

muchos badenes tuvieron estas características cuando se circuló sobre ellos, apenas un 3%, pero es algo a tener en cuenta.

Una vez presentados estos ejemplos se extrae que la lectura de los badenes contiene las siguientes características:

- La amplitud del vector de aceleración tiende a oscilar ampliamente cuando el automóvil pasa por el badén. Por lo tanto la desviación estándar de las lecturas obtenidas debe de ser mayor en la misma ventana de tiempo.

- La desviación estándar es considerablemente menor en los segundos previo y posterior a la circulación sobre el badén, tomando en cuenta que se circula por un terreno regular.

- Circular sobre un badén puede tomar entre uno y tres segundos, por lo que se deben evaluar periodos de hasta cinco segundos para determinar la existencia de un badén.

- Adicionalmente, para determinar que se circula por un badén, también debe de considerarse que el dispositivo tiene que cambiar sus coordenadas geográficas regularmente, de lo contrario podrían emularse los movimientos que caracterizan a un badén sin cambiar de posición geográfica.

Con todas estas características identificadas ha sido posible sintetizar un algoritmo capaz de detectar los badenes por los que se ha circulado en un coche. En la figura 4.11 se muestra de forma resumida y en forma de diagrama de flujo, el algoritmo mencionado.

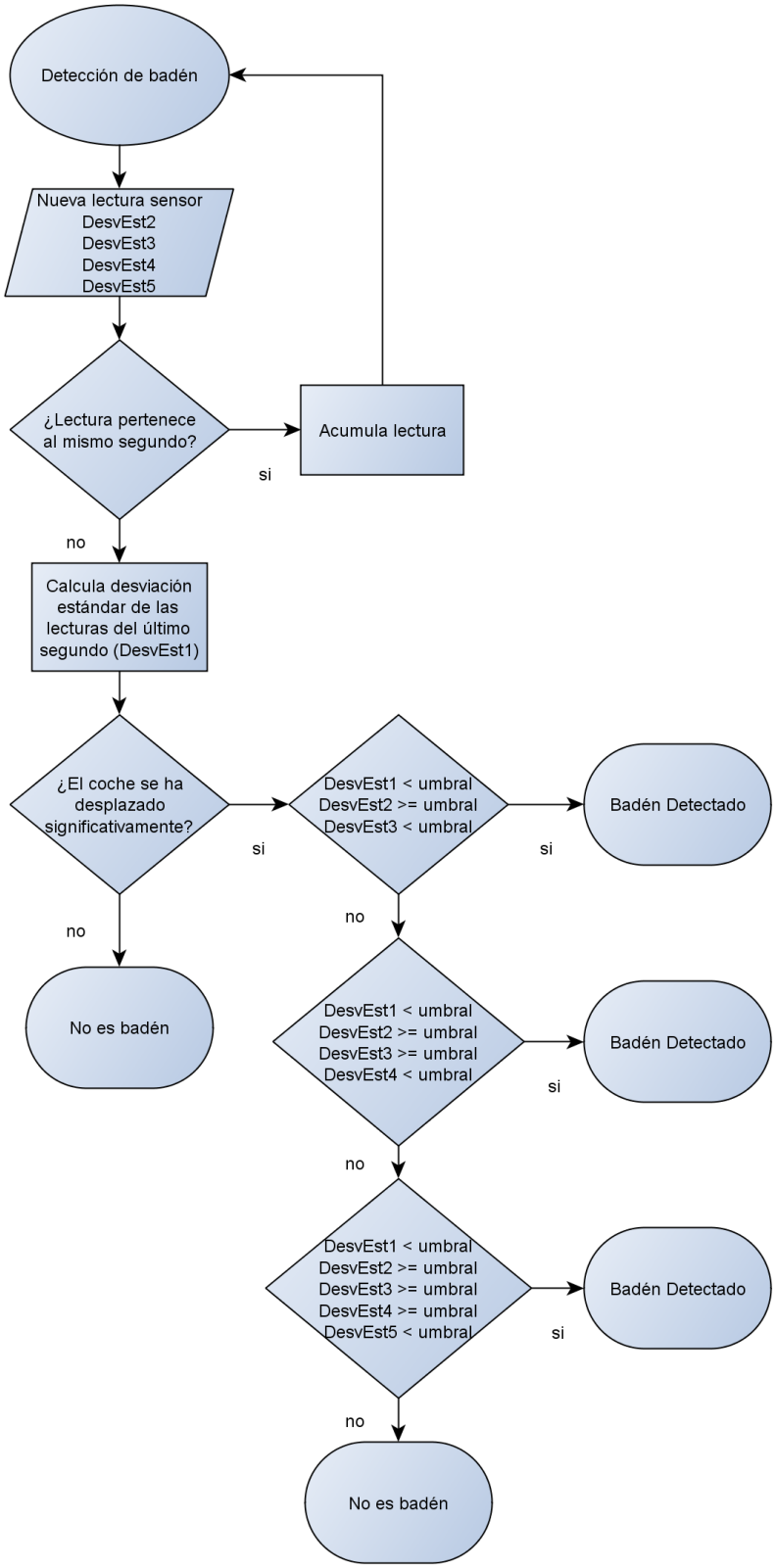


Figura 4.11 Algoritmo de detección de badenes.

El algoritmo es ejecutado cada vez que se obtienen nuevas lecturas del sensor. Una vez recibida la nueva lectura, el algoritmo evalúa si esta pertenece al mismo segundo que lecturas anteriores o si es la primera lectura del siguiente segundo. En el caso de que corresponda al mismo segundo se acumula la lectura en una variable. Cuando se tenga la lectura de un nuevo segundo entonces se calcula la desviación estándar del segundo anterior, y es guardada en la variable DesvEst1 que aparece en el diagrama de flujo.

Tal como se ha extraído de los experimentos de conducción, se deben evaluar periodos de hasta cinco segundos para determinar que se circula a través de un badén por lo que se guardan las últimas cuatro desviaciones estándar, representadas en el diagrama como DesvEst2, DesvEst3, DesvEst4 y DesvEst5 siendo 2 la más reciente y 5 la más antigua, y se añade la desviación del segundo más reciente DesvEst1 como parte de la evaluación.

Antes de comparar las desviaciones estándar de los últimos segundos, el algoritmo debe comprobar que el coche está en movimiento, algo fácilmente verificable obteniendo la localización mediante el método expuesto en la sección 4.1 de este capítulo.

Una vez que se ha verificado que el dispositivo está en movimiento, se procede a comparar las desviaciones estándar de acuerdo a los resultados extraídos de estas pruebas. Básicamente hay que comprobar que la lectura más reciente, así como la más antigua estén por debajo de un umbral de desviación estándar y que las lecturas intermedias se encuentren por arriba de dicho umbral. El umbral puede ser establecido por el usuario del dispositivo, aunque tiene por defecto un valor de 1.9, que ha sido identificado como un valor óptimo de reconocimiento de badenes en el terreno regular de la universidad sin caer en falsos positivos. Si las lecturas poseen las características mencionadas entonces se considera que se ha pasado por un badén.

Adicionalmente al algoritmo de detección, hay que recalcar que el sistema está diseñado para ser usado en modo cooperativo, por lo que será necesaria la confirmación por parte de varios dispositivos antes de que el servidor lo considere un badén. Esto ayudará a aumentar la precisión global del sistema además de que ayudará a reducir los falsos positivos.

5. Arquitectura del Sistema

En este capítulo se detallan los distintos elementos que componen el sistema, a nivel general, a nivel de la aplicación cliente y a nivel de aplicación en el servidor.

En el capítulo 3 ya se habló de los componentes que puede tener una aplicación Android, por lo que en este capítulo se describirá con detalle cuáles de estos componentes han sido utilizados, la organización de los mismos, así como las tecnologías de las que se han apoyado. Análogamente a la aplicación cliente, también se describirán las tecnologías utilizadas a nivel servidor.

También se detallará el flujo de datos que existe entre todos estos componentes, y dado el caso, el almacenamiento de información que ocurre de los mismos.

5.1. Arquitectura General

En la figura 5.1 se muestra la arquitectura, así como la interacción a nivel general de todos los elementos del sistema cuando se realiza una detección y captura de badenes, así como el componente de prevención y alarma. Modificar figura aclarando modo detección y modo prevención.

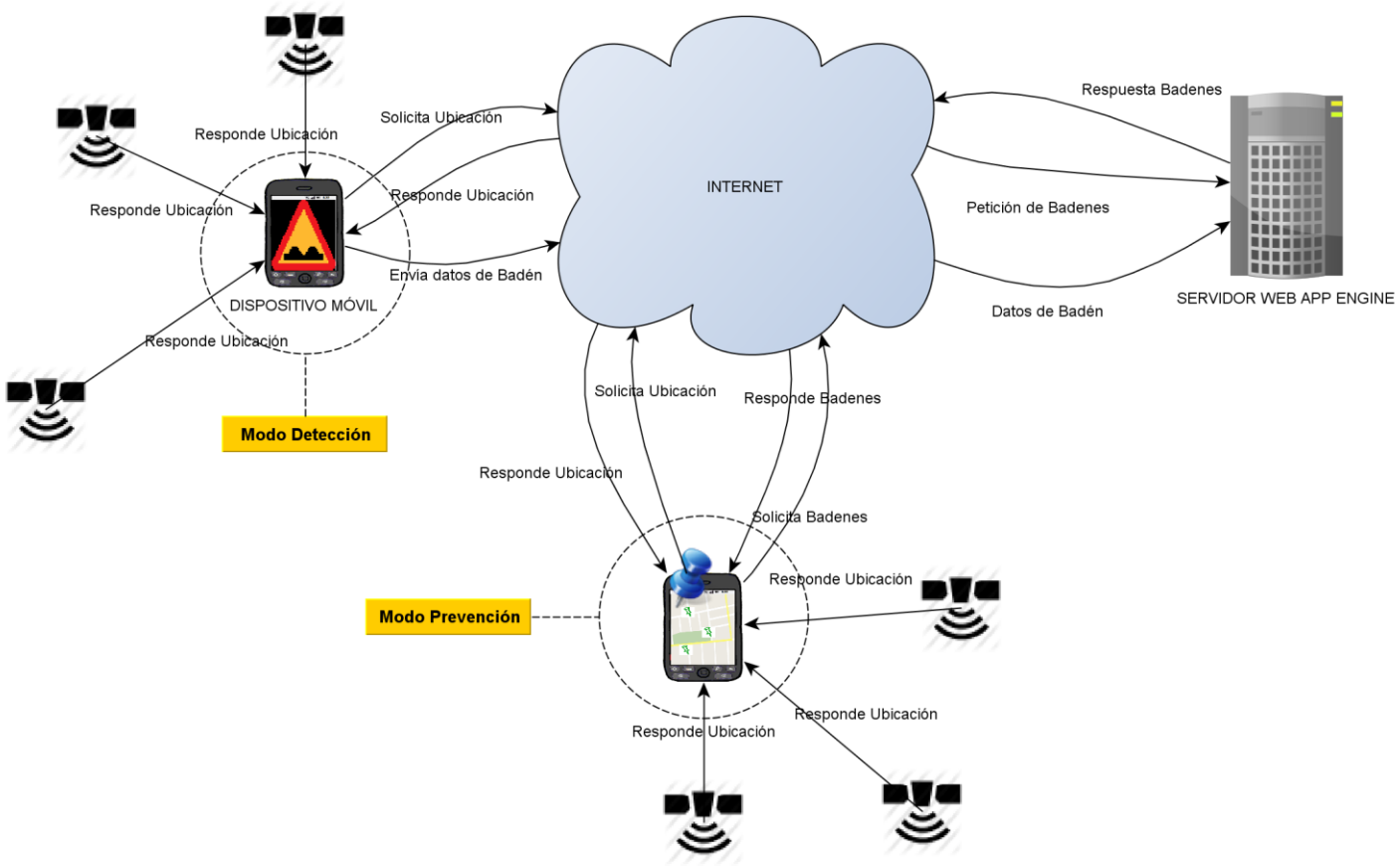


Figura 5.1 Arquitectura del sistema

Por un lado se tiene la aplicación cliente instalada en los dispositivos móviles. Uno de los modos principales es el de detección y captura de badenes, donde el cliente solicita por una parte su ubicación por medio de GPS y WiFi.

Una vez terminada la captura de datos, éstos pueden ser enviados al servidor cuando el usuario lo desee mediante una petición de tipo POST. Esta petición incluirá la latitud, longitud y dirección en la que el badén ha sido capturado.

El otro modo principal de la aplicación es el de prevención y alerta de badenes. Al igual que en el modo de detección, éste solicita la ubicación del dispositivo a través de GPS y WiFi. Cuando se tiene una ubicación, se le envía la misma al servidor mediante una petición de tipo GET. Este responde enviando todos los badenes en un radio aproximado de 2 km a la posición del dispositivo. Si el dispositivo se mueve 1.5 km desde la última posición en que se realizó la petición al servidor,

entonces vuelve a realizar la petición GET para actualizar la información de badenes que rodean al usuario en el radio antes especificado.

El servidor consta de una aplicación que corre en Google App Engine. Esta gestiona dos tipos de peticiones ya mencionadas, POST y GET. Cuando recibe la primera el servidor puede realizar dos acciones, crear el objeto e insertarlo en la base de datos si es que éste no existía previamente o actualizarlo incrementando el número de intentos o veces que se ha reportado un badén en la ubicación. Como la ubicación tiene un radio de error y no se puede ser tan específico con ella, no se considera un nuevo badén si se encuentra a un radio de distancia de 10 metros de otro reportado con anterioridad y solamente se actualiza el número de intentos.

En el caso de la petición GET, la aplicación se encarga de realizar la búsqueda en la base de datos de todos los badenes que rodeen al dispositivo en un radio de 2 km para que así la aplicación cliente los agregue visualmente y se pueda activar la alarma en caso de acercarse lo suficiente a ellos. La única condición para considerar los badenes que rodean al dispositivo es que éstos tengan un mínimo número de tres detecciones previas para ser considerados.

5.2. Arquitectura de la aplicación Cliente

Como se ha mencionado ya, las aplicaciones en Android constan de diversos componentes. Se podría decir que el componente primario de las aplicaciones son las *Activities*, que en su conjunto constituyen la interfaz gráfica de usuario y donde se aloja gran parte de la lógica de las mismas.

Las *Activities* suelen tener una jerarquía, de tal forma que para acceder a una hay que primero pasar por alguna otra, en la siguiente figura se muestra la jerarquía y flujo entre *Activities* de esta aplicación.

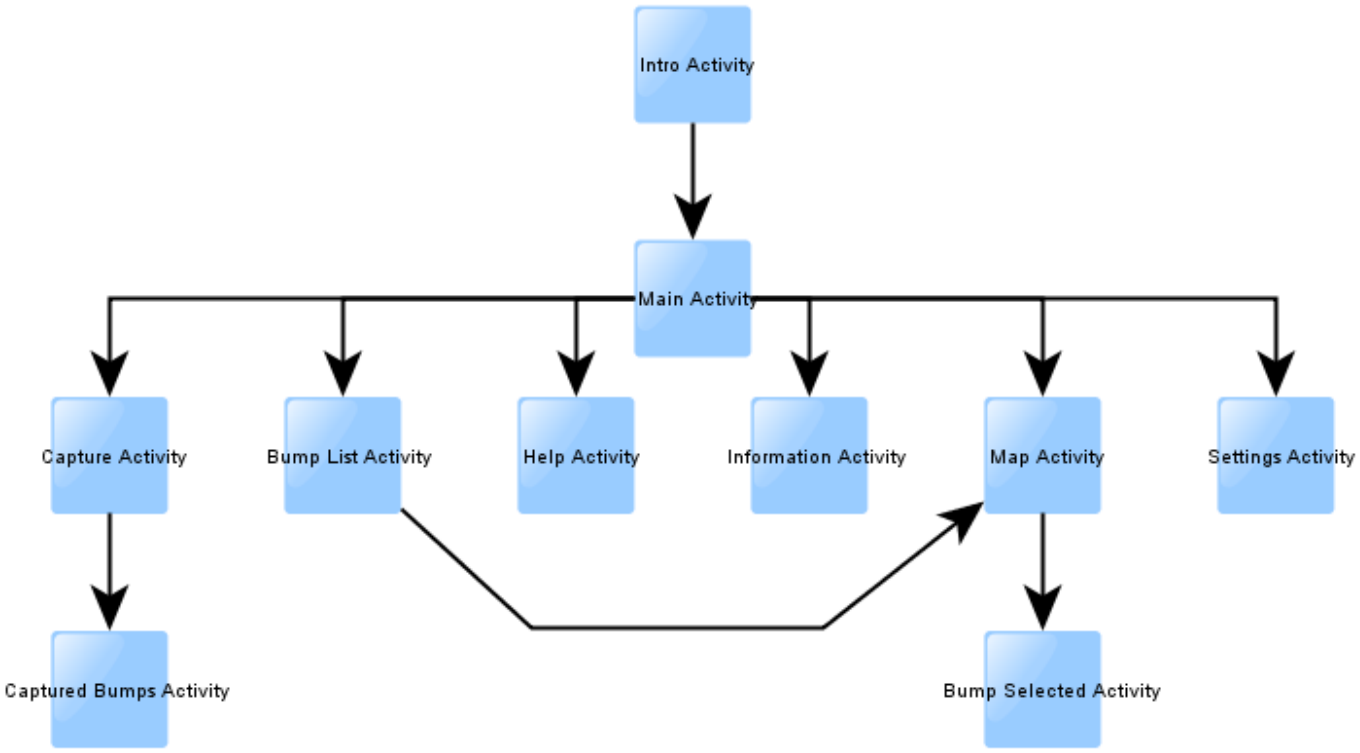


Figura 5.2 Arquitectura de aplicación cliente

La aplicación está compuesta de las siguientes *Activities*:

- **Intro Activity**: *Activity* con una imagen introductoria a la aplicación.

- **Main Activity**: En esta *Activity* se muestra la interfaz desde la cual se accede a las distintas funcionalidades de la aplicación.

- **Capture Activity**: Desde ésta se accede a la interfaz de captura de badenes. En esta *Activity* se lanza la ejecución de un *sensor listener* de tipo acelerómetro, mediante el cual se determina si hay un badén o no, así como de un *location listener* desde el cual se obtendrá la ubicación por medio de WiFi y GPS. También muestra de forma gráfica, las fuerzas de aceleración actuando sobre el dispositivo. Los badenes encontrados son guardados en un fichero temporal, que es reutilizado en la *Activity* de "Captured Bumps".

- **Captured Bumps Activity**: En esta *Activity* se muestran todos los badenes encontrados en la *Activity* anterior. El usuario es capaz de ver la gráfica generada por los vectores de

amplitud generados de las lecturas del acelerómetro del dispositivo. También tiene a su disposición la opción de enviar por correo todos los datos de la captura a su correo electrónico.

- **Map Activity**: Hay dos formas de acceder a esta *Activity* que a su vez representan dos tipos de acciones distintas. Si se accede desde el menú principal se entra al modo "conducción y prevención" de mapa en 3D en cual se puede apreciar la posición actual del dispositivo. Es un modo similar al de los sistemas de navegación tradicionales, donde se le alertará al usuario de los badenes cercanos. Para lograrlo se hace uso de un *location listener*, que obtendrá la ubicación del dispositivo por medio de WiFi y GPS. También se hace uso de un *sensor listener* para determinar la orientación del dispositivo más fácilmente. La alerta se logra mediante la implementación de un *Broadcast Provider*. Si se accede desde la *Activity* de "Bump List" se entra a un modo de especificación de localización del badén seleccionado. A través de este último modo se puede acceder a la *Activity* "Bump Selected Activity".

- **Bump List Activity**: Una interfaz desde la cual se puede ver el listado de todos los badenes capturados y guardados en una base de datos local. Se puede seleccionar cualquier badén de la lista para acceder al mapa en un modo que permitirá especificar, si así se desea, la ubicación precisa del badén. También existe la opción de borrar todos los badenes de la base de datos local.

- **Bump Selected Activity**: Es una interfaz mediante la cual se puede enviar la localización del badén al servidor así como borrar el badén de la base de datos.

- **Settings Activity:** Permite personalizar la aplicación al usuario. Mediante esta interfaz se especifica el correo electrónico del usuario para que se le envíe la información de captura, su localización de preferencia, sonidos de detección de badén y alerta del mismo, así como el radio de distancia para el aviso de badén.

- **Help Activity**: Mediante esta *Activity* se accede a un video alojado en los servidores de youtube, con un tutorial de funcionamiento de la aplicación.

- **Information Activity**: Pantalla con los créditos del desarrollador, universidad y herramientas externas utilizadas.

Como se ha mencionado, en el cliente existe una pequeña base de datos, hecha con el objetivo primordial de guardar los badenes encontrados de manera local antes de ser enviados al servidor. Si el dispositivo dispusiera de conexión permanente se podría prescindir de éste método, pero no es el caso. La ventaja de guardar la localización de los badenes antes de difundirlos al servidor es que si se conoce la posición exacta de los mismos, se puede modificar su localización antes de enviarla.

Android incorpora una pequeña base de datos llamada SQLite3. SQLite implementa la mayor parte del estándar SQL-92. La base de datos consiste en una tabla única llamada "Bumplocations", cuya definición se encuentra en la clase "**MyLocationsSQLiteHelper.java**". Asimismo, la clase contiene el nombre de todos los elementos que la componen. En la figura 5.3 se resumen los mismos.

Bumplocations
Key: Identificador del badén.
Latitude: Coordenada terrestre de latitud donde se encuentra el badén localizado aproximadamente.
Longitude:Coordenada terrestre de longitud donde se encuentra el badén localizado aproximadamente.
Address: Dirección aproximada del badén determinada por las coordenadas aproximadas de su localización.
Timestamp: Fecha en la que ha sido identificado el badén.

Figura 5.3 Propiedades de la tabla Bumplocations

Hay varias formas de realizar consultas, pero la utilizada en esta aplicación simplemente consiste en crear una instancia de "**MyLocationsSQLiteHelper**" y proceder a realizar un *query* simple como el que se muestra en la figura 5.4. Posteriormente se utilizan iteradores denominados Cursor mediante los cuales se pueden procesar las filas de la tabla de salida.

```
query( String   table,
       String[] columns,
       String   selection,
       String[] selectionArgs,
       String   groupBy,
       String   having,
       String   orderBy )
```

Figura 5.4 Sintaxis de instrucción Query en Android

Otro elemento muy importante es el *Broadcast Receiver*, mediante el cual se crean las alarmas en el modo de prevención de accidentes. Este es definido en la clase "**ProximityIntentReceiver**".

En cuanto se recibe una lista de badenes próximos a la localización del dispositivo, se crea una alerta de proximidad a la que hay que especificar latitud, longitud, el radio de alerta en metros y el tiempo de expiración. La clase "**ProximityIntentReceiver**" creará la notificación en el sistema, simplemente indicando el ícono, tiempo de activación y otras opciones como sonido y vibración.

5.3. Arquitectura de la aplicación Servidor

Una aplicación web que se ejecuta en Google App Engine tiene varios componentes. En el caso de este proyecto se ha optado por utilizar el modelo estándar de *Java Servlet* para interactuar con el entorno del servidor *web*. En esta sección se describirá la estructura y componentes más significativos que tiene la aplicación servidor.

La figura 5.5 muestra la estructura de la aplicación *web* como aparece en el entorno Eclipse.

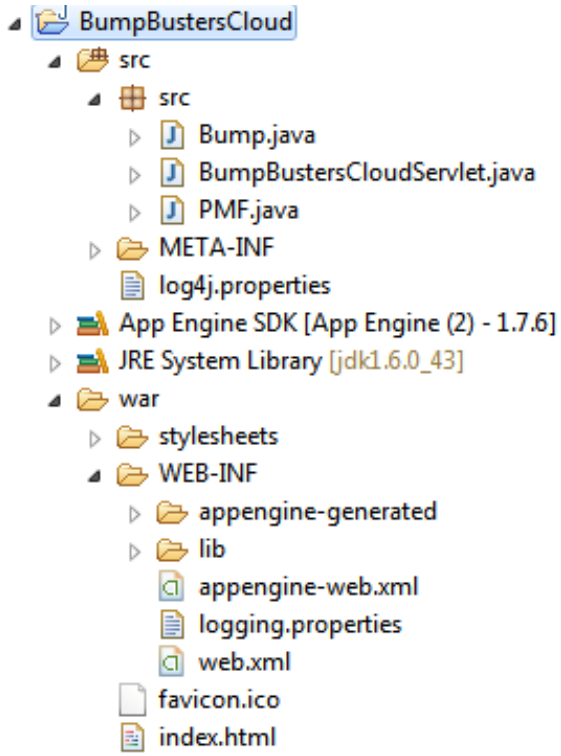


Figura 5.5 Elementos de aplicación servidor

Todos los ficheros de la aplicación, incluyendo las clases, JARs, ficheros estáticos y archivos de configuración son estructurados en el estándar WAR para aplicaciones *Java Web*. De esta manera se cuenta con un subdirectorio llamado **src/**, donde se aloja todo el código fuente Java y un subdirectorio llamado **war/** con la aplicación completa en el formato WAR. El proceso de *build* de Eclipse compila los ficheros fuente de Java y coloca las clases compiladas en el lugar apropiado de **war/**.

En el proyecto existen tres clases Java disponibles. La primera es "**Bump.java**". Esta clase corresponde con el tipo de objeto que habrá de almacenarse en la base de datos mediante JDO. Este tipo de objeto se define de forma igual a la de cualquier otro en Java. Consta de un constructor, modificador de atributos y métodos para obtener dichos parámetros. La diferencia es que habrán de añadirse algunas líneas en el código para darle capacidad de persistencia. Estas líneas se escriben justo antes de la clase y antes de cada atributo que deseemos hacer persistente. En la figura 5.6 se ejemplifica mejor.

```
@PersistenceCapable
public class Bump {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;
       @Persistent
       private float latitude; …
```

Figura 5.6 Otorgar capacidad de persistencia a la entidad Bump

Como se puede apreciar se declara a la clase como capaz de persistencia o *Persistence Capable*, y a cada uno de los atributos se les declara como persistente o *Persistent*. En la siguiente tabla se resumen los atributos y métodos que tendrá la entidad Bump mediante esta clase.

Métodos/ Operaciones	Bump: Crea un nuevo badén
	getKey: Obtiene el identificador único del badén
	getLatitude: Obtiene la coordenada de latitud del badén
	getLongitude: Obtiene la coordenada de longitud del badén
	getAddress: Obtiene la dirección del badén
	getDate: Obtiene la fecha en que fue modificado.
	getTries: Obtiene número de veces que se ha actualizado el objeto.
	setLatitude: Establece la coordenada de latitud.
	setLongitude: Establece la coordenada de longitud.
	setAddress: Establece la dirección del badén.
	setDate: Establece la fecha del badén.
	setTries: Establece el número de veces que se ha actualizado.

Atributos	Key	Identificador del badén.
	Latitude	Coordenada terrestre de latitud donde se encuentra el badén localizado aproximadamente.
	Longitude	Coordenada terrestre de longitud donde se encuentra el badén localizado aproximadamente.
	Address	Dirección aproximada del badén determinada por las coordenadas aproximadas de su localización.
	Date	Fecha en la que ha sido modificado el badén por última vez.
	Tries	Número de veces es que ha sido modificado el objeto Bump.

Otro elemento importante es la clase "**PMF.java**" o *Persistence Manager Factory*, la cual es necesaria para proveer a la aplicación de la capacidad de operar con objetos persistentes almacenados en el servidor.

Una aplicación interactúa con JDO utilizando una instancia de la clase PersistenceManager. Ésta se crea llamando a un método en una instancia de la clase **PMF**. El *Factory* de App Engine utiliza la configuración de JDO para crear instancias PersistenceManager.

Debido a que una instancia del *Persistence Manager Factory* necesita tiempo para inicializar, una aplicación debe reutilizar la misma instancia. Una forma sencilla de gestionar esta limitación es que la clase **PMF** contenga una instancia estática, y se hace como en la figura 5.7.

```
public final class PMF {

    private static final PersistenceManagerFactory pmfInstance =
            JDOHelper.getPersistenceManagerFactory("transactions-optional");

        private PMF() {}

        public static PersistenceManagerFactory get() {
            return pmfInstance;
        }

      }
```

Figura 5.7 Clase PMF

La última clase importante es el *Servlet* "**BumpBustersCloudServlet.java**". Como ya se mencionó, un *Servlet* es una aplicación en Java que procesa peticiones HTTP en un servidor, en el caso de éste proyecto únicamente de tipo POST y GET. Para ello se definen dos métodos en la clase.

```
public void doGet(HttpServletRequest req, HttpServletResponse resp){…

public void doPost (HttpServletRequest req, HttpServletResponse resp){…
```

Figura 5.8 Métodos GET y POST del *servlet* en la aplicación servidor

Como se aprecia en el pequeño fragmento de código de la figura 5.8, los dos son definidos de forma muy similar, variando solamente las acciones de un método respecto al otro.

Como ya se ha mencionado, el método *doGet* es el encargado de devolver la lista de badenes que rodean al dispositivo en un radio de 2 km. Para ello recibe como parámetros la latitud y longitud del dispositivo. Después consulta al almacén de datos con ambos parámetros para posteriormente dar una respuesta con la lista de badenes en formato *HTTP response* que el cliente plasmará de forma gráfica en la pantalla del dispositivo.

El método *doPost* por otra parte se encarga de insertar un nuevo elemento en el almacén de datos o de hacerle una actualización. Para ello primero recibe como parámetros la latitud, longitud, dirección y hora en que fue detectado. Después consulta al almacén de datos con los parámetros de latitud y longitud recibidos, si no encuentra ningún badén en la base de datos lo suficientemente cercano a la ubicación, entonces inserta el nuevo elemento con todos los parámetros enviados por el cliente. En el caso de que si encuentre una entrada previa lo suficientemente cercana a la ubicación, entonces actualiza el número de ocasiones que el badén ha sido reportado por el usuario.

Es muy importante el fichero web.xml, el cual dependiendo de la petición determina cual *Servlet* utilizar. En el caso de este proyecto solamente se hace uso de un *Servlet*, pero es necesaria la existencia de este fichero de cualquier forma para poder acceder a él. Igualmente importante es el fichero appengine-web.xml, el cual debe especificar el ID registrado de la aplicación, la versión de la misma, así como todos los ficheros estáticos que se utilicen.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
    <servlet>
        <servlet-name>BumpBustersCloud</servlet-name>
        <servlet-class>src.BumpBustersCloudServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BumpBustersCloud</servlet-name>
        <url-pattern>/bumpbusterscloud</url-pattern>
    </servlet-mapping>
```

Figura 5.9 Fichero web.xml

6. Implementación del Sistema

En este capítulo, se mostrará el funcionamiento de las aplicaciones desarrolladas tanto para el cliente, como para el servidor. Para ello se mostrarán capturas detalladas de la aplicación en funcionamiento.

6.1. Implementación en el Cliente

Como ya se ha mencionado, la aplicación cliente cumple tres funciones principales. Éstas son:

- Capturar los datos de los badenes encontrados en el camino.
- Enviar la ubicación de los badenes al servidor.
- Alertar a los usuarios de los badenes en el camino cuando este se encuentre conduciendo.

A continuación se explica cada uno de los apartados a detalle.

Captura de Badenes

Esta es la interfaz principal de la aplicación, el ícono del coche representa al modo captura. Si se accede a este modo se muestra la interfaz de la figura 6.2.

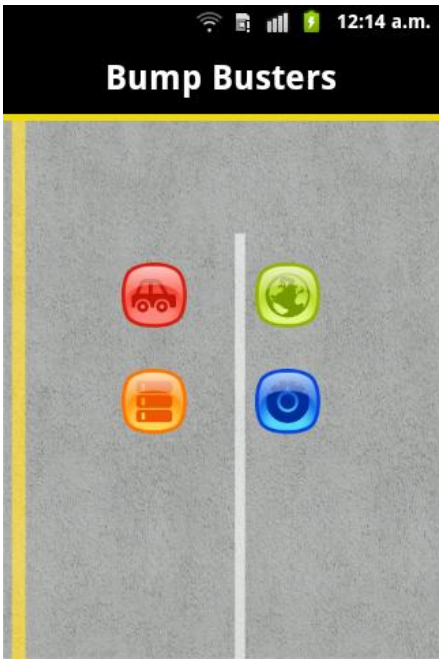


Figura 6.1 Interfaz principal aplicación cliente

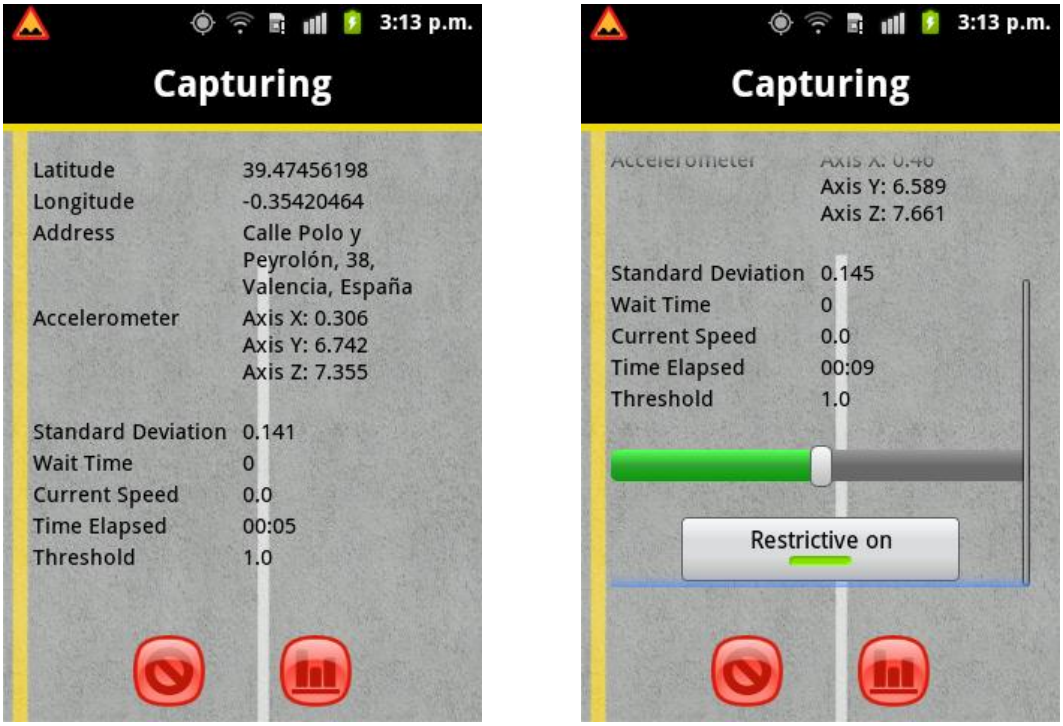


Figura 6.2 Modo de captura

En ella se muestra la latitud y longitud, ambos valores obtenidos mediante geolocalización GPS y WiFi. Después se muestra la dirección, obtenida mediante el proceso de *GeoCoding*, que se trata de obtener una dirección con base en las coordenadas. Naturalmente, este valor solamente podrá ser obtenido si existe una conexión activa a internet.

Después se muestran las lecturas del acelerómetro. El tiempo de muestreo está establecido como tipo *Normal*, lo cual equivale a aproximadamente cinco muestras por segundo, siendo suficientes para determinar si hay un badén o no. Existe la posibilidad de ver las lecturas de forma más visual presionando el ícono de gráfica como se muestra en la figura 6.3.

La siguiente lectura es la desviación estándar correspondiente a las muestras de cada segundo. Se puede apreciar como ésta es mayor cuando el dispositivo móvil es sacudido al pasar por un badén.

El siguiente campo corresponde al tiempo de espera antes de volver a considerar otra sacudida como un badén. Esto se ha implementado ya que varios de los patrones de detección mostrados en el capítulo 4 pueden repetirse dentro de una franja de tiempo en la que se pase sobre un badén. De esta forma se evita que la aplicación capture por duplicado un mismo badén.

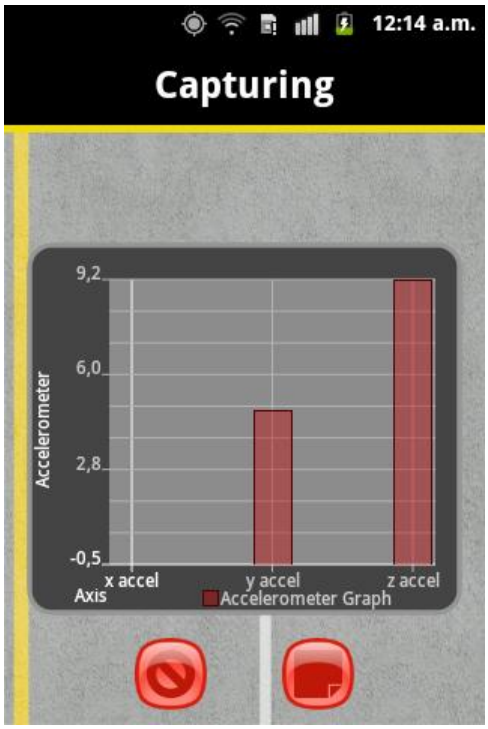


Figura 6.3 Representación gráfica de lecturas de acelerómetro

La velocidad se refiere a la del automóvil, determinada por la ubicación del GPS. Si no se dispone del mismo, la velocidad siempre será de cero. No es un dato muy relevante, ni muy exacto al momento de la realización de este trabajo, pero podría serlo en expansiones de este proyecto si es que se decidiera utilizar la velocidad como un dato determinante de detección de un badén.

Luego viene el tiempo transcurrido, que no es más que un cronómetro inicializado en cuanto se activa el modo de captura. No tiene más que un propósito exclusivamente informativo para el usuario de la aplicación, si es que éste desea conocer por cuánto tiempo ha estado capturando datos de conducción.

El siguiente campo corresponde al umbral de sensibilidad de detección, lo cual no es más que la desviación estándar mínima requerida para considerar un badén como se explicó en el capítulo 4. Ésta tiene un valor por defecto de 1.9 para la circulación en coche en el terreno regular de la Universidad, que fue el valor mínimo de detección que se distinguió al pasar sobre un badén durante la etapa de análisis de lecturas y sin obtener falsos positivos. Mientras mayor sea el umbral, el detector considerará sacudidas más fuertes como badenes y por tanto será más restrictivo. Este campo puede ser modificado en tiempo real con una barra táctil verde justo debajo.

Cuando se haya terminado de capturar, puede pararse la captura con el botón rojo asignado para este propósito, ubicado en la parte inferior izquierda de la figura.

Una vez que se ha parado la captura, se muestra la pantalla de resultados. La cual muestra la hora a la que comenzó la captura, la lista de badenes encontrados junto con su posición, dirección si es que se pudo determinar, el día y la hora exacta a la que fue detectado.

Figura 6.4 Interfaz de resultados

Si se presiona el botón con la flecha roja apuntando hacia la derecha, ubicado en la parte inferior derecha, se puede ver la gráfica generada a partir de la desviación estándar de las lecturas de amplitud tomadas por el acelerómetro. Se llega apreciar en la figura 6.5, un ejemplo de las fluctuaciones en la gráfica en los segundos 16-18 y 23-25.

Es posible enviar los datos y resultados de este análisis por correo electrónico para su posterior evaluación si así se desea. Esto se logra presionando el botón rojo de enviar, ubicado en la parte inferior de la interfaz. Lo que se envía son todas las lecturas recogidas por el acelerómetro, el tiempo preciso de lectura, las coordenadas geográficas, dirección si es que esta última se pudo determinar y la indicación de que se ha encontrado un badén por medio de un 1 o de lo contrario un 0. Ya que esta puede llegar a ser una cantidad enorme de información se puede enviar la misma a través de múltiples correos. El envío de datos se puede apreciar en la figura 6.6.

Cabe resaltar que el destinatario puede introducirse previamente por medio de la interfaz de opciones, accesible desde la interfaz principal presionando el ícono azul.

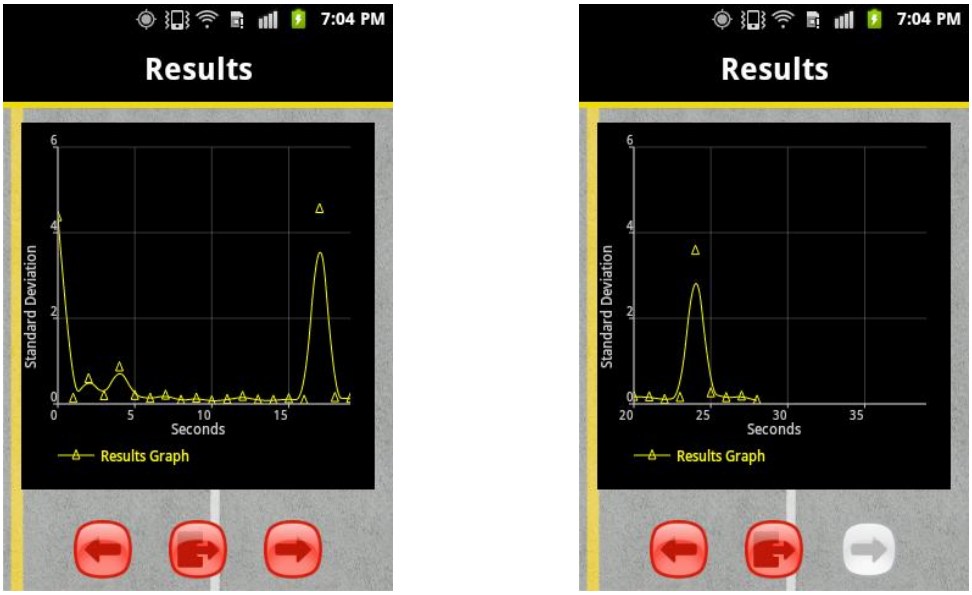


Figura 6.5 Gráfica generada por lecturas del acelerómetro

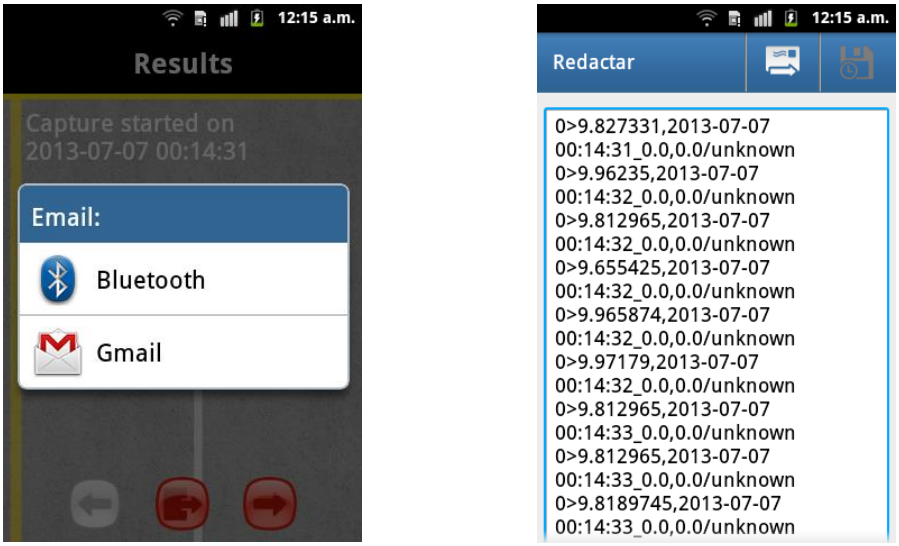


Figura 6.6 Envío de captura por correo electrónico

Envío de datos de badenes al servidor

Ya que no se dispone de conexión a internet ininterrumpida en el dispositivo Galaxy ACE, el envío de la información capturada tiene que realizarse de forma asíncrona. Esto se logra por medio de la interfaz mostrada en la figura 6.7, a la cual se accede presionando el ícono naranja de la pantalla principal.

Lo primero que muestra la *Activity* es una lista de todos los badenes guardados en la base de datos local.

Figura 6.7 Interfaz de listado de badenes en aplicación cliente

Si se selecciona un badén, éste será ubicado en la interfaz de mapa como se muestra en la figura 6.8. El motivo por el cual se accede a este mapa es que probable que al momento de la captura no se disponga de una conexión GPS, sobre todo al inicializar la *Activity*, por lo que la posición del badén puede no ser la correcta. Si se conoce la ubicación del badén, puede arrastrarse hasta la ubicación que se desee.

Una vez que se ha arrastrado, el badén guarda su nueva localización en la base de datos. Ya sea si se decide dejar en la ubicación con la que fue capturado o se prefiere cambiar de la misma, se puede seleccionar el badén presionando sobre su ubicación en el mapa y de esta forma enviarlo al servidor o borrarlo definitivamente de la base de datos local. Un ejemplo de esto se aprecia en la figura 6.9.

Figura 6.8 Interfaz de mapa para badenes almacenados localmente

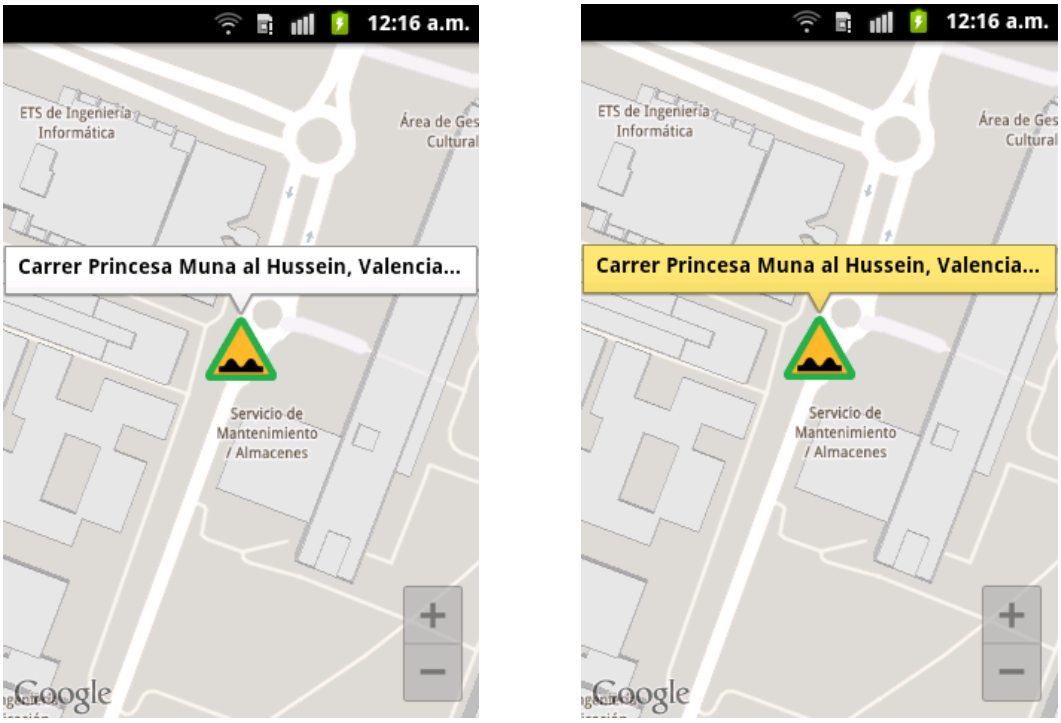


Figura 6.9 Arrastrado y selección de badén

Una vez seleccionado el badén mediante la interfaz de mapa, se accede a la pantalla de envío y/o borrado de badén. Como se aprecia en la figura 6.10, el botón rojo en la parte inferior izquierda es para enviar la información en pantalla al servidor, mientras que la otra opción disponible borra

directamente el badén de la base de datos. Cabe aclarar que si se envía la información del badén, también será eliminada localmente.

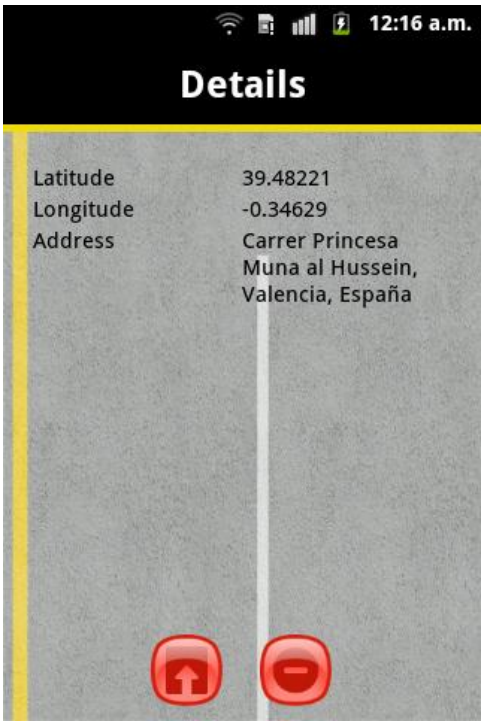


Figura 6.10 Interfaz de envío y borrado de badén local

Alerta de badenes

La aplicación también alerta de los badenes encontrados en el camino. Para acceder a este modo simplemente hay que seleccionarlo desde el menú principal presionando el ícono de mapa. Si se dispone de conexión a internet se mostrará el mapa de forma tridimensional con la ubicación actual del dispositivo.

Como se puede ver en la figura 6.11, la ubicación también señala la dirección hacia la cual se encuentra apuntando el dispositivo.

Figura 6.11 Interfaz de prevención de badenes

En cuanto la interfaz de mapa es inicializada, esta hace la petición de los badenes que rodean al usuario en un radio aproximado de dos kilómetros. Cuando la ubicación del usuario se acerque a la de algún badén, entonces se activará una alarma de aproximación al mismo. La alarma consta de un sonido especificado por el usuario, así como de un mensaje mostrado en la figura 6.12.

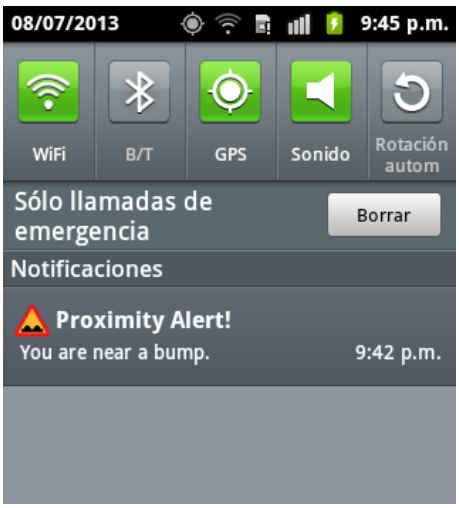


Figura 6.12 Alerta de proximidad al badén

6.2. Implementación en el Servidor

La implementación en el servidor tiene como objetivo primordial guardar la localización de los badenes, así como de enviar su información si es que se cumplen ciertas condiciones.

La aplicación servidor no cuenta con una interfaz gráfica a disposición del cliente, sin embargo GAE permite visualizar las peticiones HTTP que la aplicación ha resuelto, así como visualizar el contenido del *datastore* por medio de la consola de administración.

Cuando la aplicación cliente envía la información de un badén al servidor, hace una petición de tipo POST de acuerdo al estándar HTTP como se muestra en la figura 6.13. Si existiera algún error crítico en la solicitud, la aplicación lo mostraría con ícono rojo.

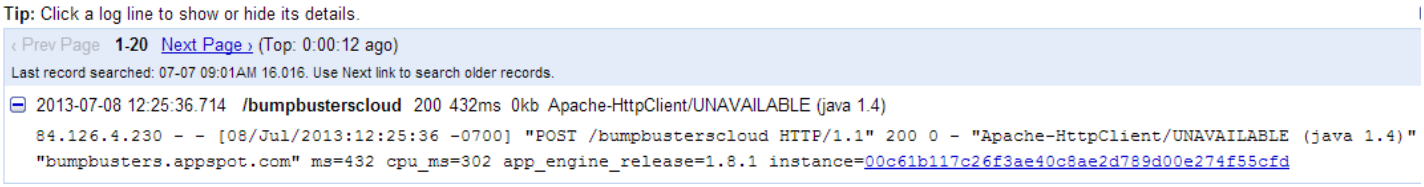


Figura 6.13 Petición POST recibida por aplicación servidor

Cuando las peticiones POST son resueltas exitosamente, los badenes son guardados en el *datastore*. En la figura 6.14 se muestra el contenido almacenado en el *datastore* de App Engine. Éstas son todas las entidades de tipo "Bump" de las que se han recibido datos.

Figura 6.14 Contenido del *datastore*

El servidor también permite ejecutar sentencias GQL sobre las entidades almacenadas. Esto último es una funcionalidad muy útil para probar *queries* antes de implementarlos en el *servlet*. En el ejemplo de la figura 6.15 se seleccionaron todos los badenes que se encontraban en cierto rango de latitud.

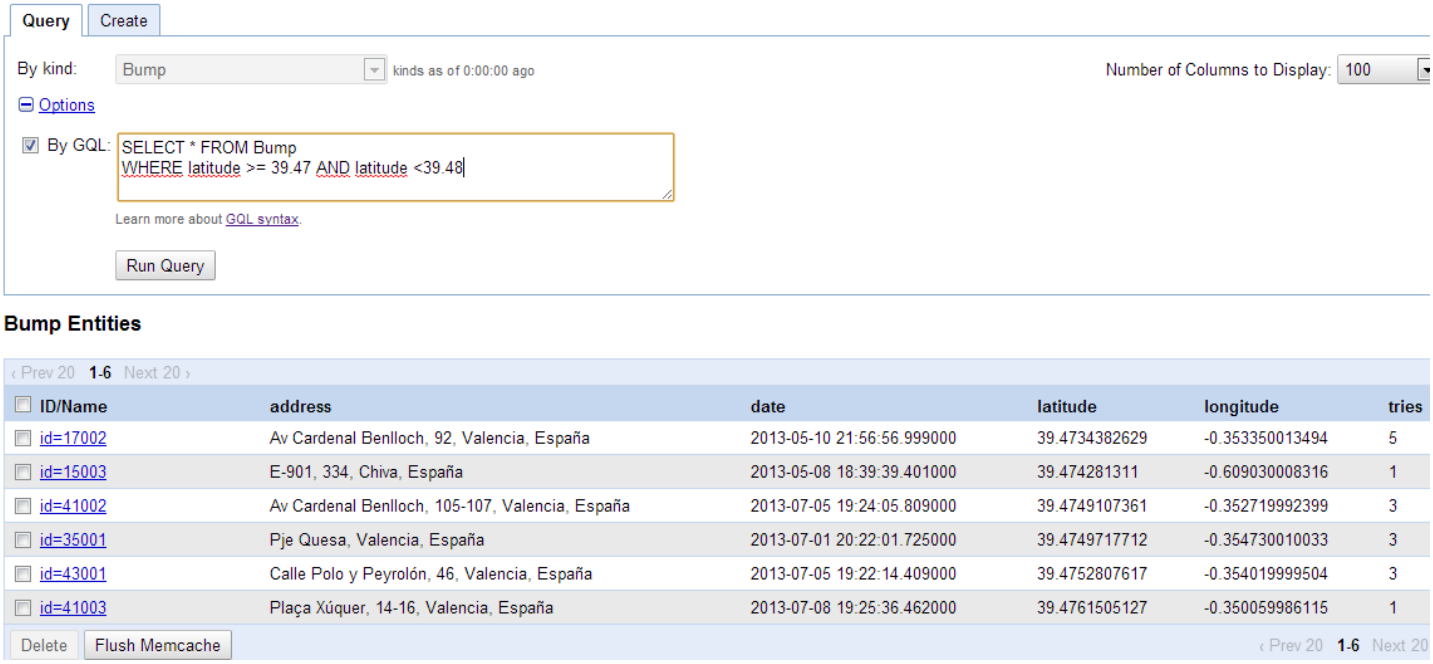


Figura 6.15 Ejemplo de sentencia GQL ejecutada

Como con las peticiones POST, se pueden observar las peticiones de tipo GET cuando la aplicación cliente las ejecuta. Si existiera algún problema con la petición, la consola de administración lo mostraría.

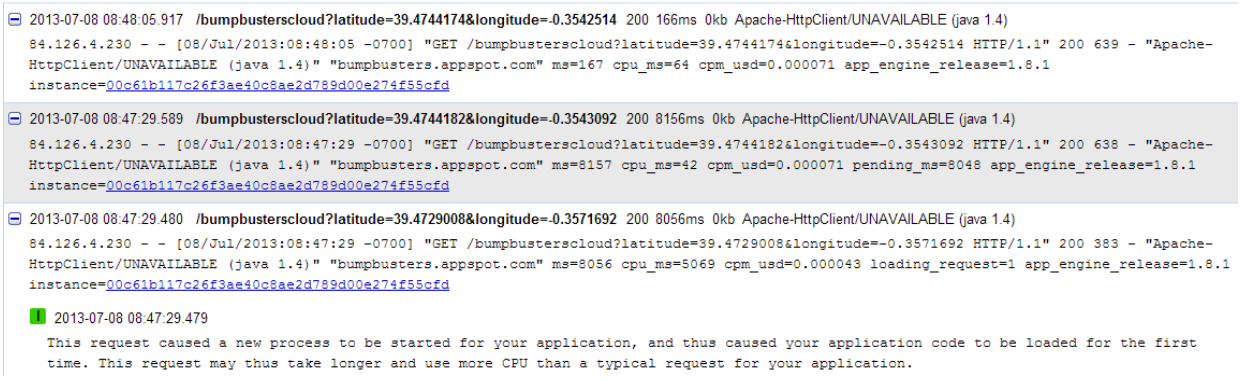


Figura 6.16 Petición GET recibida por aplicación servidor

7. Pruebas

En esta sección se detallan las pruebas realizadas en diferentes escenarios y se presentan los resultados obtenidos. La aplicación está hecha para funcionar primordialmente en automóviles, pero también se han realizado pruebas de detección en autobuses con el fin de determinar si existen diferencias o mayores posibilidades de detección de falsos positivos en un vehículo con diferentes características.

7.1. Recorrido en coche por la Universidad Politécnica de Valencia

Una vez analizados los requerimientos del sistema y después de implementar las aplicaciones cliente y servidor se procedió a realizar pruebas de detección. De nueva cuenta, se ha seleccionado el campus de la Universidad Politécnica de Valencia como terreno de pruebas debido a la alta cantidad de badenes que se pueden encontrar dentro. El recorrido, realizado múltiples ocasiones de ida y vuelta, es mostrado en la figura 7.1.

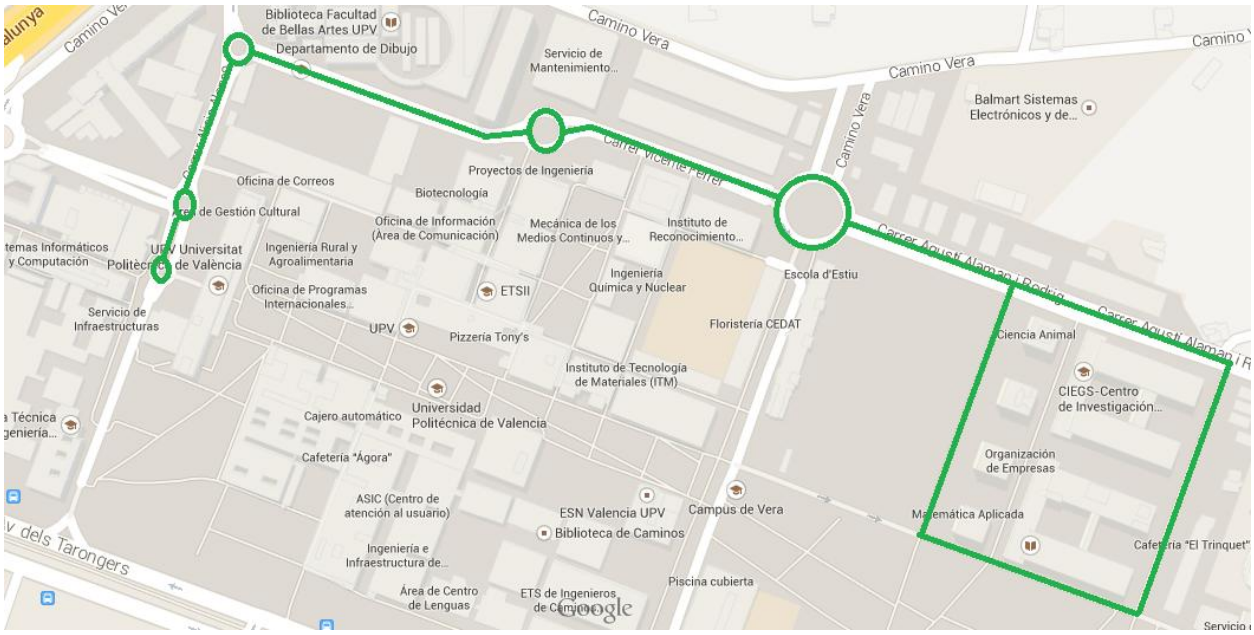


Figura 7.1 Recorrido realizado repetidas veces por el campus de la UPV

El dispositivo móvil ha vuelto a ser colocado en el automóvil de la misma forma en que se realizó en las lecturas del capítulo 4 (figura 4.4). También en la figura 7.2 se puede observar la forma en que ha sido desplegado dentro del automóvil, asegurándose que la interfaz de la aplicación sea accesible en caso de querer realizar ajustes.

Figura 7.2 Despliegue del dispositivo móvil dentro del automóvil

El objetivo primario de estas pruebas es detectar la mayor cantidad de badenes posible. Como ya se había mencionado en el capítulo 4, se pueden encontrar dos tipos de badenes en el recorrido a través del campus universitario, ancho y angosto (figuras 7.3 y 7.4), por lo que también se busca

determinar si es más fácil detectar un tipo de badén sobre otro. También se quiere encontrar cuál es el umbral óptimo de detección, ya que aunque se desea detectar la mayor cantidad de badenes posible, es importante no tener falsos positivos.

Figura 7.3 Ejemplo de badén angosto en la UPV

Figura 7.4 Ejemplo de badén ancho en la UPV

Todo el recorrido se realizó con un umbral de desviación estándar de 1.9, valor que había sido identificado como mínimo en las pruebas de análisis de lecturas. Si se desea, se puede visualizar la detección en funcionamiento a través del siguiente enlace. Los resultados obtenidos son los siguientes:

Umbral Desv. Est.	Badenes pasados	Badenes detectados	% Detectados	Falsos Positivos
1.9	44	38	86.36	0

En cuanto a la distribución de badenes, se tiene que el 50% de los badenes era de tipo angosto y el 50% era de tipo ancho.

En cuanto al porcentaje de detección por tipo de badén se obtuvo un 81.81% para los badenes anchos y de un 90.90% para los badenes angostos. Esto se debe a que las desviaciones alcanzadas en badenes anchos tienen un valor menor ya que la sacudida es menos intensa y la circulación sobre ellos era más suave.

7.2. Pruebas adicionales sobre los datos recogidos

Los resultados de la prueba anterior han sido bastante buenos, pero ya que se ha recogido toda la información del recorrido, se pueden realizar pruebas adicionales sobre los mismos datos y ver si se hubieran podido obtener mejores resultados de usar un umbral distinto. Si se ejecuta el algoritmo con un umbral menor, por ejemplo de 1.7, entonces se obtienen los siguientes datos.

Umbral Desv. Est.	Badenes pasados	Badenes detectados	% Detectados	Falsos Positivos
1.7	44	41	93.18	1

Se observa que con este umbral, el porcentaje de badenes detectados se incrementa a más de 90%, aunque se hubiera obtenido un falso positivo, posiblemente por alguna irregularidad en el terreno que ocasionara una ligera sacudida. Es evidente que un umbral de 1.7 hubiera sido un poco más apropiado para el recorrido de la universidad de no ser por el falso positivo obtenido.

En cuanto al porcentaje de detección por tipo de badén, esta vez fue del 90.90% para badenes anchos y del 95.45 % para badenes cortos.

Ahora se intentará detectar todos los badenes bajando el umbral de detección a un valor de desviación de 1.5. Se obtienen los siguientes resultados.

Umbral Desv. Est.	Badenes pasados	Badenes detectados	% Detectados	Falsos Positivos
1.5	44	44	100	7

Como se puede observar, todos los badenes por los que se ha circulado han sido detectados, pero el número de falsos positivos se incrementó a 7, lo cual es un valor bastante alto. Cinco de los falsos positivos obtenidos ocurren en ubicaciones distintas, mientras que los dos restantes ocurren en la rotonda de la figura 7.5. En realidad es preferible no capturar todos los badenes y evitar los falsos positivos en la medida de lo posible, por tanto, este no parece un umbral de detección adecuado ya que incluso aparecen falsos positivos en lugares recurrentes.

Figura 7.5 Rotonda con falsos positivos

La probabilidad de detección de un badén está muy ligada a la velocidad y forma de conducir del usuario, por lo que si un badén no es detectado inicialmente por un usuario, es probable que otro usuario lo haga, así que sería mejor tener un umbral de detección más bien alto como se puede ver en la figura 7.6.

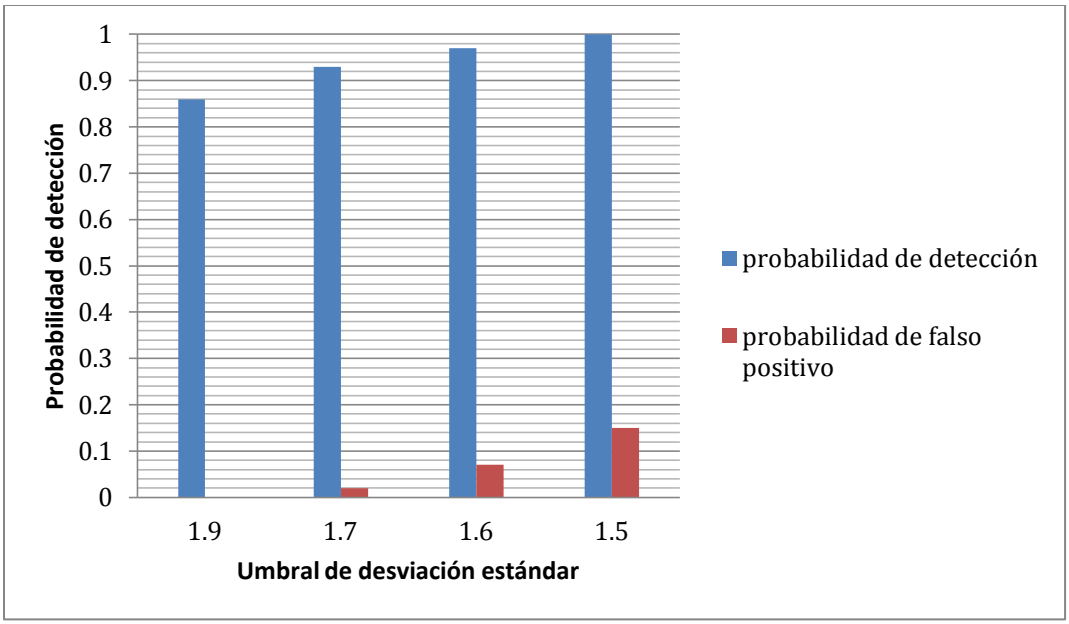


Figura 7.6 Probabilidad de detección de badenes y falsos positivos dependiendo de umbral de detección

7.3. Recorrido en autobús

El recorrido en autobús se realizó con el propósito de determinar si las características de lectura de un coche serían igual de válidas para el autobús. En este recorrido no se encontraron badenes, pero en su lugar se buscó que no hubiera falsos positivos en el recorrido. En la figura 7.7 puede observarse el camino seguido. Al igual que las pruebas en el campus universitario, éste fue hecho en repetidas ocasiones de ida y de regreso.

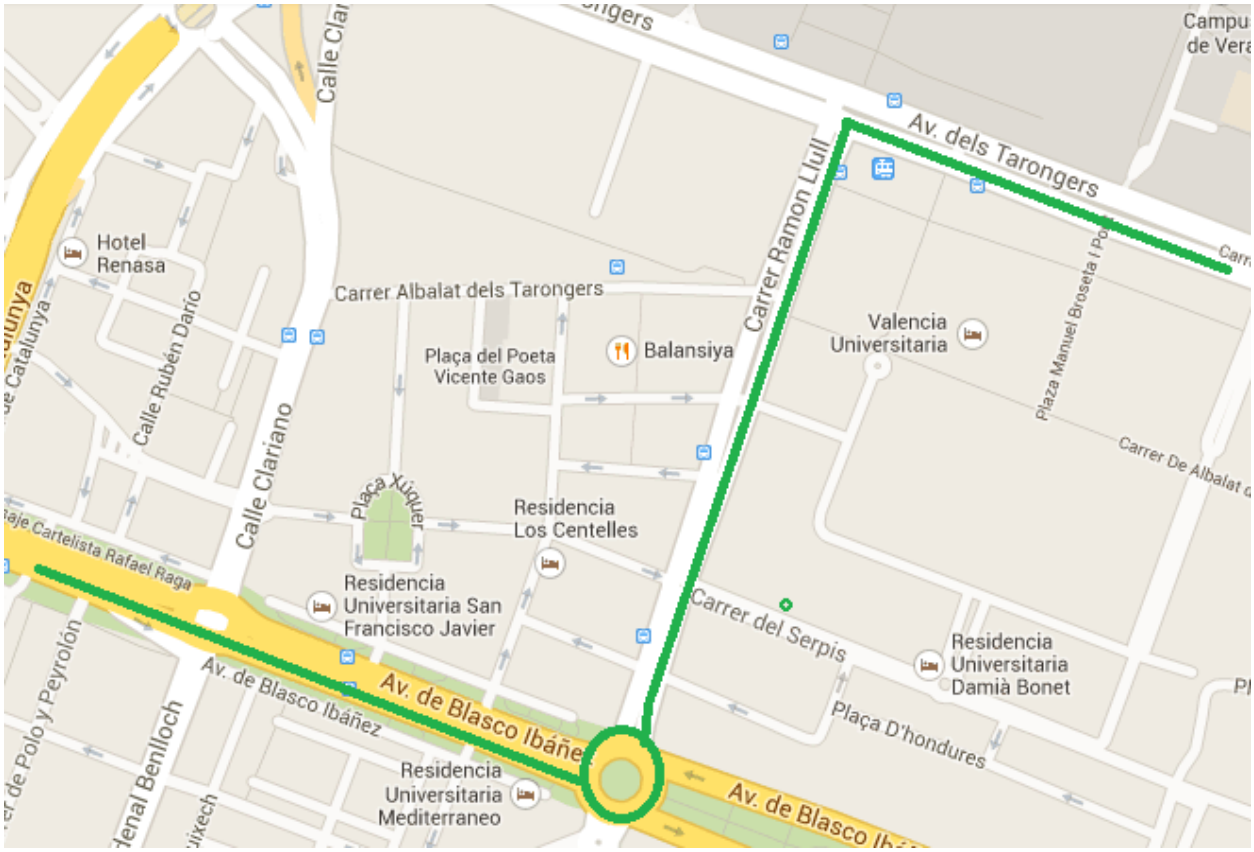


Figura 7.7 Recorrido en autobús buscando falsos positivos

De manera análoga al experimento en coche, el dispositivo fue desplegado de forma y dirección similar, con la diferencia de que el éste fue colocado sobre una pierna como se muestra en la figura 7.8 y de que tuvo que sostenerse en todo momento con una mano para evitar que se cayera.

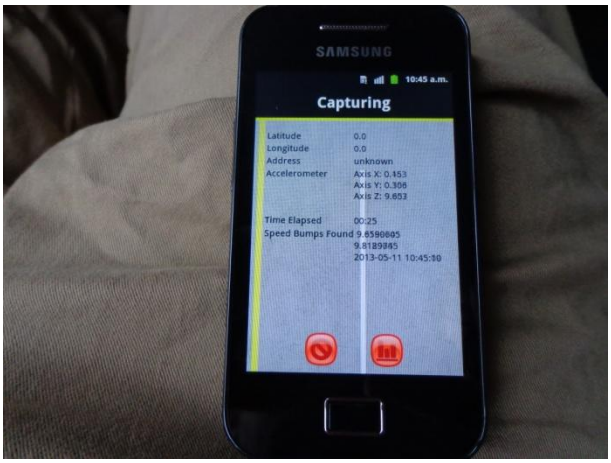


Figura 7.8 Despliegue de dispositivo en autobús

El recorrido se realizó con un umbral de desviación estándar por defecto de 1.9, idéntico al utilizado en el coche anteriormente. Los resultados se muestran a continuación:

Umbral Desv. Est.	Falsos Positivos
1.9	8

Sin duda alguna demasiados falsos positivos en un terreno con ningún badén circulado. Esto se debe a que aún en un terreno regular como el del recorrido, el autobús vibra más y realiza movimientos oscilatorios más notables en comparación con el coche.

Se probó a aumentar los umbrales de detección, obteniendo los siguientes resultados.

Umbral Desv. Est.	Falsos Positivos
2	7
2.5	3
3	0

Como puede observarse, se tuvo que incrementar considerablemente el umbral de detección para no obtener falsos positivos en el experimento del autobús.

Es evidente que el umbral utilizado en las pruebas en coche no es apropiado para realizarlas en un autobús, por lo que tendrán que utilizarse valores mayores. Lamentablemente no se han encontrado recorridos en autobús donde se puedan capturar badenes, por lo que al momento de redacción de este trabajo, resulta imposible determinar si hay que realizar ajustes al algoritmo para que este sirva en la detección de badenes en este tipo de vehículo.

8. Conclusiones

Como se mencionó en la introducción, la computación ubicua requiere la creación de aplicaciones que sean conscientes del contexto, en el caso de esta aplicación, la ubicación geográfica del usuario, el uso del acelerómetro para la detección de badenes y también del sensor de orientación para mostrar al usuario en el mapa.

La computación ubicua también requiere de objetos inteligentes, o procesadores embebidos como los de nuestros teléfonos móviles y de tecnologías inalámbricas que conecten a todos los componentes.

Se ha desarrollado una aplicación distribuida que tiene las características mencionadas y que funciona de acuerdo a los objetivos propuestos al inicio de este trabajo. Se han utilizado varias de las tecnologías disponibles en los dispositivos móviles de la actualidad para desarrollar una aplicación que ayude a prevenir accidentes.

Del lado del cliente se tiene a Android, el sistema operativo para dispositivos móviles más extendido en la actualidad, tanto en número de terminales como en número de desarrolladores. Su núcleo Linux está probado desde hace años y es muy fiable. Proporciona un entorno de desarrollo amigable, bien documentado y muy completo, si bien uno de los grandes problemas de la plataforma es la fragmentación, un problema que persiste en la actualidad y que de acuerdo a las últimas tendencias seguirá persiguiendo a la plataforma.

Por el lado del servidor se trabajó con Google App Engine, una plataforma como servicio que prácticamente ofrece a los desarrolladores e investigadores independientes cómputo ilimitado, algo que hace algunos años era totalmente imposible. Por otra parte y a pesar de todas sus ventajas, la

plataforma GAE todavía dista de ser perfecta ya que tiene algunas restricciones, mencionadas anteriormente en el capítulo 2.

8.1. Limitaciones

A lo largo del desarrollo del proyecto se han encontrado algunas limitaciones y retos con los que se ha tenido de coexistir o simplemente hacerlos a un lado con la esperanza de que puedan ser subsanados en proyectos futuros.

Por un lado, se ha echado en falta más tiempo para desarrollar la solución. Hay varias cosas que pudieron haber tenido un estudio más profundo. En la parte del cliente se realizaron varios experimentos, pero limitados al campus de la UPV y al uso de un solo tipo de vehículo para la detección. Sin lugar a dudas, el coche es el entorno primario bajo el que se ha desarrollado la aplicación ya que proporciona un ambiente más fiable y autonomía al dispositivo respecto al consumo de batería, pero como se ha ejemplificado en el experimento del recorrido del autobús, los datos recolectados no eran suficientes para realizar la detección en otro tipo de vehículos.

Ya que solamente se circuló por el campus universitario, no se encontraron caminos en mal estado o con baches. Si se hubiera podido capturar la información de ese tipo de caminos, tal vez se hubiera podido hacer un algoritmo más sofisticado con la capacidad de distinguir la diferencia entre badenes y baches.

Por otra parte, no se hizo un estudio de la precisión en los sensores de los dispositivos móviles, particularmente el acelerómetro y de esta forma optimizar la detección o simplemente validar que es fiable en cualquier tipo de dispositivo.

Otra cosa que sin lugar a dudas hubiera beneficiado enormemente a la aplicación, es el uso de una conexión a internet continua ya que se tuvo que enviar la información de los badenes de forma asíncrona al servidor. Además la falta de conexión afecta al modo de prevención de accidentes, ya que si no se dispone de ella cada 2 km no se podrá obtener la lista de badenes actualizada del servidor.

En la parte del servidor también hubo cosas que pudieron ser optimizadas. Por una parte las consultas al *datastore* tienen que ser muy simples, ya que JDO no permitía búsquedas utilizando más de una propiedad como parámetro. En consecuencia los resultados de las consultas eran muy

grandes por lo que se tuvo que hacer procesamiento adicional en el *servlet*. No se hizo uso de *Memcache*, algo que hubiera mitigado el efecto de las consultas tan grandes que se hacen.

Otra mejora que derivaría en una aplicación más sofisticada sería la implementación de un mecanismo automático en el servidor que pudiera eliminar badenes en el *datastore* para los que no se hayan recibido positivos en un determinado periodo de tiempo.

8.2. Futuros trabajos

En la unión europea existe un fuerte empuje por el desarrollo de sistemas de transporte inteligentes con dos motivaciones principales, la seguridad y la eficiencia del transporte. Las ciudades inteligentes también son uno de los objetivos a largo plazo de la unión europea y de varias compañías de tecnología en general. Los dispositivos móviles y las tecnologías de las que son provistos jugarán un papel esencial hacia el cumplimiento de estos objetivos.

Los sensores tienen múltiples aplicaciones y no tienen que estar limitados al acelerómetro o a la detección de accidentes en la carretera como se ha hecho en este trabajo. Un ejemplo sería la cooperación distribuida entre usuarios para notificar detectores de velocidad en la autopista.

Este tipo de trabajos también se verá ampliamente beneficiados con la investigación y avances en las comunicaciones inalámbricas. Los dispositivos eventualmente incorporarán tecnologías con mayor rango de transmisión, como WAVE, WiMAX y LTE. Sin duda alguna, el uso extendido de estas tecnologías dará lugar a aplicaciones más creativas y beneficiosas para el ser humano.

9. Bibliografía

1. **Sistemas distribuidos: conceptos y diseño** (George F. Coulouris)
2. **Cloud computing: a practical approach** (Anthony T. Velte)
3. **Programming Google App Engine** (Dan Sanderson)
4. **Android Programming** (Nicolas Gramlich)
5. **Professional Android Application Development** (Reto Meier)
6. **Android Developers**, http://developer.android.com/index.html
7. **GAE developers**, https://developers.google.com/appengine/
8. **Stack Overflow**, http://stackoverflow.com/
9. **A Chart Engine**, http://www.achartengine.org/index.html
10. **Wikipedia**, http://www.wikipedia.org/
11. **Developer Nokia**, http://developer.nokia.com/
12. **Developer BlackBerry**, https://developer.blackberry.com/
13. **Developer Microsoft**, http://msdn.microsoft.com/en-us/
14. **Developer Tizen**, https://developer.tizen.org/
15. **Amazon Web Services**, http://aws.amazon.com/es/elasticbeanstalk/

16. **Windows Azure**, http://www.windowsazure.com/es-es/

Artículos Consultados

1. ASLAN, S., KARCIOGLU, O., KATIRCI, Y., KANDI, H., EZIR- MIK, N., AND BILIR, O. Speed bump induced spinal column in- jury. The American Journal of Emergency Medicine 23, 4 (2005), 563 – 564.
2. BOWREY, D., THOMAS, R., EVANS, R., AND RICHMOND, P. Road humps: accident prevention or hazard? Journal of accident & emergency medicine 13, 4 (07 1996).
3. Mohit Jain Ajeet Pal Singh; Soshant Bali Sanjit Kaul, "Speed-Breaker Early Warning System," USENIX Federated Conferences Week, 2012
4. Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. 2008. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (SenSys '08). ACM, New York, NY, USA, 323-336.
5. Mednis, A.; Strazdins, G.; Zviedris, R.; Kanonirs, G.; Selavo, L., "Real time pothole detection using Android smartphones with accelerometers," Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on , vol., no., pp.1,6, 27-29 June 2011 doi: 10.1109/DCOSS.2011.5982206
6. Bhoraskar, R.; Vankadhara, N.; Raman, B.; Kulkarni, P., "Wolverine: Traffic and road condition estimation using smartphone sensors," Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on , vol., no., pp.1,6, 3-7 Jan. 2012 doi: 10.1109/COMSNETS.2012.6151382
7. Ming Liu, "A Study of Mobile Sensing Using Smartphones," International Journal of Distributed Sensor Networks, vol. 2013, Article ID 272916, 11 pages, 2013. doi:10.1155/2013/272916
8. Roxin, A.; Gaber, J.; Wack, M.; Nait-Sidi-Moh, A., "Survey of Wireless Geolocation Techniques," Globecom Workshops, 2007 IEEE , vol., no., pp.1,9, 26-30 Nov. 2007 doi: 10.1109/GLOCOMW.2007.4437809