UNIVERSIDAD DE LA REPÚBLICA

Facultad de Ingeniería

Instituto de Computación

"Redes cooperativas móviles y su aplicación al transporte público"

Informe de Proyecto de Grado

Vicente Acosta

Juan Pablo Revello

Ignacio Tisnés

Tutor

Ariel Sabiguero

Tribunal

Eduardo Grampín

Antonio Mauttone

Sebastián Pizard

Mayo 2013

Resumen

El presente trabajo de grado, enmarcado dentro del proyecto Stic-AmSud CUDEN, propone una solución colaborativa para la ubicación en tiempo real de la flota de transporte de una ciudad.

Se implementó un prototipo funcional basado en una aplicación Android que permite a sus usuarios compartir información sobre los ómnibus usados por éstos. A partir de esta información, un servidor central construye un mapa en tiempo real con las posiciones de los ómnibus de la ciudad, que puede ser consultado desde la aplicación.

El servidor central se encarga de recibir y resumir la información provista por los usuarios, para luego distribuirla entre los que la requieran. Se diseñó un mecanismo eficiente de distribución de información para grupos de usuarios que comparten un interés. Este interés puede ser conocer la ubicación de los ómnibus de una línea determinada.

La aplicación explota aspectos de redes sociales, permitiendo el intercambio de mensajes dentro de los grupos de usuarios. Estos mensajes pueden proveer información adicional a la ubicación, como por ejemplo informar sobre retrasos o desvíos en la ruta, o pueden ser con fines sociales.

Para facilitar el uso de la aplicación se explotan los sensores presentes en los dispositivos para la detección automática de eventos relevantes al sistema. Los eventos detectados son: la llegada del usuario a la parada, el inicio de un viaje en ómnibus y su fin.

La solución fue diseñada para poder ser utilizada en cualquier ciudad, ya que no se basa en datos provistos por gobiernos o empresas locales. La información sobre las líneas existentes, sus recorridos y paradas, es provista por los propios usuarios del sistema.

Luego de su implementación, distribución como prototipo en una población voluntaria y recopilación de datos, creemos que su evolución a una aplicación funcional es técnicamente posible. Dado que el proceso de recopilación de información es cooperativo, es requerido un grupo de usuarios significativo para que la misma sea útil. Un problema que no fue resuelto en este proyecto refiere a cómo motivar a los usuarios en el uso de la aplicación.

Palabras clave: *Dispositivos Móviles, Android, Sistemas Colaborativos, Transporte Público, Acelerómetro, GPS, Sistemas Distribuidos, Redes Sociales.*

Índice general

Glosario

Sandbox Mecanismo para ejecutar múltiples programas de forma segura, restringiendo los recursos a los que estos programas pueden acceder.

3G Tercera generación de transmisión de voz y datos a través de telefonía móvil.

API Application Programming Interface - Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Arquitectura ARM Arquitectura de computadores desarrollado por ARM Holdings, apliamente usada en dispoditivos móviles por su bajo costo de fabricación y bajo consumo energético.

Arquitectura Intel Arquitectura de computadores ampliamente difundida (en particular, casi todos los sistemas Windows usan esta arquitectura), de tipo CISC. Creada por la empresa con el mismo nombre, pero implementada también por otras. También llamada arquitectura x86.

ASCII American Standard Code for Information Interchange - Es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno y en otras lenguas occidentales.

CUTCSA. Compañía Uruguaya de Transportes Colectivos S.A. - Empresa privada dedicada a proveer servicios de transporte en la ciudad de Montevideo, Uruguay.

DBMS Database Management System - Componente de software dedicado al manejo de bases de datos.

GMT Greenwich Mean Time - El tiempo medio de Greenwich es un estándar de tiempo que originalmente se refería al tiempo solar medio en el Real Obser-

vatorio de Greenwich, en Greenwich, Inglaterra, y se convirtió más tarde en un estándar global de tiempo.

GSM Groupe Spécial Mobile - Es un sistema estándar, libre de regalías, de telefonía móvil digital.

g La intensidad del campo gravitatorio o la aceleración de la gravedad

HTTPS Hypertext Transfer Protocol Secure - Protocolo usado en Internet para la transmisión de hipertexto de forma segura.

HTTP Hypertext Transfer Protocol - Protocolo usado en Internet para la transmisión de hipertexto.

IEEE Greenwich Mean Time - El tiempo medio de Greenwich es un estándar de tiempo que originalmente se refería al tiempo solar medio en el Real Observatorio de Greenwich, en Greenwich, Inglaterra, y se convirtió más tarde en un estándar global de tiempo.

IETF Internet Engineering Task Force - Es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad

iOS Sistema operativo desarrollado por Apple orientado a dispositivos móviles, principalmente teléfonos y tabletas.

JPA Java Persistence API

JSON JavaScript Object Notation - Es un formato ligero para el intercambio de datos.

Licencia BSD Es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva.

Linux Linux es un sistema operativo libre y de código abierto similar a Unix. El término es usado para denominar también al núcleo de este sistema operativo.

MariaDB DBMS libre y de código abierto, basado en MySQL.

MQTT Message Queue Telemetry Transport - Protocolo diseñado por IBM para la transferencia de información de telemetría entre dispositivos.

OAuth Open Authorization - Es un protocolo abierto que permite autorización segura de un API de modo estándar y simple para aplicaciones de escritorio, móviles y web.

P2P Peer-to-peer - Mecanismo de comunicación entre dispositivos en una red que se caracteriza por ser distribuído, donde cada componente puede actuar tanto como cliente como servidor.

REST Representational State Transfer - Estilo de arquitectura de software para sistemas distribuidos como la World Wide Web.

SDK Software Development Kit - Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado.

SMS Short Message Service - Sistema incorporado en el estándar GSM para permitir el envío de mensajes de texto entre teléfonos celulares.

TCP Transmission Control Protocol - Protocolo usado ampliamente en Internet en la capa de transporte.

UTF-8 8-bit Unicode Transformation Format - Es un formato de codificación de caracteres Unicode e ISO 10646 utilizando símbolos de longitud variable. Actualmente es una de las tres posibilidades de codificación reconocidas por Unicode y lenguajes web, o cuatro en ISO 10646.

UUID Universally Unique Identifier - Estándar para la construcción distribuida de identificadores, garantizando la unicidad de estos en condiciones normales.

Wi-Fi Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Es una marca de la Wi-Fi Alliance, la organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados a redes inalámbricas de área local.

XML eXtended Markup Language - Es un lenguaje que define un conjunto de reglas para la codificación de documentos en un formato que es a la vez leíble por un humano y una computadora.

Capítulo 1

Introducción

Este documento es un informe de proyecto de grado de la carrera Ingeniería en Computación, realizado en la Facultad de Ingeniería de la Universidad de la República de Uruguay.

Este proyecto fue motivado por el reciente aumento en el acceso a dispositivos móviles inteligentes, y la masificación del uso de las redes sociales. El objetivo principal es explotar estos hechos para construir una solución aplicada a un problema particular, en este caso mejorar la experiencia de los usuarios en el uso del transporte público.

El enfoque dado al proyecto es práctico, por lo que se implementaron prototipos para evaluar la aplicabilidad de la solución en contextos reales. Estos prototipos fueron probados por usuarios voluntarios, entre ellos estudiantes de Facultad de Ingeniería.

1.1. Problemática y motivación

El uso de dispositivos inteligentes en la sociedad ha tenido un gran incremento en los últimos años. Poseen una diversa cantidad de sensores y tienen un gran poder de cómputo, lo que los transforma en una pequeña computadora que acompaña al usuario durante la mayor parte del día.

Por otro lado las redes sociales han captado la atención de millones de personas de todo el mundo, facilitando la creación de comunidades de usuarios con intereses u objetivos en común. La existencia de estos grupos de usuarios conlleva un intercambio constante de información.

Este proyecto, enmarcado dentro del proyecto Stic-AmSud CUDEN[1], pretende explotar las capacidades de los dispositivos inteligentes con el fin de brindar un servicio sobre una infraestructura construida de forma cooperativa entre todos los usuarios, formando además grupos colaborativos de usuarios que comparten información en pos de objetivos comunes.

En particular, se busca mejorar la experiencia de los usuarios en el sistema de transporte público, brindando información precisa sobre el posicionamiento de la flota de transporte, minimizando el tiempo de espera y permitiendo el intercambio social entre grupos de usuarios.

Se enfocó el desarrollo de este proyecto únicamente en los dispositivos inteligentes con sistema operativo Android, dado que en su inicio se contaba con dispositivos de estas características para experimentar.

1.2. Conocimientos previos

El presente documento está orientado a estudiantes o profesionales de carreras afines a Ingeniería en Computación, con conocimientos sobre redes de computadores e ingeniería de software.

Los lectores con experiencia en el desarrollo de aplicaciones para dispositivos móviles podrán entender más rápidamente ciertas secciones del documento, aunque el conocimiento en esta área no es requerido.

1.3. Objetivo general

El objetivo general del presente proyecto, al igual que del proyecto Stic-AmSud CUDEN, es proveer un soporte para comunidades cooperativas entre dispositivos, con el fin de facilitar actividades colaborativas mejorando la experiencia del usuario.

1.4. Objetivos específicos

Alineado con el objetivo general, el objetivo principal del presente proyecto es construir un sistema colaborativo aplicado al sistema de transporte público.

En el marco de la construcción de este sistema, se definieron los siguientes objetivos:

[1] http://www-npa.lip6.fr/cuden/

- Investigar formas en las que los usuarios pueden compartir información sobre la ubicación de los ómnibus en los que se encuentran con el fin común de mejorar su experiencia en el sistema de transporte.
- Permitir que los usuarios compartan información adicional sobre el estado del ómnibus, por ejemplo, si viene con muchas personas, si tuvo algún desvío, o si existe algún desperfecto/atraso.
- Construir un mapa en tiempo real de la posición de los ómnibus para una ciudad, haciendo uso de los dispositivos inteligentes de los usuarios.
- Investigar las posibilidades de la plataforma Android para el uso de los sensores incluidos en el dispositivo, para determinar entre otras cosas la posición geográfica del usuario.
- Evaluar la factibilidad de detectar eventos relevantes al sistema haciendo uso de los sensores de los dispositivos.
- Permitir que el sistema pueda funcionar en varias ciudades simultáneamente.
- Generar alertas de proximidad en distancia y/o tiempo de un ómnibus, o alertas de llegada a un destino indicado por el usuario.

1.5. Organización del documento

Los siguientes capítulos del documento están organizados de esta manera:

En el Capítulo 2 se presentan características y estadísticas sobre el uso de dispositivos inteligentes, el estado de la plataforma Android y sus proyecciones, y por último una presentación de las aplicaciones móviles existentes aplicadas al sistema de transporte público.

En el Capítulo 3 se describe el problema a resolver, mostrando una visión general sobre el comportamiento esperado del sistema en el largo plazo, y algunos requerimientos particulares para el proyecto presentado.

En el Capítulo 4 se describe la solución implementada, comenzando por definir el alcance del proyecto actual, y continuando con una comparación sobre las aplicaciones existentes aplicadas al transporte público. Luego se presenta cómo se modeló la red de transporte, continuando con la arquitectura del sistema, indicando cómo interactúan los diferentes actores: el servidor central y los dispositivos inteligentes. Más adelante, se presenta el comportamiento de la aplicación en los dispositivos

inteligentes, y las diferentes formas de comunicación entre los dispositivos y el servidor central. Por último, se presentan las heurísticas incluidas en la aplicación, cuyo objetivo es la detección automática de eventos relevantes al sistema.

En el Capítulo 5 se presentan algunos puntos relevantes sobre la gestión del proyecto, empezando por las experiencias sobre el aprendizaje de Android, y finalizando por la descripción de la evolución del proyecto en cuanto al desarrollo del sistema.

En el Capítulo 6 se presentan resultados prácticos sobre el uso de la aplicación, poniendo énfasis en los datos obtenidos con las versiones liberadas del producto y las pruebas realizadas sobre la heurística de descenso de ómnibus.

En el Capítulo 7 se exponen las conclusiones del proyecto realizado.

En el Capítulo 8 se muestran posibles líneas de trabajo futuro, algunas presentes en la visión a largo plazo del proyecto.

Adicionalmente, se presentan varios anexos con el objetivo de profundizar sobre ciertos temas e incluir información detallada sobre ciertos aspectos del proyecto.

Capítulo 2

Estado del arte

En este capítulo se presenta el estado actual de las áreas de desarrollo tecnológico e investigación relacionadas a este proyecto.

En la primer sección, se presenta un breve resumen de los últimos avances en materia de desarrollo de dispositivos móviles inteligentes, mostrando la razón por la que es atractiva la posibilidad de crear aplicaciones para estas plataformas.

En segundo lugar, se incluye una breve reseña de Android. Se presenta una breve reseña histórica de este sistema operativo, para pasar a mencionar las características actuales del mismo, contando ventajas y desventajas de éste como plataforma de desarrollo.

Finalmente, se hace mención a aplicaciones existentes para plataformas móviles aplicadas al sistema de transporte.

2.1. Dispositivos inteligentes

Un teléfono inteligente es un teléfono móvil con un sistema operativo que permite la instalación de software de una manera similar a lo que sucede con las computadoras personales. Antes, el teléfono tenía los contenidos que el fabricante incluía, y la posibilidad de añadir nuevas funcionalidades era restringida. Actualmente, los teléfonos inteligentes permiten agregar nuevas funcionalidades de una manera sencilla a través de tiendas de aplicaciones virtuales, como Google Play para Android o AppStore para iOS.

Los teléfonos inteligentes poseen una gran variedad de sensores, como son: GPS, acelerómetro, giroscopio, brújula, sensor de luz. Tienen APIs que permiten acceder a los datos de estos sensores, poniéndolos disponibles al uso de aplicaciones en el

teléfono. También se poseen entornos de desarrollo para facilitar la programación y el diseño.

Tienen a su vez potentes procesadores, como por ejemplo el Samsung Exynos 4 Quad[1] basado en el procesador de aplicaciones móviles Quad ARM Cortex-A9 1.4/1.6GHz[2]. También pueden poseer almacenamiento tanto interno como externo (tarjetas SD) de hasta 64 GB.

Por ejemplo, la versión 4 de Google Nexus, tiene hasta 16 GB de memoria incorporada en el teléfono, 2 GB de RAM y una CPU de 4 núcleos de 1.5 GHz (Qualcomm Snapdragon™ S4 Pro CPU)[1, 2].

Las tabletas poseen las mismas características que los teléfonos inteligentes, y se diferencian por tener un mayor tamaño de pantalla, comúnmente mayor a siete pulgadas. Al igual que los teléfonos inteligentes, son consideradas dispositivos inteligentes.

Más allá de las características técnicas del hardware, es importante destacar el cambio que se presenta en la interacción entre el usuario y el dispositivo. En general, los dispositivos inteligentes suelen ser usados por un único usuario, por lo que el contenido a mostrarse puede ser personalizado completamente para éste. En el caso de los celulares, puede explotarse el hecho de que el usuario del dispositivo tiene un contacto casi permanente con el dispositivo.

Para tener una idea de la explosión en las ventas de los dispositivos inteligentes, en el último trimestre del 2011 se vendieron alrededor de 149 millones de teléfonos inteligentes a usuarios finales, lo que significa un crecimiento de 47,3 % en comparación al mismo periodo del 2010. El número total de unidades vendidas durante el año 2011 alcanzó los 472 millones [3]. En el último trimestre del año 2012 se alcanzo la cifra récord de ventas de 207.7 millones de unidades vendidas, lo que significa un aumento de 38,3 % en comparación al mismo periodo del 2011.

2.2. Plataforma Android

Android es un sistema operativo libre y de código abierto orientado a dispositivos móviles desarrollado originalmente por Android Inc. y actualmente mantenido por la Open Handset Alliance, liderada por Google.

Este sistema operativo está basado en el núcleo de Linux, y posee varios componentes desarrollados sobre este núcleo. Este sistema operativo permite la instalación

[1] http://www.samsung.com/global/business/semiconductor/minisite/Exynos/products4quad.html
[2] http://www.arm.com/products/processors/cortex-a/cortex-a9.php

de aplicaciones desarrolladas por terceros.

Estas aplicaciones deben ser desarrolladas en lenguaje Java, que es ejecutado por una máquina virtual llamada Dalvik, diferente a nivel de *bytecode* a la del estándar Java. El *framework* asociado al sistema operativo posee varias APIs para acceder a muchos de los sensores presentes en los dispositivos, por ejemplo GPS o acelerómetro [39].

El sistema tiene varios mecanismos para garantizar la seguridad de los datos de los usuarios aún permitiendo la incorporación de aplicaciones de distintos proveedores. En primer lugar se implementa un *sandbox* a nivel de procesos, basado en el sistema de usuarios, grupos, y permisos de Linux. A cada aplicación se le asigna un usuario Linux, con permisos para acceder solamente a sus propios archivos, asegurando independencia en los datos.

Android incorpora además un sistema de permisos orientados a limitar el acceso de la aplicación a las diversas funciones del dispositivo. Estos permisos pueden restringir por ejemplo el acceso a Internet, o el acceso a los diversos sensores. Cuando el usuario va a instalar una aplicación, se despliega la información de cuáles permisos desea poseer la aplicación, y el usuario debe autorizar el acceso a estos permisos antes de que se efectúe la instalación. Con este mecanismo, se busca que las aplicaciones que deban acceder a información sensible, o deban acceder a funciones del dispositivo que pueden tener efectos negativos sobre su funcionamiento, o deban realizar operaciones que pueden tener un costo monetario estén explícitamente autorizadas por el usuario para hacer estas acciones.

Android posee una importante comunidad de desarrolladores distribuida globalmente, que pueden distribuir sus aplicaciones a través de Google Play. Esta tienda de aplicaciones sigue la tendencia actual para la distribución de aplicaciones, iniciada por la App Store de Apple, y aplicada también en el *Marketplace* de Microsoft. En estas tiendas, los fabricantes de software pueden publicar aplicaciones que luego serán descargadas por los usuarios finales en sus dispositivos usando la conexión a Internet que poseen. Las aplicaciones publicadas pueden ser gratis o pagas, en cuyo caso la empresa administradora de la tienda de aplicaciones generalmente obtiene una porción del precio de venta.

Este sistema operativo tuvo una adopción muy rápida en la industria y en los consumidores finales, llegando a tener el 50 % del mercado de teléfonos inteligentes a fines de 2011 y 2012 [3, 4]. No se encontraron estadísticas de cuotas de mercado específicas para el mercado uruguayo, pero se puede esperar que la adopción de este sistema operativo en nuestro país sea rápida dada la existencia de modelos

económicos con este sistema operativo. La creciente oferta de dispositivos Android en el mercado es un claro indicador de este hecho.

Debido a que es libre y de código abierto, y es elegido por diversos fabricantes como sistema operativo para sus dispositivos, se crearon varias versiones adaptadas de Android. Esto, sumado a que muchos fabricantes no actualizan el sistema operativo que contienen modelos antiguos, y que Google libera versiones nuevas del sistema operativo periódicamente, aproximadamente 1 cada 4 meses (13 versiones desde su primer versión comercial liberada en febrero del 2009 hasta su última versión 4.2 liberada en octubre del 2012, como se detalla en el Anexo E), ha llevado a una gran fragmentación del sistema operativo.

Este es uno de los problemas más grandes que posee la plataforma actualmente. La fragmentación puede verse desde dos puntos de vista: por un lado, la existencia de muchas versiones (y por lo tanto muchas APIs) que están activas en un momento dado; por otro, la fragmentación a nivel de hardware, ya que modelos distintos, aún corriendo la misma versión del sistema operativo, pueden poseer hardware distinto en términos de pantalla, sensores, o incluso en las CPU's utilizadas (existen actualmente versiones de Android disponibles para arquitecturas Intel y ARM).

2.3. Aplicaciones existentes aplicadas al sistema de transporte

Como se mencionó en el Capítulo 1, uno de los objetivos de este proyecto es construir un sistema para mejorar la experiencia de los usuarios que usan el sistema de transporte público.

Se encontraron varias aplicaciones desarrolladas sobre plataformas móviles que proveen servicios aplicados a este dominio, tanto locales como aplicadas a otras ciudades. A continuación se presenta una breve reseña de cada una de éstas, resaltando los aspectos considerados importantes.

Cómo ir - A qué hora pasa Estas dos herramientas fueron desarrolladas por la Intendencia de Montevideo, con el objetivo de brindar al usuario información sobre las diferentes líneas de ómnibus, sus recorridos y paradas, y cómo usarlas para formar una ruta hacia un determinado destino. La información presentada es estática, mostrando los horarios teóricos en los que deberían pasar los ómnibus [5].

"A qué hora pasa" es una aplicación web recientemente desarrollada orientada al uso en celulares, que provee la misma información que "Cómo ir" con la diferencia de que en este caso se aprovechan los sensores del dispositivo para encontrar su ubicación.

STM Montevideo STM Montevideo es una aplicación disponible en Google Play. Su objetivo es proveer los horarios de los ómnibus del Sistema de Transporte Metropolitano de Montevideo (STM). Funciona a partir de las páginas web de la Intendencia de Montevideo y de la empresa CUTCSA, persistiendo los datos presentes en éstas en una base de datos local. Al persistir sus datos localmente, se logra que la aplicación pueda ser usada sin conexión a Internet [6].

Posee algoritmos de ruteo para mostrar las líneas de ómnibus desde un origen a un destino, indicados por el usuario en un mapa. Esta aplicación hace uso del sensor de GPS o redes Wi-Fi para conocer la ubicación del usuario, pero no ofrece la posibilidad de marcarla como un origen o un destino válido.

GXbus GXbus es una aplicación disponible tanto en la Play Store como en el AppStore de iOS. Posee funcionalidades similares a la aplicación anterior, con algunas diferencias. Al igual que STM Montevideo, GXbus muestra información obtenida de la Intendencia y de la empresa CUTCSA, y posee algoritmos de ruteo. Como principal desventaja frente a STM Montevideo, esta aplicación debe estar siempre conectada a Internet para funcionar. La única ventaja es que, además de conocer la posición del usuario, ofrece la posibilidad de indicarla como un posible origen o destino [7].

Google Transit Google Transit es un sistema provisto por Google integrado a Google Maps, que permite a compañías o gobiernos hacer pública información sobre paradas, recorridos, y horarios de sus sistemas de transporte público [9]. Las compañías deben proveer esta información en el formato GTFS (*General Transit Feed Specification* por sus siglas en inglés) [8], definido por Google. Esta información es procesada por el sistema, integrándolo a Google Maps como una capa de información extra donde se muestran las paradas y recorridos, y además como entrada para los algoritmos de navegación incluidos en el sistema, que permiten al usuario obtener qué rutas debe tomar para llegar a determinado destino, y en qué parada subirse y bajarse. Los recorridos determinados por Google Transit pueden incluir trayectos en distintos medios de transporte públicos, por ejemplo una parte del recorrido en

ómnibus y el resto en tren. Por la arquitectura elegida, este sistema está orientado a proveer información estática de recorridos y paradas, no información en tiempo real del estado de la flota.

iBus iBus se diferencia de las aplicaciones anteriores principalmente por no tener un componente de software específico instalado en los terminales. Está disponible en Montevideo, y es provista de forma paga por la empresa telefónica Movistar. La aplicación funciona basada en comunicaciones por SMS (lo que permite su uso por dispositivos antiguos). Al no tener componentes instalados en los terminales, no utiliza sensores del dispositivo. Sin embargo, obtiene la posición del usuario basado en la intensidad de señal recibida del teléfono por las antenas celulares cercanas [10].

Frente a las anteriores, tiene la gran diferencia de mostrar información basada en datos en tiempo real. A diferencia de la anterior, la información mostrada no son horarios, sino tiempos de espera para una línea de ómnibus particular en una parada particular.

One Bus Away Esta aplicación es la primera encontrada realizada dentro de un marco académico. Fue desarrollada en 2009 dentro de la Universidad de Washington. Posee varias formas de interacción con el usuario, entre ellas una línea telefónica, un sistema basado en SMS, una página web, y aplicaciones para varios sistemas operativos móviles, entre ellos Android. Debido a esto, una mayor cantidad de usuarios puede acceder al sistema.

Provee al usuario de información en tiempo real sobre el estado del sistema de transporte para el área de Seattle, partiendo de información provista por las compañías de transporte locales. En las aplicaciones móviles, incorpora también mecanismos de notificación a los usuarios indicando eventos que puedan afectarlos, por ejemplo el desvío o cancelación temporal de una línea, o el atraso de un ómnibus particular [11, 12].

Investigación en Nanyang Technological University, Singapur En esta universidad se llevó a cabo un proyecto donde se utilizaron sensores de teléfonos celulares de usuarios para colaborativamente determinar la posición de ómnibus en su ruta y determinar el tiempo de espera en la parada para usuarios haciendo uso del servicio [13].

Se usaron las antenas celulares para determinar en qué parte de su ruta se encuentra el ómnibus en el que se encuentra el usuario. Dada esta información, y

conociendo la velocidad promedio del ómnibus, se obtiene el tiempo de espera para usuarios ubicados en las siguientes paradas en la ruta del ómnibus.

Para lograr la detección de la presencia del usuario en un ómnibus, sin que éste lo indique, se realizó un análisis de las intensidades de señal de las antenas detectadas por el celular, y se elaboró un mecanismo en el que el teléfono tiene su micrófono abierto, y analiza el sonido ambiente buscando el sonido característico de la máquina expendedora de boletos. Esta información es procesada completamente dentro de un servidor central.

2.4. Resumen

En este capítulo se presentaron los dispositivos inteligentes como plataformas de desarrollo, poniendo énfasis en los dispositivos Android. Puede concluirse que estos dispositivos constituyen entornos interesantes para el desarrollo de sistemas centrados en el usuario que exploten los sensores presentes en el dispositivo.

Por otro lado, se presentaron aplicaciones ya existentes aplicadas al dominio del problema que será atacado. Éstas presentan distintos enfoques al problema de mejorar la experiencia de los usuarios en el transporte público, variando su uso de sensores, y el origen de la información usada. En el próximo capítulo se presentará el problema que se aborda en este proyecto, y más adelante la solución planteada a este problema. En este momento, al definir el alcance del proyecto, se volverá a estas aplicaciones para indicar las diferencias entre éstas y la aplicación a construir.

Capítulo 3

Descripción del problema

En este capítulo se presenta la descripción teórica del problema abordado en este proyecto. En la primer sección se presenta una visión general del problema, para pasar luego a describir en la segunda sección requerimientos específicos del proyecto.

3.1. Visión general

Con el fin de comprender mejor la idea y problemática del proyecto se optó por dividir esta sección en tres subsecciones. Inicialmente, se presentan los pilares fundamentales del problema, para mostrar luego un ejemplo práctico de cómo debería comportarse el sistema, y terminar mostrando aspectos complementarios al problema original que lo complejizan.

3.1.1. Problema principal

El problema que se intenta resolver refiere a cómo recolectar información de manera cooperativa y ponerla a disposición de los usuarios del sistema de transporte público. En particular, se trata de ubicar colaborativamente y en tiempo real toda la flota de ómnibus correspondiente al sistema de transporte urbano de una ciudad determinada. La resolución de este problema tiene como objetivo principal mejorar la experiencia del usuario en el uso del sistema de transporte.

La posición de los ómnibus será obtenida a partir de información aportada por los usuarios del sistema, a través de sus dispositivos móviles. Para esto, resultan fundamentales los mecanismos provistos por los dispositivos para determinar su posición y su conexión a Internet, como vía para compartir esta información.

Estos mismos usuarios podrán también consultar al sistema para obtener información sobre el estado de la flota. Es muy probable que en el sistema existan muchos usuarios interesados en la misma información, por lo que se pueden definir grupos de usuarios a partir de estos intereses. Por ejemplo, un grupo de usuarios podría estar definido por el interés en conocer la posición de los ómnibus de una determinada línea. Esto define uno de los principales problemas a resolver: diseñar mecanismos eficientes de entrega de información a grupos de usuarios, que se asemejan a grupos *multicast*. Se quiere resolver esta entrega de mensajes sobre la capa de aplicación.

Además de la información relacionada a la posición de los ómnibus, se proveerán mecanismos para que los usuarios puedan intercambiar mensajes con el fin de mejorar su experiencia de uso del transporte. Por ejemplo, los usuarios podrán enviar mensajes para advertir a otros de desvíos, cambios en rutas, retrasos, o con un fin social.

Resumiendo lo descripto hasta el momento, la premisa principal bajo la que funcionará la aplicación es: los usuarios entregan información relativa a sus viajes en ómnibus, incurriendo en gastos de batería y de uso de la conexión de datos de su dispositivo, a cambio de información en tiempo real de la posición de los ómnibus conocidos por el sistema y un espacio para el intercambio social.

3.1.2. Uso típico de la aplicación

El propósito de la aplicación instalada en el dispositivo del usuario es mejorar su experiencia mientras se encuentra utilizando el sistema de transporte. Una visión simplificada del uso del sistema de transporte, implica que el usuario parte de un cierto punto de la ciudad, se dirige a alguna de las paradas de ómnibus, donde luego tomará un ómnibus. El usuario permanecerá en este ómnibus hasta llegar a una parada destino, desde donde continuará a pie hasta su destino final.

Durante el trayecto hasta la parada de ómnibus, y una vez allí, el usuario podría consultar la posición de los ómnibus conocidos por el sistema para líneas que le sean de utilidad para ese viaje.

Luego de subido al ómnibus, deberá comenzar a compartir su posición, para que el sistema pase a conocer este ómnibus si es que no lo conocía aún. Deberá seguir compartiendo hasta que el usuario descienda del ómnibus, por lo que el sistema deberá conocer de algún modo el momento en que esto ocurra.

En la Figura 3.1 se muestra una ejemplo de uso del sistema.

En el momento 1 mostrado, se tienen tres usuarios que se encuentran esperando

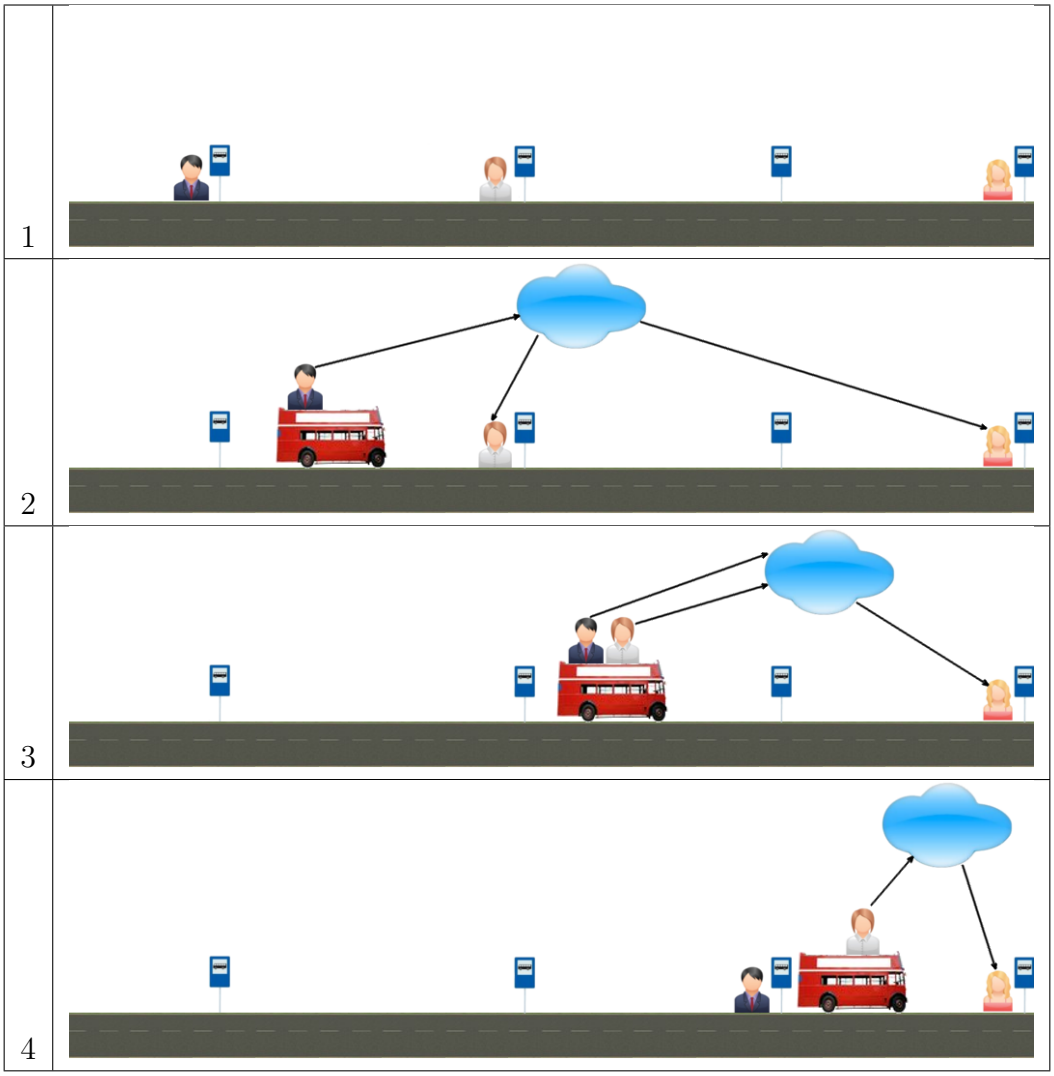


Figura 3.1: Ejemplo de uso del sistema

un ómnibus de la misma línea, en tres paradas distintas. No se tiene información sobre la línea de ómnibus que los tres están esperando.

En el momento 2, se puede ver que el primer usuario subió al ómnibus, y a partir de este momento comenzó a enviar la información sobre dónde se encuentra. Dicha información es transmitida a los otros dos usuarios, que podrán visualizar en un mapa de su dispositivo la posición del ómnibus. Aquí se puede definir un grupo de usuarios, conformado por los tres usuarios que colaboran con un objetivo común: conocer la posición de los ómnibus pertenecientes a una línea particular.

En el momento 3, se ilustra la situación en la que un segundo usuario sube al mismo ómnibus que se encuentra el primer usuario. En este caso, el sistema deberá detectar que si bien existen dos usuarios publicando posiciones de ómnibus, estas

posiciones se corresponden al mismo ómnibus. El tercer usuario continúa recibiendo la información, pero no sabe cuántos usuarios se encuentran compartiéndola, sino que sólo puede ver en su mapa por dónde viene el ómnibus.

Por último, en el momento 4, se presenta la situación en la que el primer usuario llegó a su destino, y desciende del ómnibus. En este momento, el primer usuario deja de compartir la posición del ómnibus, mientras que el segundo usuario continúa con la publicación, y, como en el paso anterior, el tercer usuario (que aún no ha subido), sigue recibiendo la información sobre la ubicación del ómnibus.

3.1.3. Características avanzadas

En esta sección se describen dos características complementarias que complejizan el problema planteado en la Sección 3.1.1.

En primer lugar, se desea que la aplicación esté construida de forma tal que toda la información que maneje sea recolectada de forma cooperativa por los usuarios. Esto implica que el sistema deberá poder iniciar sin ningún conocimiento sobre las líneas de ómnibus, sus recorridos, sus horarios, las ubicaciones de las paradas, ni nada correspondiente al sistema de transporte, y a partir de las colaboraciones de los usuarios deberá aprender todo lo relativo a éste. Para ello el sistema debe permitir a los usuarios ingresar todos estos datos, incluyendo mecanismos para su auditoría, de forma de evitar poblar la base de datos del sistema con información imprecisa o falsa.

En segundo lugar, la aplicación en su versión más simple depende de que los usuarios indiquen manualmente las acciones tomadas cada vez que utilizan el sistema de transporte, como por ejemplo, al subirse o descender de un ómnibus. Esto es una barrera importante en el uso de la aplicación, ya que implica acciones conscientes del usuario y con cierta disciplina; debe indicar el momento justo en el que sube a un ómnibus, para no perder parte del recorrido y también debe indicar correctamente el momento en el que desciende del ómnibus, para no brindar a otros usuarios información incorrecta.

Los usuarios pueden desmotivarse en el uso de la aplicación, al no recibir ningún estímulo para tener esta disciplina de indicar las acciones realizadas de forma manual. Esto a su vez provoca que haya poca información en el sistema, por lo que usuarios consultando posiciones de ómnibus no recibirán información suficiente, y tampoco se verán motivados a usar la aplicación. La combinación de estos factores, al igual que en cualquier sistema colaborativo, llevaría a una baja adopción de la

aplicación, siendo este segundo aspecto fundamental para el éxito de la aplicación.

Se plantea entonces la posibilidad de que la aplicación detecte las acciones del usuario de manera automática o semiautomática mediante el uso de heurísticas. Esto significa que la aplicación deberá ser capaz de detectar cuándo el usuario llega a la parada, cuándo se sube a un ómnibus, a qué línea pertenece y cuándo desciende. En el caso ideal de que la aplicación sea capaz de detectar todos los pasos del usuario de forma automática, el usuario deberá únicamente prender la aplicación y consultar la información que requiera. El uso de heurísticas implica un desafío adicional, ya que su ejecución implica un gasto extra de batería para el usuario.

Un ejemplo de heurística para la detección de la llegada de un usuario a la parada podría estar basado en la siguiente idea: la aplicación determina que el usuario está en la parada si este se mantiene quieto y está cerca de una parada conocida por el sistema. En base a eso la aplicación podría suscribirse automáticamente a todas las líneas que pasan por esa parada, de manera que sin requerir intervención del usuario, la aplicación puede informarle por dónde vienen los ómnibus.

Las heurísticas pueden basarse en datos conocidos sobre la red de transporte, como por ejemplo la ubicación de las paradas u ómnibus que pasan por determinada parada. Esto está en contraposición al objetivo de partir de una base de datos completamente vacía. En este sentido son aceptables heurísticas que no funcionen inicialmente pero sí lo hagan cuando exista un mayor volumen de datos conocidos por el sistema.

3.2. Requerimientos específicos

Se presentan a continuación otros requerimientos sobre el sistema a desarrollar.

3.2.1. Aplicación para Android

La aplicación a desarrollar debe implementarse para el sistema operativo Android. La decisión se basó en que el equipo de proyecto tenía dispositivos con Android a su disposición para poder experimentar, el crecimiento en la cuota de mercado que tiene dicho sistema operativo, así como por el hecho de tratarse de software libre. Además, como se comentó en la Sección 2.2, el sistema provee acceso a los sensores presentes en los dispositivos, lo cual es fundamental para el presente proyecto.

3.2.2. Aplicación web de monitoreo

Para poder visualizar el uso de la aplicación en tiempo real, será necesario tener una aplicación web que permita ver los ómnibus actuales representados de manera gráfica en un mapa, así como la información intercambiada entre los dispositivos y el servidor. Esto permite saber sobre el uso de la aplicación, identificando si por ejemplo existen usuarios publicando o consultando sobre las posiciones de los ómnibus.

3.2.3. Optimización de recursos

Las tecnologías de los dispositivos han ido evolucionando, permitiendo obtener resoluciones de alta definición, procesos ejecutándose al mismo tiempo, conexiones más veloces a Internet y la utilización de una gran cantidad de sensores. Esto ha reducido la duración de las baterías drásticamente y presenta un problema muy común en los dispositivos inteligentes actuales. La guía de desarrolladores de Android tiene esto en cuenta, y sugiere que se optimice el uso de la batería [39, 42].

Este problema lleva a tener que utilizar de forma eficiente los recursos del dispositivo, por ejemplo minimizando el tráfico de datos, el procesamiento en el dispositivo y el uso de los sensores.

3.3. Resumen

En este capítulo se presentó el problema planteado para este proyecto. En su versión más simple, el objetivo es la construcción de un sistema colaborativo para dispositivos Android, que aproveche los sensores presentes en estos dispositivos para ubicar en tiempo real los ómnibus de una ciudad determinada. Se presentaron también aspectos agregados al problema que aumentan su complejidad.

En el próximo capítulo, se definirá el alcance de este proyecto, indicando cuáles aspectos de este problema serán abordados y con qué profundidad.

Capítulo 4

Descripción de la solución

En este capítulo se describe la solución diseñada al problema planteado en el capítulo anterior. Se inicia con la definición del alcance del proyecto, y se compara el sistema construido con las aplicaciones vistas en la Sección 2.3. Luego, se indica cómo se modeló la red de transporte. Se continúa con una descripción de alto nivel de la arquitectura implementada, pasando luego al comportamiento de la aplicación. Se explica luego la forma en que se diseñaron e implementaron los diferentes tipos de comunicación en el sistema, y finalmente se presenta una descripción de las heurísticas utilizadas por la aplicación para la detección de eventos relevantes al sistema.

4.1. Alcance del proyecto

En esta sección se define el alcance del proyecto, tomando como base la descripción del problema presentada en el capítulo anterior.

4.1.1. Sistema cooperativo

Se debe construir un sistema cooperativo que permita a los usuarios compartir la posición del ómnibus en el que se encuentran, así como consultar la posición de ómnibus de su interés, tal como se describe en las secciones 3.1.1 y 3.1.2. También debe considerarse que el sistema pueda iniciar su funcionamiento sin ninguna información sobre la red de transporte y comience a aprender sobre ésta a partir de los datos provistos por los usuarios, tal como se describe en la Sección 3.1.3.

4.1.2. Identificación y comunicación eficiente en "grupos de usuarios"

Como se menciona en la Sección 3.1.1, el sistema debe contar con un mecanismo de identificación de "grupos de usuarios", definidos a partir de los intereses de estos. Un posible grupo de usuarios son los interesados en recibir la información conocida sobre los ómnibus de una línea particular. Además, el sistema debe tener mecanismos de comunicación que permitan enviar información a todo el grupo de una forma eficiente y en tiempo real.

4.1.3. Componente social

El sistema debe incluir un componente social que debe permitir el intercambio de mensajes de *chat* dentro de los grupos de usuarios, tal como se menciona en la Sección 3.1.1. El objetivo es propiciar la formación de comunidades de intercambio de información definidas a partir de las líneas de ómnibus existentes en el sistema, o definidas para el área del usuario.

La información intercambiada podría ser relativa al sistema de transporte, por ejemplo en la comunidad de una cierta línea de ómnibus enviar un mensaje indicando un atraso o un desvío, o tener otros tipos de fines sociales, como comentar sobre el programa radial que se escucha en el ómnibus.

4.1.4. Heurísticas basadas en sensores

El sistema debe incluir heurísticas basadas en los sensores presentes en los dispositivos, para detectar eventos relevantes al sistema, como se describe en la Sección 3.1.3. Se deberán desarrollar heurísticas para la detección de alguno de los siguientes eventos:

1. Llegada del usuario a la parada de ómnibus
2. Subida a un ómnibus
3. Descenso de un ómnibus

También se debe diseñar un mecanismo que permita sustituirlas o complementarlas con otras heurísticas que puedan ser desarrolladas en el futuro.

4.1.5. Requerimientos específicos

Los puntos contenidos en la Sección 3.2 deberán ser contemplados en su totalidad en el alcance de este proyecto. Por lo tanto, el sistema debe incluir una aplicación construida para el sistema Android, una aplicación web para monitorear su uso, y debe ser construida considerando la optimización de los recursos disponibles en los dispositivos.

4.2. Comparación con aplicaciones existentes

La aplicación desarrollada como parte de este proyecto tiene una característica que la diferencia de todas las expuestas en el Capítulo 2, que es el hecho de tener como objetivo trabajar con un mecanismo completamente colaborativo, sin apoyo de empresas o gobiernos para obtener los datos. Esto vale tanto para los datos en tiempo real (la posición de la flota en un momento dado) como para la definición de recorridos, paradas y horarios.

Además, es importante destacar que se espera que el sistema provea información en tiempo real relativa al estado de la flota de transporte, a diferencia de algunas de las aplicaciones presentadas, que se limitan a presentar la información estática de los datos teóricos del sistema de transporte.

Existe un componente importante dentro de algunas de las aplicaciones presentadas anteriormente, que es el componente de ruteo. Este componente es importante en una aplicación de este tipo, y sería un interesante agregado a este proyecto, pero no está dentro de su alcance.

La aplicación más similar a la esperada como solución al problema presentado es la desarrollada en la Nanyang Technological University, que parte de las mismas ideas de construcción colaborativa en tiempo real, y uso de sensores de los dispositivos. Las diferencias más grandes encontradas están en los dispositivos usados por ambas aplicaciones, y por el hecho de que el trabajo mencionado se basa en que existe una fase de recolección de datos previa a la puesta en producción, donde se relevan los datos de intensidades de antenas dentro de las rutas de ómnibus habilitadas en el sistema. En este proyecto, se optó por no tener una etapa previa de relevamiento de datos, por lo que inicialmente el sistema partirá de una base datos completamente vacía, y toda la información deberá ser provista por los usuarios.

4.3. Modelo de la red de transporte

Para la implementación del sistema es necesario realizar previamente un modelo del sistema de transporte urbano, identificando las distintas entidades presentes en el mismo, sus relaciones, y las posibles restricciones que podrían aplicar al modelo.

Con este fin, en el inicio del proyecto se realizó un relevamiento para obtener un modelo genérico para redes de transporte basadas en ómnibus. Para hacer esto, se tomó en cuenta la realidad montevideana tomando precauciones para no introducir elementos específicos de esta realidad en el modelo final.

4.3.1. Entidades

Se presentan a continuación las entidades identificadas en el sistema. Se muestran sus nombres en inglés, para corresponderse con la implementación

4.3.1.1. Area (Área)

Se observó en la realidad que existen separaciones en las redes de transporte urbano, generalmente asociadas a una ciudad (o al "área metropolitana", en el caso de Montevideo), o a un área geográfica en general, a las que está asociado el alcance de la red de transporte urbano. Se optó por agregar una definición sencilla de estas áreas, principalmente para poder separar los datos de redes de transporte independientes (por ejemplo, que se puedan separar los datos de Montevideo de los datos de Punta del Este), para simplificar el uso de la aplicación.

4.3.1.2. Bus Line (Línea de ómnibus)

Se modela con esta entidad al grupo de ómnibus que realizan un recorrido específico. Estos grupos están identificados por un número y un destino. En este modelo, líneas con el mismo número no guardan ninguna relación entre sí.

Esto fue modelado de esta forma, porque muchas veces el destino no determina solamente el sentido en el que se mueve el ómnibus, sino que cambia también el camino a seguir (incluso para ómnibus que se mueven en un mismo sentido, por ejemplo, de oeste a este). Por ejemplo, si se toman línea de ómnibus con número 64, en Montevideo, existen 3 posibles destinos: «Pza. Independencia», «Portones», o «Pte. Carrasco». El primero se mueve de este a oeste, mientras que los otros dos se mueven de oeste a este, y tienen recorridos que comparten solo algunas secciones.

En algunos casos, cuando se tienen dos destinos funcionando en un mismo sentido de la línea, uno de los destinos sigue un camino que es un subconjunto del otro (por ejemplo 142 - Pza. Independencia y 142 - Ejido). Sin embargo, en el caso del 64 los dos destinos con sentido hacia el este tienen caminos distintos, con algunos tramos en común.

Al no poder generalizar una condición que determine en qué casos puede inferirse información de otra línea con el mismo número, se decidió que estas se modelen como líneas independientes, y no se explota en ningún momento la coincidencia del número para inferir datos.

4.3.1.3. Bus (Ómnibus)

Esta entidad modela a una instancia de una determinada línea de ómnibus. Es importante destacar, que no se trata de una instancia de ómnibus "físico" (el coche en sí mismo), sino que representa a un coche que en este momento está dando servicio a la línea de ómnibus correspondiente.

No fue modelada dentro de esta entidad ninguna información referente al coche "físico", principalmente porque un coche físico no necesariamente corresponde siempre a una misma línea. Por ejemplo, los ómnibus de la empresa CUTCSA están organizados en grupos identificados con letras, y toda línea de ómnibus pertenece a un grupo. Los coches están también asignados a grupos particulares. Es normal que un coche asignado a un grupo varíe cuál de las líneas del mismo está sirviendo, e incluso en ocasiones extraordinarias puede pasar que coches sirvan a líneas de grupos distintos al propio.

Esta entidad tiene un ciclo de vida en el sistema muy corto, se inicia cuando se sube el primer usuario a esa instancia particular, y termina un tiempo después de que se baja el último usuario.

4.3.1.4. Bus Stop (Parada de ómnibus)

Se representa con esta entidad a las paradas de ómnibus definidas en el sistema. De una parada particular interesa su posición geográfica y la lista de líneas de ómnibus que paran en ella.

4.3.2. Esquema de la base de datos

Partiendo de las entidades descriptas en la sección anterior, se realizó un modelo relacional para la base de datos asociada a la aplicación.

El modelo contempla además la construcción colaborativa de los datos estáticos de la aplicación, como es la definición de las paradas y líneas de ómnibus. Por esta razón, se incorporaron para estas entidades estructuras «paralelas», con el fin de tener por un lado la información que fue suministrada por los usuarios, y por otro la información que es considerada correcta.

Si bien ambas estructuras son usadas por el sistema, tanto para almacenar los datos provistos por los usuarios como para consultar información considerada correcta, no se implementó en este proyecto el algoritmo necesario para inferir a partir de los datos publicados por usuarios la información resumida y correcta. Esto, como se describe en la Sección 8.1, es uno de los posibles trabajos de investigación que pueden realizarse a partir de los resultados de este proyecto.

Se incluyeron además estructuras en el modelo con el fin de mejorar el rendimiento del servidor. Por más detalles sobre el diseño del modelo físico de datos, ver el Anexo F. También con el objetivo de obtener un mejor rendimiento, se optó por no utilizar *frameworks* de persistencia (como JPA o similares) para tener un control fino sobre las operaciones realizadas en la base de datos y evitar procesamientos innecesarios.

4.4. Arquitectura

Se muestra a continuación algunas consideraciones previas que fueron tomadas al definir la arquitectura del sistema, describiendo luego cada uno de sus componentes.

4.4.1. Tipos de arquitectura evaluados

Para el desarrollo de un sistema colaborativo que satisfaga los requerimientos descriptos, se evaluó dos tipos de arquitectura:

1. Red ad-hoc donde los dispositivos se comunican entre sí la información que poseen (un paradigma P2P).

2. Red centralizada en un servidor predeterminado, a donde todos los dispositivos envían su información. El servidor a su vez es el encargado de distribuir la información a los clientes que la requieran.

La opción uno, tiene un alto costo en el desarrollo de mecanismos de comunicación, dado que cada dispositivo tendría que conocer a todos los dispositivos interesados en la información que está compartiendo, y eventualmente la comuni-

cación podría ser todos contra todos. Además, es muy difícil lograr algún tipo de información resumida puesto que todo está distribuido.

La opción dos, centraliza la información en un servidor, de manera que ahí se puede procesar la información para luego enviarla resumida a todos los dispositivos. El servidor deberá mantener actualizada la información sobre qué información espera cada usuario, y en el momento que lleguen nuevos datos deberá distribuirlos de manera eficiente a los usuarios que correspondan.

Se optó por la opción dos, centralizando la información de forma resumida en un servidor central. Si bien el sistema es centralizado desde el punto de vista de las comunicaciones existentes en él, tiene su lógica distribuida entre los componentes de software que lo componen. En este sentido, se tomó la decisión de arquitectura, de que los dispositivos inteligentes conectados al sistema no sean simples interfaces para una lógica centralizada en el servidor, sino que se aprovechan sus capacidades de cómputo para ejecutar parte de la lógica del sistema.

Esto es notorio en el componente de evaluación de heurísticas incluido en el dispositivo, descrito en la Sección 4.7. La lógica de evaluación de heurísticas, si bien no funciona de forma desconectada al servidor, se ejecuta de forma independiente a éste, salvo momentos particulares en los que obtiene información necesaria para continuar la evaluación.

El servidor también posee lógica propia, para procesar los datos recibidos y generar información resumida y actualizada. Se puede catalogar este sistema como un híbrido en la distribución de la lógica, dado que se realizan procesamientos tanto en los dispositivos Android como en el servidor central.

4.4.2. Componentes

En la Figura 4.1 se presenta la arquitectura del sistema, mostrando sus componentes principales. En las siguientes secciones, se brinda una descripción de la función que cumple cada uno de los componentes dentro del sistema.

En las secciones 4.5 y 4.7 se describe en detalle el comportamiento de la aplicación incluida en el dispositivo Android. Por otra parte, en la Sección 4.6 se describen los mecanismos de comunicación presentes en el sistema, y las razones para que existan dos mecanismos para establecer la comunicación entre el dispositivo y el servidor.

Para aclarar los conceptos, en lo que resta del documento, se utilizará el nombre «aplicación» para referirse únicamente a la aplicación Android, «servidor» únicamente para el servidor central y «sistema» para el sistema en su totalidad, abarcando

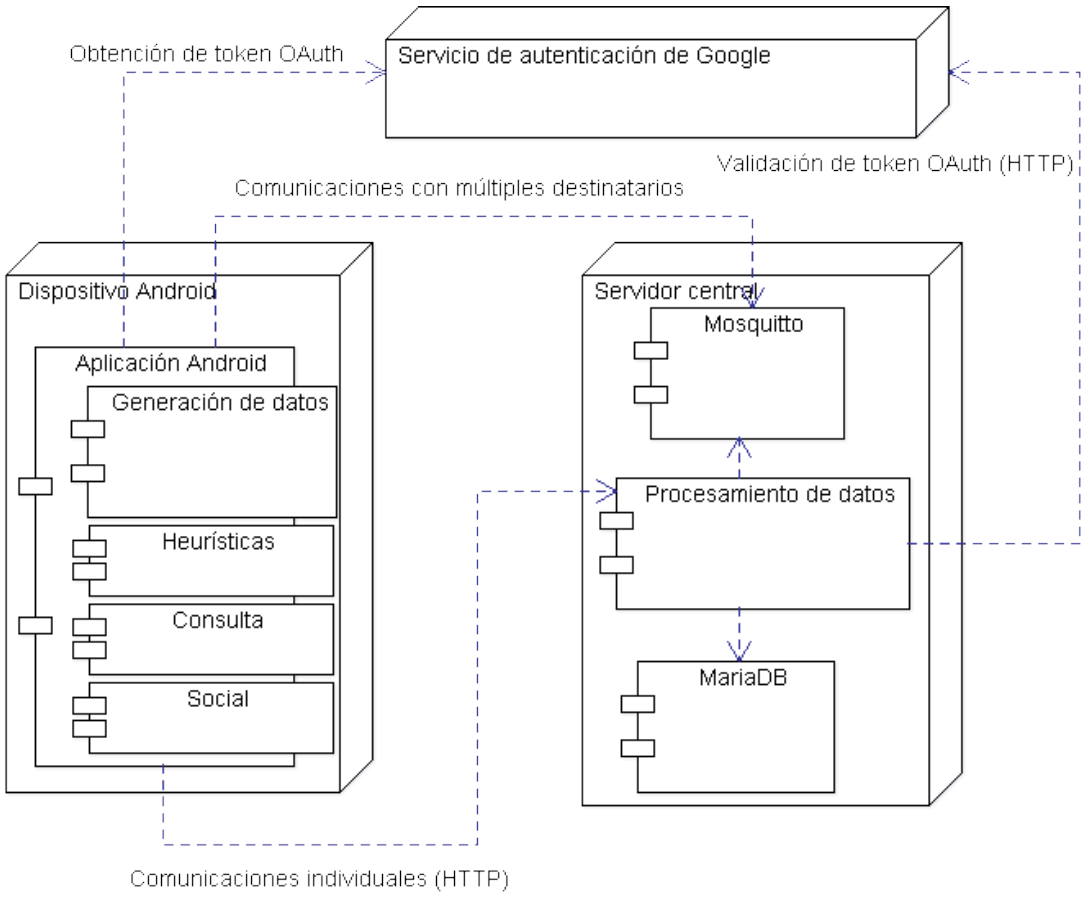


Figura 4.1: Arquitectura del sistema

todos los componentes de la solución.

4.4.2.1. Servicio de Autenticación de Google

En primer lugar, se encuentra el «servicio de autenticación de Google». Este componente es provisto por Google para permitir a desarrolladores independientes autenticar a sus usuarios usando sus identidades de Google (entre otras funcionalidades).

Es necesario incorporar un mecanismo de autenticación en la aplicación para poder hacer un control sobre los usuarios que ingresan, los datos que éstos aportan, y poder ofrecer información contextualizada al usuario que está usando la aplicación. En el inicio del proyecto, se evaluó mecanismos de autenticación, tomando en cuenta la seguridad del mecanismo, su integración con Android y la facilidad para el usuario en la autenticación.

Cuando el usuario comienza a usar un dispositivo Android, el sistema sugiere que se inicie sesión con una cuenta Google. Incluso, para el uso de algunas de las funcionalidades provistas por la plataforma es necesario configurar al menos una cuenta. A partir de ese momento el dispositivo queda ligado a una o varias cuentas de Google.

Al ser Android un sistema desarrollado por Google, la integración entre la aplicación y el «servicio de autenticación de Google» es directa. La seguridad del servicio se basa en el protocolo OAuth, un estándar abierto para la autenticación de aplicaciones de terceros.

Teniendo en cuenta lo presentado, se optó por usar el servicio de Google para la autenticación de los usuarios.

Por más detalles sobre cómo se realizó la integración de la aplicación con el servicio de autenticación de Google, ver el Anexo C.

4.4.2.2. Servidor central

En segundo lugar, se encuentra el servidor central. Este servidor es el responsable de manejar toda la información compartida por los usuarios, y ocuparse que ésta sea entregada a los usuarios que realizan consultas. Para cumplir con estas responsabilidades, el servidor cuenta con dos componentes: «procesamiento de datos» y «Mosquitto». Estos componentes están desplegados en un mismo servidor físico en este proyecto, pero podrían fácilmente instalarse en servidores separados.

Procesamiento de datos Este componente fue desarrollado en Java usando Tomcat como servidor de aplicaciones, y MariaDB como DBMS, es el componente que contiene la lógica ejecutada por el servidor del sistema.

Todos los pedidos de las aplicaciones son resueltos dentro de este componente. Este componente no mantiene estados para los usuarios, por lo que cada pedido de un usuario es procesado de forma independiente.

Existen procesos responsables del envío de la información conocida por el sistema a los grupos de usuarios (a través de sus dispositivos) interesados en ésta. Estos procesos ejecutan periódicamente sin intervención de los usuarios, y sin importar la existencia de usuarios esperando recibir la información.

Además, existe un mecanismo dentro del componente para determinar cuándo dos usuarios se encuentran en un mismo ómnibus y debe manejarse una única instancia en el sistema. Hay casos difíciles de contemplar en este mecanismo, por

ejemplo, las ocasiones donde dos ómnibus de una misma línea realizan el recorrido uno inmediatamente después del otro.

Dada la imprecisión de los sensores de los dispositivos utilizados, y la naturaleza de algunos casos como el anterior, existirán casos donde se presentarán falsos positivos y negativos.

Por último, el servidor expone una interfaz web orientada a dar herramientas a un posible administrador del sistema, para ver rápidamente el estado del servidor, y qué información está circulando por él.

Se muestra en la Figura 4.2 una captura de pantalla de la herramienta web. En este caso se puede visualizar la posición conocida para ese momento de un ómnibus de la línea 494, así como la traza de los puntos obtenidos para este ómnibus hasta ese momento.

En ambos lados se muestran los últimos mensajes enviados entre cliente y servidor, con el objetivo de poder diagnosticar posibles problemas en el sistema.

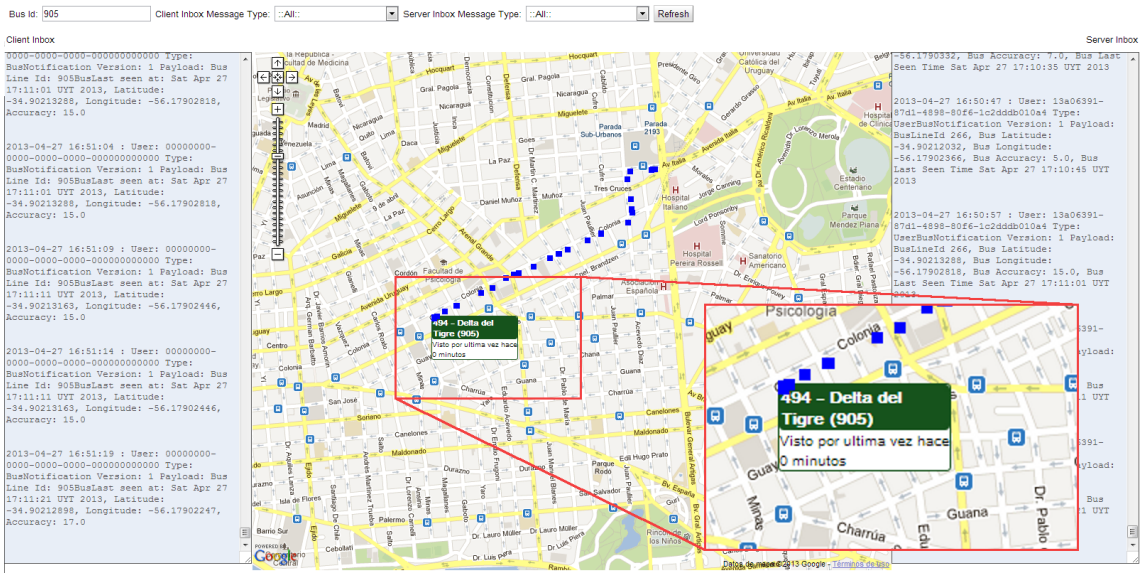


Figura 4.2: Interfaz web de administración

Mosquitto Mosquitto es un servidor implementado por terceros, liberado con licencia BSD. En este proyecto, es usado dentro de este sistema para manejar la entrega de mensajes [25].

Parte de la investigación inicial que se realizó antes del inicio de la implementación del proyecto fue sobre posibles mecanismos de transporte que permitiesen fácilmente establecer grupos determinados por el interés de sus participantes en al-

gún evento particular. Además, se investigaron opciones que no consumiesen grandes cantidades de ancho de banda tanto en el cliente como en el servidor y que cumpliesen otros requisitos. Las opciones manejadas en este sentido, junto con las razones por las que se tomó la decisión final se encuentran en el Anexo A.

4.4.2.3. Aplicación Android

Existen muchas instancias de este componente en el sistema (su número no se encuentra acotado). Cada una de estas instancias se encuentra asociada a un usuario en el sistema, mediante su cuenta de Google.

La aplicación mantiene estados durante cada ejecución. Estos estados están asociados a las distintas situaciones en que se encuentra el usuario y determinan el comportamiento de la aplicación. En la Sección 4.5.1 se detallan el ciclo de vida de la aplicación.

La aplicación tiene cuatro componentes principales: «generación de datos», «consulta», «social» y «evaluación de heurísticas». Estos cuatro componentes se implementaron dentro de un mismo paquete distribuible, con el formato estándar Android para la distribución e instalación de aplicaciones. Más adelante se detalla el comportamiento de cada componente.

En la Figura 4.3 se muestra parte de la interfaz de usuario de la aplicación Android. En la imagen de la izquierda, se puede observar la interfaz principal, donde pueden verse los ómnibus conocidos por el sistema. En este caso el usuario se encontraba publicando la posición de un ómnibus de la línea «494 - Ruta 1 Km 26». En la parte superior de la pantalla se encuentra una barra de acciones que el usuario puede realizar. De izquierda a derecha: como el usuario se encontraba en un ómnibus, tiene la opción de indicar que descendió del mismo, si no se encontrase en un ómnibus, en ese mismo lugar el usuario tiene la opción de indicar que se subió a un ómnibus. Luego tiene la opción para consultar líneas de su interés, accediendo a la pantalla que se muestra en la imagen de la derecha. Por último, la acción de más a la derecha ofrece la opción de acceder al componente social.

Generación de datos Este componente se encuentra activo solo cuando el usuario se encuentra en un ómnibus. Se obtiene la ubicación conocida de los sensores presentes en el teléfono, y se publica este dato al servidor junto con la línea de ómnibus en que se encuentra el usuario. Esta es la información que luego el sistema proveerá a otros usuarios. El usuario puede también indicar cuando lo desea que se encuentra en una parada de ómnibus, colaborando así con información al sistema. Para obtener

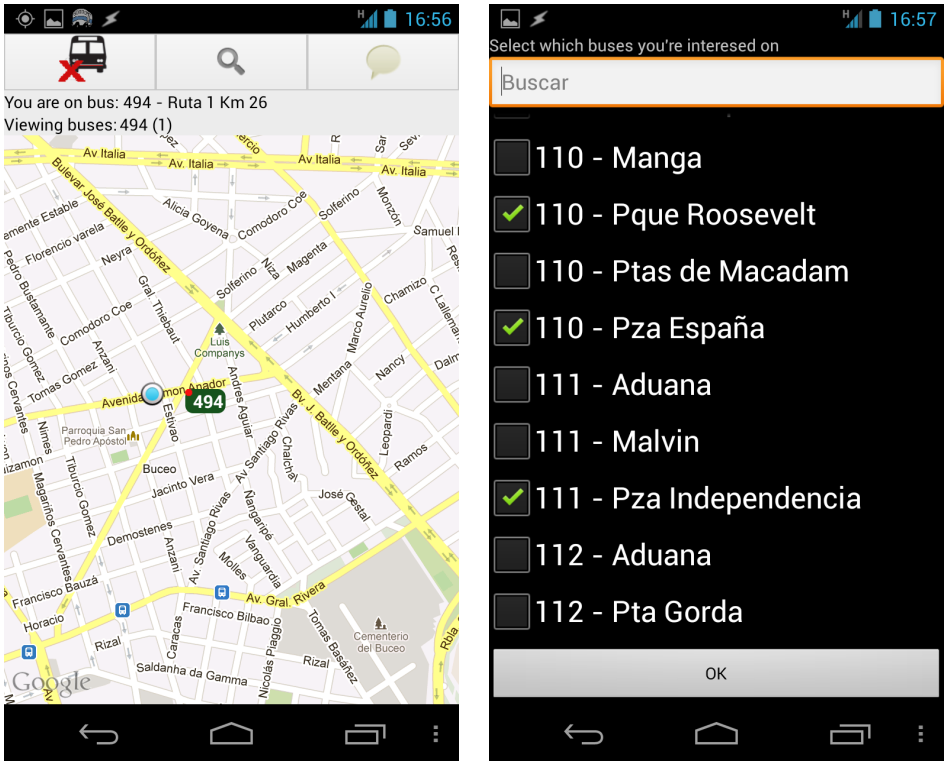


Figura 4.3: Interfaz de usuario aplicación Android

la ubicación, el dispositivo lleva una lógica en la que da preferencia a las posiciones obtenidas a partir del sensor de GPS, teniendo las posiciones obtenidas a partir de antenas de celulares como alternativa.

Consulta Este componente es el que provee al usuario la posibilidad de consultar datos del sistema. Se provee es una interfaz donde el usuario puede seleccionar las líneas de ómnibus en las que está interesado, y una pantalla donde se muestran gráficamente en un mapa, en tiempo real, las posiciones de ómnibus conocidos para estas líneas.

Social Este componente permite la comunicación entre los usuarios de la aplicación, permitiéndoles intercambiar mensajes de *chat*. No es la intención que se puedan realizar intercambios entre dos usuarios individualmente, sino que se pretende que la comunicación se realice dentro de grupos creados automáticamente por la aplicación. Estos grupos pueden ser de dos tipos: asociados al área donde se encuentra el usuario, o asociados a las líneas de ómnibus pertenecientes a esta área.

Este componente puede tener dos usos: el intercambio de información relevante

a los usuarios de transporte público (por ejemplo, advirtiendo de un cierto desvío, o demora, en una línea particular), o intercambios relativos a otros temas de interés de los usuarios, como una interacción social más general dentro de los usuarios de una ciudad o de una línea particular. Se cree que éste es un estímulo para el uso de la aplicación.

Evaluación de heurísticas Este componente permanece activo permanentemente en el dispositivo, y tiene como objetivo detectar eventos basado en la información provista por los sensores del teléfono. Para esto, la aplicación define algunos procesos que se encuentran en espera la gran mayoría del tiempo, haciendo un consumo mínimo de batería, hasta que se detecta un posible evento, y pasan a estar activos para verificar que ese evento haya tenido lugar. En caso de verificarse que el evento ocurrió, se presentarán al usuario las opciones correspondientes basadas en este evento.

Las heurísticas implementadas dentro de este componente podrían mejorarse en proyectos futuros, como se menciona en la Sección 8.6.

4.5. Comportamiento de la aplicación

En esta sección se detalla el comportamiento de la aplicación. Como se mencionó en la Sección 4.4.2, la palabra «aplicación» en este contexto se refiere únicamente a la aplicación Android desarrollada como parte de la solución.

Se dividió la descripción en dos partes, la definición de una máquina de estados ilustrando el comportamiento, y la descripción de los eventos manejados dentro de la aplicación.

4.5.1. Ciclo de vida de la aplicación

El modelo de la Figura 4.4 describe el ciclo de vida de la aplicación, con el fin de brindar una visión global del comportamiento de la misma.

Se identificaron los estados relevantes para la aplicación, basándose en la interacción del usuario con el transporte público.

Los estados identificados son los siguientes:

- INITIAL: Se permanece en este estado mientras la aplicación realiza tareas de inicialización y automáticamente lo abandona al terminarlas. Ocurre cuando el usuario acaba de ingresar a la aplicación.

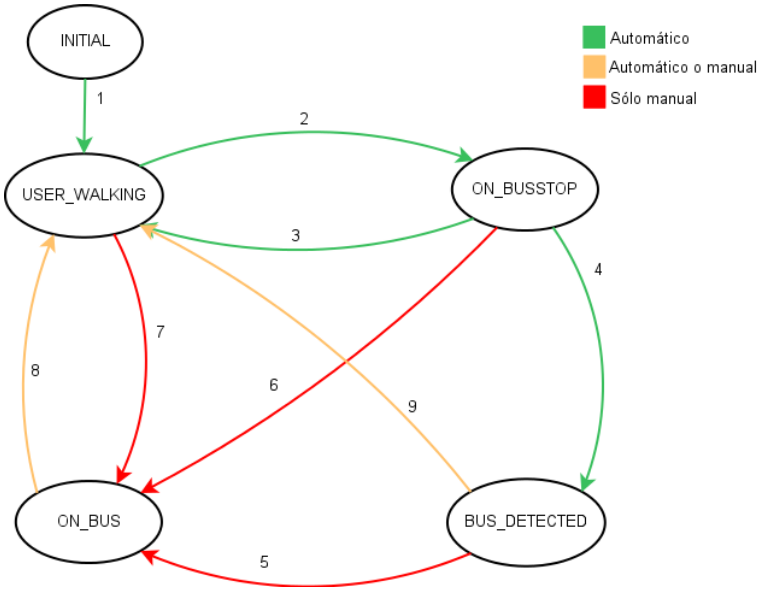


Figura 4.4: Diagrama de estados de la aplicación

- USER_WALKING: Modela que el usuario no se encuentra utilizando el transporte público.

- ON_BUSSTOP: Modela que el usuario se encuentra esperando un ómnibus en una parada.

- BUS_DETECTED: Modela que el usuario se encuentra en un ómnibus, pero no se tiene información sobre la línea a la que pertenece.

- ON_BUS: Modela que el usuario se encuentra en un ómnibus, y sí se tiene información sobre la línea a la que pertenece.

La Figura 4.4 muestra las interacciones entre estos estados, indicando para cada transición si ésta se realiza de forma automática (a través de una heurística), de forma manual por el usuario, o si ambas opciones son posibles.

Para simplificar la lectura, se omitieron en el diagrama las condiciones que deben cumplirse para realizar cada transición, y en su lugar se colocaron números. En el Cuadro 4.1 se presentan las condiciones que deben cumplirse para que se ejecute cada transición, según su número. En las transiciones que poseen más de una entrada, cualquiera de las condiciones puede causar la transición de forma independiente.

N°	Condición(es)
1	Inicio de aplicación completado
2	Detección de que el dispositivo se encuentra quieto y en la posición de una parada conocida. Ver Sección 4.7.1.
3	Detección de que el usuario abandonó la parada caminando. Ver Sección 4.7.1.
4	Detección de que el usuario abandonó la parada sobre un ómnibus. Ver Sección 4.7.1.
5	Determinación por parte del usuario de cuál fue el ómnibus al que se subió, con ayuda de la aplicación que provee candidatos posibles según la parada. Ver Sección 4.7.1.
6 y 7	Determinación de forma manual (mediante un botón específico) por parte del usuario indicando que se encuentra dentro de un ómnibus.
8	Determinación de forma manual (mediante un botón específico) por parte del usuario indicando que ya no se encuentra dentro de un ómnibus.
8	Heurística de detección de descenso de ómnibus. Ver Sección 4.7.2.
8	Alerta de proximidad de parada. Ver Sección 4.7.3.
9	La aplicación estuvo demasiado tiempo en el estado *BUS_DETECTED*, o el usuario indicó manualmente que no se encuentra en un ómnibus.

Cuadro 4.1: Condiciones de transición de estados

4.5.2. Eventos

Dos importantes objetivos del diseño del sistema operativo Android fueron maximizar la reutilización de componentes y permitir cambiar un componente por otro de una forma sencilla [41].

Para lograr este objetivo, Android utiliza un mecanismo que permite realizar llamadas entre diferentes aplicaciones en tiempo de ejecución, usando *intents*. Un *intent* almacena información sobre qué acción se desea realizar y opcionalmente con cuáles parámetros.

Cada aplicación indica cuáles *intents* puede resolver, registrando en el sistema operativo componentes para la recepción de dichos *intents*. Estos componentes son denominados *broadcast receivers* en Android. El sistema operativo tiene la responsabilidad de llevar el estado de cuáles *broadcast receivers* están suscritos a una determinada acción, y de notificar a estos cuando corresponde.

Cuando una aplicación envía un *intent*, el sistema operativo resuelve cuáles son los *broadcast receivers* que pueden actuar sobre él, y los invoca reenviándoles ese

intent. De esta manera la aplicación que envía el *intent* se desentiende de quién lo recibirá y qué hará con la información.

Los *intents* en Android son usados principalmente con los siguientes objetivos:

1. Envío de avisos sobre eventos del sistema mediante mensajes *broadcast*: por ejemplo, el sistema puede enviar avisos de batería baja, y las aplicaciones que reciban el *intent* podrían adaptar su funcionamiento para ahorrar batería.

2. Invocación de actividades: los cambios de actividad (de pantalla) son siempre manejados a través de *intents*. Por ejemplo, esto permite que una aplicación envíe un *intent* indicando que desea invocar una pantalla de envío de emails, y el sistema operativo resuelve cuál es la aplicación que debería resolver este pedido.

En este proyecto se definió una acción por cada evento identificado, siguiendo la idea del primer punto. Los eventos definidos son los mostrados en el Cuadro 4.2. Cada evento se corresponde con una transición de la máquina de estados presentada en la Figura 4.4.

Desde cualquier componente de la aplicación pueden enviarse mensajes *broadcast* indicando que ocurrió un cierto evento, mediante un *intent* con la acción correspondiente. A su vez, cada componente de la aplicación se suscribe a los eventos que son de su interés, registrando un *broadcast receiver* en el sistema operativo.

Los distintos componentes de la aplicación utilizan este mecanismo para informarse sobre todos los cambios de estado dentro de la aplicación, implementando así la máquina de estados presentada en la sección anterior.

Nombre Evento	Condición de disparo
STARTUP_COMPLETED	Salida del estado *INIT*
USER_WALKING	Entrada al estado *USER_WALKING*
BUSSTOP_DETECTED	Entrada al estado *BUSSTOP_DETECTED*
GETONBUS_DETECTED	Entrada al estado *BUS_DETECTED*
ON_BUS	Entrada al estado *ON_BUS*
APPLICATION_STATE_CHANGED	Luego de cualquier cambio de estado

Cuadro 4.2: Eventos manejados por la aplicación

El uso de este mecanismo tiene como ventaja la facilidad de extender la aplicación, ya que por ejemplo, si se define una nueva heurística, ésta puede declararse como *broadcast receiver* para un evento particular, y en el caso de determinar algún evento de los presentados se envía un *intent* con la acción correspondiente.

Por ejemplo, podría agregarse una nueva heurística que iniciase cuando se realiza la transición al estado *BUS_DETECTED* con el cometido de determinar cuál es el ómnibus al que subió el usuario. En caso de determinar cuál es este ómnibus, enviaría un *intent* con la acción correspondiente al evento *ON_BUS*. La incorporación de esta nueva heurística tendría un bajo impacto sobre el resto de la aplicación.

4.6. Comunicación cliente-servidor

En la presente sección se presentan los mecanismos utilizados por la aplicación para comunicarse con el servidor. Se parte de un análisis a grandes rasgos de los tipos de comunicaciones existentes en el sistema, y el enfoque con que se resolvió cada uno de estos tipos de comunicaciones, para pasar luego a describir el protocolo diseñado para las comunicaciones.

4.6.1. Tipos de comunicaciones existentes en el sistema

En la comunicación entre el cliente y el servidor existen 2 tipos de mensajes bien definidos:

1. Los mensajes destinados a «grupos de usuarios». Por ejemplo, son los que va a usar el servidor para publicar posiciones de las líneas de ómnibus al grupo de usuarios que esperan recibir esta información.

2. Los mensajes destinados a destinatarios «individuales». Por ejemplo, un caso es el mensaje donde, dada la posición de un usuario determinado, el servidor debe enviar la lista de paradas cercanas. Usando estos mensajes frecuentemente se intercambia información sensible con el servidor, por ejemplo para realizar la autenticación de los usuarios en el sistema. Por esta razón, interesa que la privacidad de estas comunicaciones esté garantizada.

4.6.2. Solución implementada según el tipo de comunicación

Se diseñaron mecanismos independientes para los mensajes enviados dentro de la aplicación, para cada uno de los tipos descriptos en la sección anterior.

4.6.2.1. Mensajes dirigidos a grupos de usuarios

Para la implementación del primer tipo de mensajes, fueron evaluadas varias opciones de herramientas, implementadas en capa de aplicación, que simplificaran el envío de mensajes en forma similar a la implementación de *multicast* a nivel de capa de red. Las opciones están resumidas en el Anexo A, junto con las razones por las que fue elegido el protocolo MQTT. En este protocolo la comunicación se da a través de *topics* en los que se puede publicar información, y al que pueden subscribirse clientes para recibir las publicaciones correspondientes. La definición de los *topics* se hace en tiempo de ejecución, lo que permite una gran flexibilidad a la hora de definir cuáles serán los *topics* a utilizar.

Un actor fundamental dentro del protocolo MQTT es el llamado *message broker,* que es encargado de centralizar la recepción y enrutamiento de los mensajes a sus respectivos destinatarios. Para este proyecto se eligió el proyecto Paho[1] como implementación de MQTT y Mosquitto como *message broker*, dada su disponibilidad como software libre y de código abierto, y su facilidad de instalación.

Es importante destacar que el protocolo MQTT está desarrollado sobre el protocolo TCP, por lo que no se trata realmente de una red *multicast*, sino que abstrae el concepto permitiendo a la aplicación manejar mensajes como si fueran a distribuirse sobre una red con estas características. Lo que ocurre cuando se utiliza este protocolo, es que cada cliente conectado al *message broker* comparte una (como mínimo) conexión TCP con éste, y el *message broker* se encarga de distribuir los paquetes entrantes por las conexiones adecuadas.

Los mensajes dirigidos a grupos de usuarios representan la mayoría de los mensajes del sistema, por lo que se trabajó en buena parte del proyecto en la definición de un protocolo que manejase de forma óptima la información a enviar sobre MQTT, con el fin de lograr una comunicación eficiente. El diseño del protocolo tuvo dos componentes: por un lado la generación de *topics* adecuados para definir claramente los grupos a donde debía llegar la información, y por otro la semántica y codificación de los mensajes en sí.

[1] http://www.eclipse.org/paho/

4.6.2.2. Mensajes punto a punto

El protocolo MQTT maneja de forma correcta mensajes orientados a grupos de usuarios. Sin embargo, también es necesario contar con comunicaciones punto a punto, para los cuales este protocolo no fue diseñado. En esta sección, se consideran conexiones punto a punto a conexiones iniciadas desde alguno de los clientes con destino el servidor, donde se envía un cierto pedido de información, y se espera una respuesta concreta a ese pedido.

Inicialmente se optó por simular todas las conexiones punto a punto diseñando comunicaciones dentro de MQTT que tuvieran un comportamiento similar a éstas. Por limitaciones de Mosquitto en la definición de usuarios, roles, y permisos (deben ser definidos estáticamente antes de iniciar el servidor), no era sencillo encontrar una forma de garantizar la privacidad de los mensajes usando esta herramienta.

Para los mensajes que esperaban respuesta del servidor, pero donde la respuesta era válida para más de un usuario, y no se transmitía información sensible, se optó por usar MQTT, de forma que la respuesta llegue simultáneamente a todos los interesados, en caso de que hubiese alguno más aparte del usuario que inició el pedido.

Para mensajes dentro del segundo caso (información relevante para un único usuario, posiblemente sensible), se modificó la estrategia para garantizar su privacidad. Este tipo de comunicaciones es manejado a través de pedidos HTTP al servidor; eventualmente podrían ser HTTPS para mejorar la seguridad, en tal caso se deberá asumir el costo computacional adicional.

4.6.3. Protocolo de mensajes destinados a grupos de usuarios

En la presente sección se presenta el diseño del protocolo de mensajes destinados a grupos de usuarios. Se comienza explicando cómo se identifican los usuarios, continuando por la semántica de los mensajes intercambiados entre los clientes y el servidor. Luego se presenta la forma en que se diseñó la estructura de los *topics* para usar eficientemente las características de Mosquitto, finalizando con una descripción sobre cómo evolucionó la sintaxis de los mensajes.

4.6.3.1. Identificación de los usuarios

Para la autenticación de los usuarios en el sistema se usa el mecanismo provisto por el sistema operativo para obtener acceso a la cuenta de Google del usuario. Detalles de este mecanismo se encuentran en el Anexo C.

En la gran mayoría de los mensajes enviados usando MQTT, es preciso incluir alguna referencia al usuario que los genera. Por ejemplo es de interés saber, en el momento que se publica la posición de un ómnibus particular, cuál es el usuario que está realizando esta acción. Esto es importante por dos razones:

1. Identificación del usuario con el fin de proveer opciones personalizadas en un futuro.

2. Protección del sistema de intrusos, evitando accesos anónimos, y eventualmente penalizando a usuarios que sistemáticamente provean información falsa.

Para lograr este objetivo inicialmente se podría haber usado la dirección de correo del usuario, o alguna otra identificación del mismo existente en el sistema (por ejemplo, un identificador autonumerado en la base de datos). No se tomó esta opción, ya que como la seguridad provista por Mosquitto es laxa, un atacante con acceso a Mosquitto podría ver los identificadores válidos y pasar a simular la identidad de esos usuarios. Incluso, un atacante con acceso a Mosquitto podría identificar el identificador asociado a un usuario particular mirando las trazas en el sistema, analizando por ejemplo los tiempos en que el usuario sube y baja de los ómnibus. Luego de tener este identificador, podría fácilmente rastrear la posición del usuario, lo que implicaría una violación grave en la privacidad del mismo.

Por esta razón, se genera en cada autenticación de usuario un identificador temporal (que en adelante llamamos *ID temporal*), que será el usado en las comunicaciones dentro de Mosquitto para dar identidad a los usuarios. Estos *ID temporales* son generados usando la especificación para identificadores UUID provista por la IETF, usando la implementación existente en java, dada por la clase UUID.

Se eligió este mecanismo principalmente por dos propiedades de los números generados con esta técnica:

1. El espacio de posibles identificadores es muy grande (lo que dificulta a un atacante “adivinar” valores válidos)

2. La probabilidad de obtener el mismo valor desde el generador de identificadores es muy baja (si se accede a la aplicación con un *ID temporal* viejo, es poco

probable que exista otro usuario que se encuentre ahora usando ese mismo identificador).

3. Es más complejo el escenario que permitía a un atacante rastrear usuarios en la aplicación. Ahora el atacante podrá solo obtener el *ID temporal*, que por su naturaleza temporal solo permitirá rastrear el viaje en el que es usado. Si bien sigue siendo posible el rastreo en una escala menor, se estima que este riesgo es menos significativo, ya que para lograr identificar al usuario el atacante debe saber dónde se encuentra el usuario (y en qué ómnibus), y luego analizar la traza de conexiones para encontrar los candidatos a ser el usuario. En caso de encontrar un único usuario, podría teóricamente "seguirlo" hasta su destino, pero perdería el rastro en ese momento. Considerando que el atacante tiene que conocer inicialmente el ómnibus en el que se encuentra el usuario, este escenario no sería muy riesgoso.

Concluyendo, el resultado del proceso de autenticación es la generación de un nuevo *ID temporal* para el usuario, que usará durante esa sesión para comunicarse con el servidor. El servidor, por su parte, mantiene en la base de datos la asignación de *ID temporales* a usuarios, para poder resolver el usuario a partir de un *ID temporal* dado cuando sea necesario. Estos identificadores temporales serán enviados en todos los mensajes que los dispositivos envíen al servidor.

4.6.3.2. Semántica de los mensajes

En esta sección se presenta una descripción del protocolo desde un punto de vista semántico, partiendo de una descripción de alto nivel de las interacciones presentes en el sistema e infiriendo desde éstas los tipos de mensajes a utilizar dentro del protocolo.

Interacciones entre dispositivo y servidor Se presentan a continuación descripciones de las distintas interacciones entre clientes y servidor durante el uso de la aplicación. Estas interacciones fueron divididas en 4 partes:

1. Inicio de la Aplicación.
2. Publicación de posiciones y paradas de ómnibus.
3. Subscripción a línea de ómnibus.
4. Componente social

Al final de cada una de las secciones, se modelan las interacciones mediante máquinas de estados donde las transiciones se encuentran acompañadas por una etiqueta, con el siguiente formato: «condición / acción». La condición descripta es la necesaria para que se efectúe la transición, y las acciones descriptas son las ejecutadas en el momento de efectuar la transición. Las condiciones denotadas con un asterisco están asociadas a acciones del usuario.

Todas las comunicaciones llevadas a cabo en las interacciones que se describirán, usan el protocolo MQTT y el Mosquitto como *Message Broker* (a excepción del mensaje de autenticación que es transportado usando HTTP).

Inicio de la Aplicación Esta interacción se da cada vez que el usuario inicia la aplicación.

Cuando el usuario inicia la aplicación por primera vez, ésta envía un mensaje para indicar que desea recibir la lista de áreas disponibles para que el usuario pueda elegir el área a la que pertenece. Luego de que el servidor responde con la lista de áreas conocidas, el usuario debe seleccionar el área y la cuenta de Google que desea usar (por más información sobre la interacción con las cuentas de Google, ver Anexo C). Se realiza en este momento la autenticación con el servidor. Como se describió anteriormente, la comunicación en este caso, por tratarse de una comunicación punto a punto, se realiza mediante una conexión HTTP entre el dispositivo y el servidor. Luego de la validación, asumiendo que la misma fue correcta, la aplicación envía al servidor 2 mensajes, uno de ellos indicando que desea conocer la lista de líneas de ómnibus, y otro para conocer los grupos sociales disponibles para el área del usuario. Luego de que el servidor procesa los mensajes, responde con las listas solicitadas.

Es importante recordar que como las respuestas del servidor son enviadas mediante Mosquitto, si existen varios usuarios distintos que estén esperando el mismo mensaje, todos recibirán estas respuestas.

En inicios de la aplicación posteriores, como la aplicación ya está configurada, no ingresa al proceso de selección de áreas y cuenta, sino que se pasa directamente a intentar autenticar al usuario. Si la autenticación es correcta, se vuelven a solicitar las listas de líneas de ómnibus y grupos sociales, para actualizar esta información en el dispositivo.

A partir de la interacción descripta, se identifican los siguientes tipos de mensaje:

- **AreaListSolicitation**: Solicitud por parte del cliente de la lista de áreas.

- **AreaListNotification**: Notificación del servidor con la lista de áreas conocidas en el sistema.
- **BusListSolicitation**: Solicitud por parte del cliente de la lista de líneas de ómnibus.
- **BusListNotification**: Notificación del servidor con la lista de las líneas de ómnibus.
- **SocialChannelSolicitation**: Solicitud por parte del cliente de la lista con los grupos sociales disponibles.
- **SocialChannelNotification**: Notificación del servidor con la lista de grupos sociales disponibles.

Se muestra en la Figura 4.5 el diagrama que representa la interacción entre el dispositivo y el servidor con el proceso de autenticación, haciendo uso de los tipos de mensaje identificados.

Publicación de posiciones de ómnibus y paradas de ómnibus El escenario de publicación de posiciones de ómnibus se da cuando el usuario indica que se subió a un determinado ómnibus, mientras que la publicación de paradas de ómnibus puede darse en cualquier momento.

Cuando el usuario indica que se subió a un ómnibus, la aplicación pasa a un estado en el cual empieza a enviar periódicamente mensajes al servidor para notificar las posiciones del usuario, y por tanto las del ómnibus en el que éste se encuentra. Cada determinado tiempo, la aplicación envía al servidor un mensaje con la información sobre la posición del usuario y la línea de ómnibus a la que pertenece el ómnibus en el que se encuentra el usuario.

Luego de que el usuario indica que se bajó del ómnibus, el envío de estos mensajes se detiene.

A partir de la interacción descripta, se identifican los siguientes tipos de mensaje:

- **UserBusNotification**: Notificación enviada por la aplicación con la posición actual del usuario y la línea de ómnibus en el que éste se encuentra
- **UserBusStopNotification**: Notificación enviada por la aplicación con la posición actual del usuario indicando que en esta posición existe una parada.

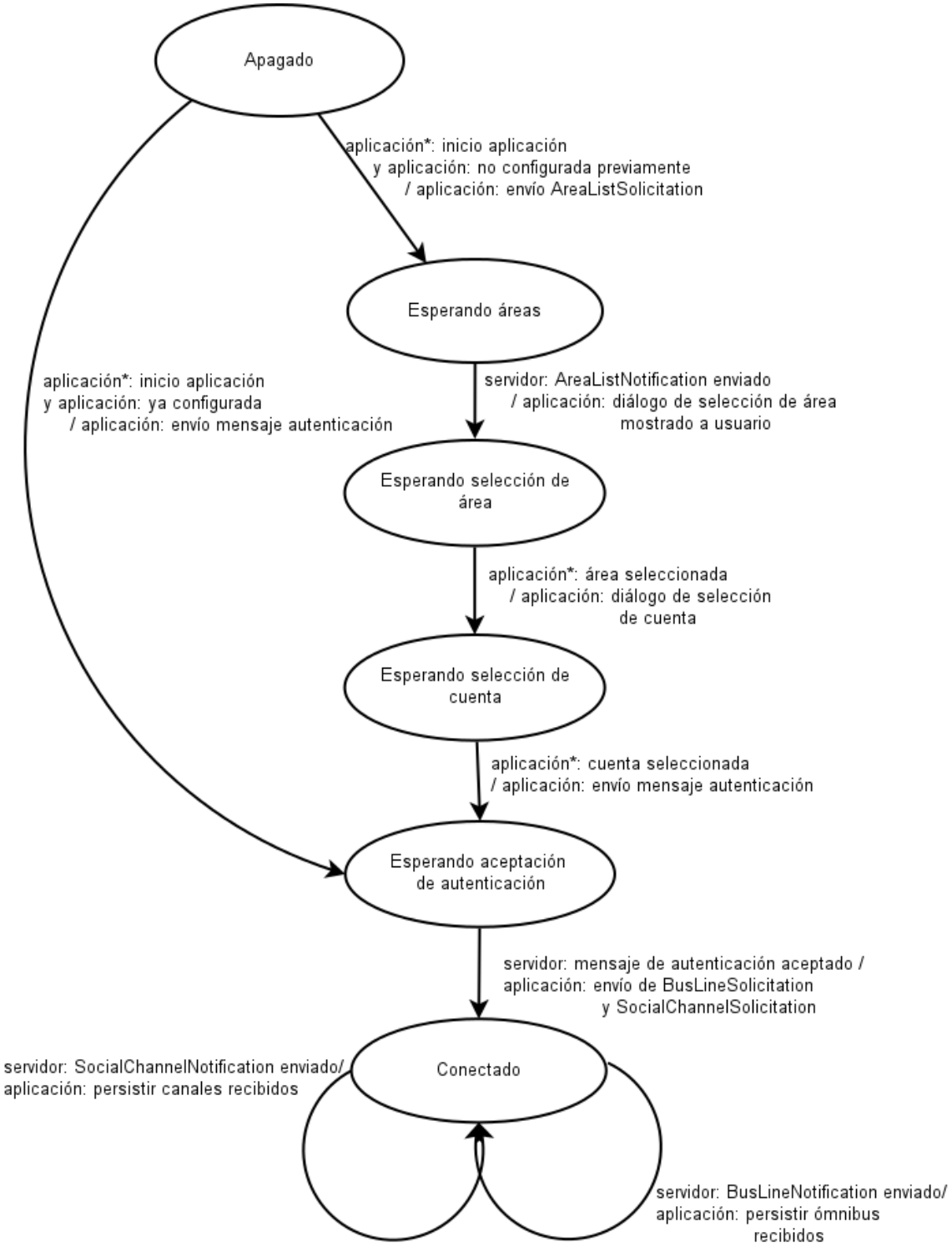


Figura 4.5: Inicio de la aplicación

Se muestra en la Figura 4.6 el diagrama que representa la interacción entre el dispositivo y el servidor en la publicación de la posición del ómnibus del usuario, así como

de posibles paradas de ómnibus, haciendo uso de los tipos de mensaje identificados.

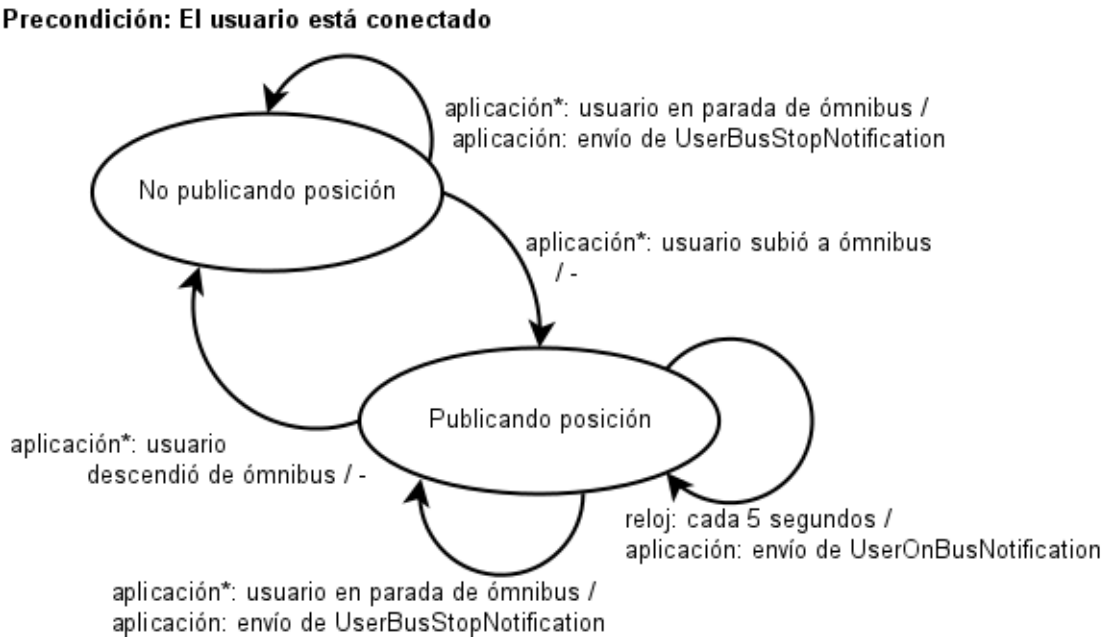


Figura 4.6: Publicación de posiciones de ómnibus

Subscripción a línea de ómnibus Esta interacción se da cuando el usuario desea ver por dónde vienen los ómnibus de determinada línea.

Si el usuario desea conocer la posición de ómnibus para una determinada línea de ómnibus, la aplicación se subscribe a dicha línea para recibir las posiciones que el servidor publique. El servidor enviará periódicamente mensajes, que serán recibidos por todos aquellos dispositivos que se encuentren suscriptos a la línea en cuestión, con todas las posiciones conocidas de los ómnibus pertenecientes a dicha línea.

Cuando el usuario indica que ya no desea ver la posición de los ómnibus de la línea previamente seleccionada, la aplicación cancela la suscripción a la misma, y por lo tanto deja de recibir los mensajes con las posiciones conocidas por el sistema.

La situación representada por el diagrama se puede dar de forma concurrente para varias líneas de ómnibus, ya que el usuario puede querer conocer la posición de los ómnibus para varias líneas, teniendo cada una su propio estado.

A partir de la interacción descripta, se identifica el siguiente tipo de mensaje:

- **BusNotification**: Notificación del servidor con la posición actual de un ómnibus.

Se muestra en la Figura 4.7 el diagrama que representa la interacción entre el dispositivo y el servidor cuando el usuario desea conocer la posición de una (o varias) línea(s) de ómnibus, haciendo uso del tipo de mensaje identificado.

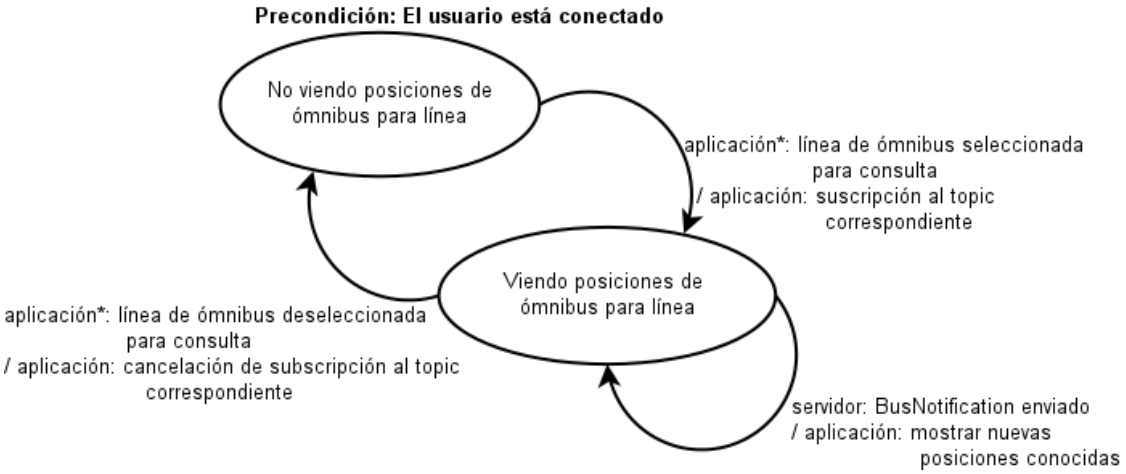


Figura 4.7: Suscripción a líneas de ómnibus

Componente social Esta interacción se da cuando el usuario quiere hacer uso del componente social.

Cuando el usuario indica que quiere entrar en un grupo social dado, la aplicación se suscribe al *topic* correspondiente, para empezar a recibir los mensajes enviados al grupo. Una vez que el usuario está suscripto, puede enviar y recibir mensajes dentro del grupo. Los mensajes enviados por el cliente tienen como destino el servidor, quien se encarga de validar el mensaje, y reenviarlo a todos los clientes que se encuentran suscriptos al grupo.

Luego de que el usuario sale del grupo, la aplicación cancela la suscripción al *topic* correspondiente y deja de recibir información del mismo.

La situación representada por el diagrama se puede dar de forma concurrente para varios grupos sociales, ya que el usuario puede estar en varios, teniendo cada uno su propio estado.

A partir de la interacción descripta, se identifican los siguientes tipos de mensaje:

- **SocialUserMessage:** Mensaje enviado por un usuario a un grupo.
- **SocialServerMessage**: Mensaje reenviado por el servidor a los usuarios de un grupo.

Se muestra en la Figura 4.8 el diagrama que representa la interacción entre el dispositivo y el servidor en el componente social de la aplicación, haciendo uso de los tipos de mensaje identificados.

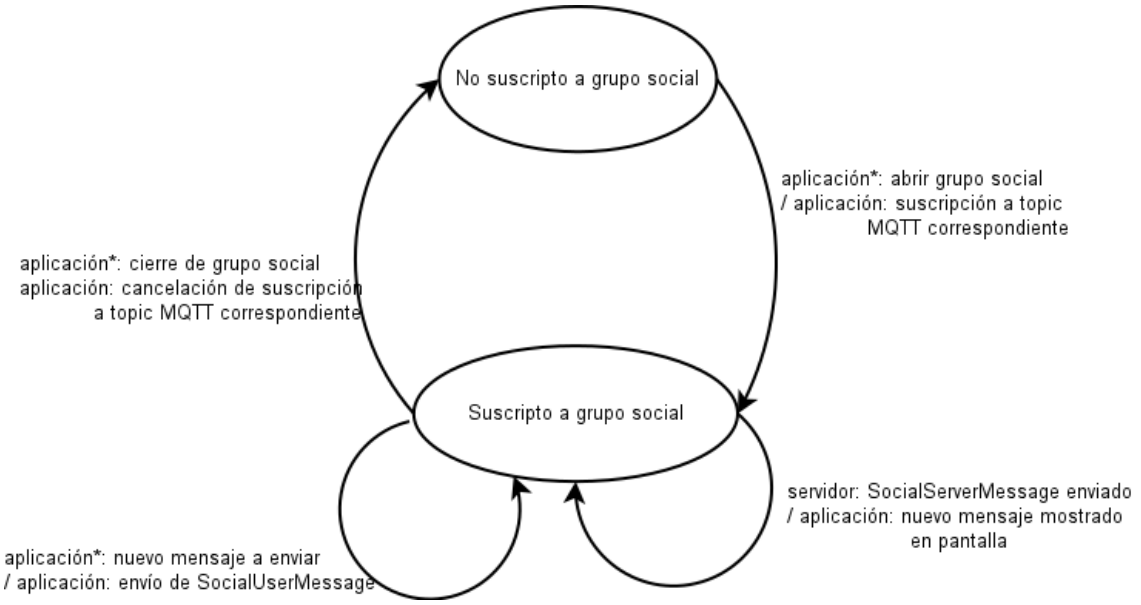


Figura 4.8: Grupos sociales

Tipos de Mensaje Luego de la presentación de las posibles interacciones entre el dispositivo y el servidor, e identificados los tipos de mensajes para llevar la comunicación a cabo, se muestran a continuación un resumen de todos los tipos de mensaje que serán enviados entre los dispositivos y el servidor mediante Mosquitto:

- **AreaListSolicitation**: Solicitud por parte del cliente de la lista de áreas.
- **AreaListNotification**: Notificación del servidor con la lista de áreas conocidas en el sistema.
- **BusListSolicitation**: Solicitud por parte del cliente de la lista de líneas de ómnibus.
- **BusListNotification**: Notificación del servidor con la lista de las líneas de ómnibus.
- **SocialChannelSolicitation**: Solicitud por parte del cliente de la lista con los grupos sociales disponibles.

- **SocialChannelNotification**: Notificación del servidor con la lista de grupos sociales disponibles.
- **UserBusNotification**: Notificación por parte del cliente sobre la posición actual del usuario y el ómnibus en el que este se encuentra.
- **BusNotification**: Notificación del servidor con la posición actual de un ómnibus.
- **SocialUserMessage:** Mensaje enviado por un usuario a un grupo social.
- **SocialServerMessage**: Mensaje reenviado por el servidor a los usuarios de un grupo social.
- **UserBusStopNotification**: Notificación por parte del cliente sobre la posición actual del usuario y una determinada parada de ómnibus en la que este se encuentra.

4.6.3.3. Estructura de los *topics*

Para poder enviar y recibir los mensajes que se describieron en la sección anterior de una manera eficiente y usando las facilidades que provee el MQTT, se realizó un análisis sobre la mejor forma de estructurar los *topics*.

Se crearon 2 estructuras independientes, una para los mensajes enviados desde el dispositivo hacia el servidor, y otra para los mensajes del servidor al dispositivo. De esta manera resulta simple distinguir mensajes que tienen como destino el servidor de mensajes que tienen como destino el cliente. Se presentan en los Cuadros 4.3 y 4.4 los *topics* definidos, junto con los tipos de mensajes que circulan por estos *topics*. Más adelante, se explica porqué se eligieron estos *topics* particulares.

Topic	**Tipo de Mensaje**
/client/area/	AreaListNotification
/client/area/<area_id>	BusListNotification
	SocialChannelListNotification
/client/area/<area_id>/line/<bus_line_id>	BusNotification
/artbus/client/social/area/<area_id>	SocialServerMessage

Cuadro 4.3: *Topics* del cliente: *Topics* a los cuales los dispositivos (clientes) se van a suscribir, y que utiliza el servidor para publicar

Topic	**Tipo de Mensaje**
/server/area	AreaListSolicitation
/server/area/<area_id>	BusListSolicitation
	SocialChannelListSolicitation
/server/area/<area_id>/line/<bus_line_id>	UserBusNotification
/server/area/<area_id>/busstop/	UserBusStopNotification
/artbus/server/social/area/<area_id>	SocialClientMessage

Cuadro 4.4: *Topics* del servidor: *Topics* a los cuales el servidor se va a suscribir, y que utilizan los dispositivos (clientes) para publicar

Estos *topics* fueron definidos de esta forma para aprovechar algunas posibilidades que provee MQTT al momento de subscribirse a éstos. La principal característica tomada en cuenta para esta definición fue la posibilidad de utilizar *wildcards* en la subscripción. Esto significa que, por ejemplo (y usando la estructura de *topics* definida en el sistema), el servidor puede eventualmente escuchar todas las publicaciones de posiciones de ómnibus realizadas por los usuarios, en el área con identificador 1, subscribiéndose a «/server/area/1/line/#».

Además, los *topics* fueron estructurados de forma que los mensajes de tipo broadcast que envíe el servidor a varios clientes que esperan la misma información usen correctamente las posibilidades otorgadas por MQTT. Por tanto el *topic* a usar debe estar determinado por lo que los clientes esperan recibir. Por ejemplo, los clientes que quieran conocer la posición de los ómnibus de la línea “117 - PTA. CARRETAS” de Montevideo, estarán suscriptos al mismo *topic*, en este caso «/client/area/1/line/24» (asumiendo que Montevideo es el área con identificador 1, y 24 es el identificador de la línea). Así, cuando el servidor envíe la posición de los ómnibus para esta línea, lo hará al *topic* indicado, y Mosquitto se encargará de enviarlo a los clientes correspondientes.

4.6.3.4. Sintaxis de los mensajes

El protocolo fue desarrollado durante el proyecto en dos grandes etapas. Una descripción más detallada de las versiones resultantes, y una justificación más detallada de los factores que motivaron el cambio y las decisiones que se tomaron para atacar estos problemas se encuentra en el Anexo B. A continuación, se presenta un resumen de estos temas.

En el inicio del proyecto, ya habiendo decidido que sería usado el protocolo MQTT para el transporte de mensajes dentro de la aplicación, se pasó a la cons-

trucción de prototipos con el objetivo de validar que la aplicación era factible tecnológicamente, y que MQTT podía cumplir correctamente el rol que le había sido asignado. Como el diseño del sistema y de las interacciones entre sus componentes todavía no estaba completamente cerrado, se optó por que la sintaxis de los mensajes enviados sobre MQTT fuera la serialización en JSON de los datos a transmitir.

Esta codificación tuvo grandes ventajas, ya que permitió un desarrollo ágil de los componentes, donde se agregaron muchos cambios en la información transmitida muy rápidamente (al poseer una serialización/deserialización automática de los datos para este formato, era posible modificar su definición y no realizar cambios explícitamente en la codificación).

Mientras el foco estuvo en el desarrollo del sistema, investigando principalmente el desarrollo en Android y sus ramificaciones, este protocolo cumplió su propósito correctamente. Cuando esta etapa fue culminada, se pasó a una nueva fase donde la sintaxis del protocolo fue redefinida desde sus bases para lograr una implementación más eficiente en términos de tamaño de mensajes, uso de ancho de banda, y costo de codificación y decodificación.

En esta nueva etapa, se analizaron los mensajes enviados en el sistema, se identificaron los datos mínimos necesarios a ser enviados en cada uno, y se definió una codificación para los mensajes directamente en bytes. Para hacer esto, se elaboraron pautas de cómo codificar cada tipo de datos usado, y luego cómo codificar mensajes completos a partir de estos. Por más información sobre los cambios realizados en la sintaxis, ver el Anexo B.

Realizar el cambio insumió aproximadamente un mes en el que se diseñó, validó, implementó, y finalmente testeó la nueva versión. Si bien llevó un tiempo considerable, se mejoró bastante el tamaño de los paquetes enviados, llegando a reducir a un veinte por ciento del tamaño original algunos de los mensajes más frecuentes en el sistema, obteniendo una mejora global significativa.

4.7. Heurísticas

En la descripción original del problema se presenta el comportamiento ideal del sistema: lograr que la aplicación funcione de forma automática una vez instalada en los dispositivos de los usuarios.

Para realizar estas detecciones se deben incorporar heurísticas para la detección de eventos relevantes al sistema. Estos eventos pueden ser, por ejemplo, el momento en que un usuario se sube a un ómnibus, y a cuál ómnibus se subió, o los ómnibus

que el usuario usa habitualmente para aprovechar esta información más adelante. Para realizar la detección, las heurísticas deberán combinar la información de uno o más sensores con el conocimiento existente sobre la red de transporte.

Las heurísticas son fundamentales en la idea original del proyecto, ya que su existencia es una condición necesaria para la adopción masiva de la aplicación. De no existir, se dependería exclusivamente de los usuarios para ingresar manualmente la información en el sistema.

Se presentan en las secciones siguientes las heurísticas desarrolladas en este proyecto para detectar eventos relevantes al sistema a partir de los datos obtenidos desde los sensores presentes en los dispositivos.

En las pruebas realizadas con usuarios reales se encontró que es muy difícil que éstos recuerden indicar a la aplicación el momento en que se encuentran sobre un ómnibus, y cuándo bajan de estos. Las heurísticas están orientadas por lo tanto a estos dos momentos críticos de la interacción: el momento en el que el usuario sube al ómnibus, y el momento en que baja de él.

El desarrollo de las heurísticas se realizó bajo el marco descripto en la Sección 4.5.2. Al momento de definir una heurística, se establece cuál es el evento que dará inicio a su ejecución. Luego, si durante esta ejecución la heurística realiza una determinada inferencia, disparará el evento asociado al dato inferido.

Como consecuencia de este diseño, el acoplamiento entre las heurísticas, y entre éstas y la aplicación en general es mínimo, permitiendo realizar cambios en cada componente de forma separada, o incluso sustituir una heurística por otra más precisa.

4.7.1. Detección de paradas e inicio de viaje

Se presenta a continuación la heurística de detección de inicio de viaje, que tiene dos fases bien diferenciadas: la detección de que el usuario está en una parada de ómnibus y la detección del inicio de viaje en sí. Si efectivamente la heurística infiere los eventos correspondientes, se determina que el usuario se subió a un ómnibus.

Haciendo referencia al diagrama de estados de la aplicación, presentado en la Figura 4.4, mediante la primer fase de la heurística se puede pasar del estado *USER_WALKING* al estado *ON_BUSSTOP*, y mediante la segunda fase se podría determinar si desde el estado *ON_BUSSTOP,* pasa al estado *BUS_DETECTED* o vuelve al estado *USER_WALKING*.

Se detalla en la próxima sección el funcionamiento de los acelerómetros en dispositivos Android. Se presentará tanto desde una perspectiva teórica como práctica creada a partir de las experiencias realizadas durante la investigación previa al diseño de la heurística.

Luego, se presenta una descripción resumida, en un alto nivel de abstracción, de la heurística. En las secciones posteriores se describe en detalle el funcionamiento de las dos fases en que se dividió la heurística.

4.7.1.1. Acelerómetros en dispositivos Android

El primer hecho que debe destacarse, es que la medida obtenida de un acelerómetro es la llamada "aceleración propia" a la que está sometido el marco de referencia del sensor (en el caso de un dispositivo Android, el propio dispositivo). Generalmente, la aceleración propia no se corresponde con la aceleración del dispositivo, definida como el cambio en su velocidad observada desde un punto fijo sobre la superficie de la tierra.

Dado que el diseño del acelerómetro se basa en el desvío de la posición de equilibrio de una masa de referencia incluida dentro del sensor, el acelerómetro no detecta la aceleración gravitatoria experimentada por el dispositivo, ya que tanto la masa de referencia como el marco del sensor están sometidos a la misma fuerza gravitatoria. Por esta razón, si un dispositivo se encuentra estacionario en un plano horizontal, el valor retornado por el acelerómetro será una aceleración con valor g y dirección vertical con sentido hacia arriba (la aceleración asociada a la fuerza normal). El resultado sería el mismo si el dispositivo se encuentra en un movimiento rectilíneo uniforme sobre un plano horizontal. Por el contrario, si el dispositivo se encuentra en caída libre, el valor retornado por el acelerómetro es igual a cero [40].

Como etapa previa al diseño de la heurística, se realizaron pruebas empíricas para evaluar los valores retornados por el sensor en diversos ámbitos. En particular, se intentó buscar si era posible detectar si el usuario se encuentra en un estado estacionario o no con respecto a la tierra, asumiendo que lleva su dispositivo consigo. En teoría, y en un escenario ideal, realizar esta distinción es imposible en el caso de un movimiento rectilíneo uniforme, ya que en ambos casos la suma de las fuerzas (y por lo tanto las aceleraciones) ejercidas sobre el dispositivo sería nula. Sin embargo, si se analizan casos de uso reales, se encuentra que si un usuario lleva un dispositivo mientras se encuentra en movimiento éste se ve sometido a diversas fuerzas cuya suma no es nula. Esto es fácil de intuir si se analiza el caso de un usuario

caminando. Si bien el movimiento del usuario en su globalidad podría calificarse de rectilíneo uniforme, el movimiento del dispositivo, que probablemente se encuentre en alguna de las extremidades del usuario, no tiene esta característica. En otros casos podrían realizarse conjeturas similares, por ejemplo, en vehículos automotores podrían intervenir fuerzas asociadas a vibraciones inherentes al vehículo, o asociadas a irregularidades del terreno.

Se realizaron mediciones informales usando dispositivos Android para ver el comportamiento del acelerómetro en situaciones cotidianas, como las descritas en el párrafo anterior. Se observaron las medidas del acelerómetro mientras un usuario camina, está en un ómnibus, o está en un automóvil, y en todos los casos se encontró que si el usuario se encuentra en movimiento los valores detectados por el acelerómetro tienen una gran desviación respecto de su promedio, mientras que si el usuario está quieto la desviación de estos valores es menor.

4.7.1.2. Descripción

El propósito de la primer fase de la heurística es detectar cuando un usuario llega a una parada de ómnibus. La idea general para hacer ésto es detectar si el usuario está quieto, y en caso de que lo esté, verificar si su posición se corresponde con la de alguna parada de ómnibus.

La detección del movimiento o quietud del usuario, se realiza mediante los valores obtenidos del acelerómetro presente en su dispositivo. Para un período de muestras, la heurística desarrollada infiere que el usuario está quieto en casos donde la desviación de los valores obtenidos del acelerómetro respecto a su promedio es baja. Si existen sucesivos períodos que indican que el dispositivo no está en movimiento, se infiere que el usuario está quieto.

A continuación, se espera a tener una posición de GPS lo suficientemente buena para determinar si dicha posición se corresponde con una parada de ómnibus. En caso de que se confirme que está en una parada de ómnibus, la primer fase se considera finalizada, y empieza a ejecutarse la segunda fase.

El objetivo de la segunda fase es detectar el inicio de viaje, es decir, el hecho de que el usuario se subió a un ómnibus. Sabiendo que el usuario se encuentra en una parada, la idea general de esta fase es determinar si la velocidad con la cual se aleja de la parada se corresponde con la de un ómnibus.

Dado que cuando el usuario se encuentra en la parada de ómnibus pueden existir movimientos mínimos cerca de ésta, se empieza a calcular la velocidad una vez que

se alejó una cantidad dada de metros de la parada. Cuando esto ocurre, se toma un número de posiciones obtenidas por GPS, y en caso de que la velocidad media calculada haciendo uso de dichas posiciones sea similar a la de un ómnibus, se determina que el usuario inició el viaje.

4.7.1.3. Implementación de la primera fase: «Detección del usuario en la parada de ómnibus»

Esta primer fase de la heurística, además de inferir el estado de quietud o movimiento del usuario, necesita información sobre la ubicación de las paradas de ómnibus para poder detectar (en el caso en que se detecta que el usuario está quieto) si el usuario está en alguna de ellas. Dichas paradas y sus ubicaciones son almacenadas en el dispositivo, y son actualizadas cuando es necesario, dependiendo de la posición del usuario. Esto quiere decir que en todo momento el dispositivo conoce las paradas de ómnibus cercanas a su posición actual.

Con el propósito de consumir menos batería, y dado que para detectar movimientos del usuario en cientos de metros no es necesario contar con una posición precisa, la heurística se suscribe al posicionamiento basado en la red de celulares y consulta al servidor sobre las paradas que están cerca del usuario, a una distancia menor a mil metros. En caso de que el usuario se mueva más de cierta distancia desde la última vez que se solicitaron las paradas, se vuelve a consultar al servidor sobre las paradas más cercanas usando la nueva posición del usuario, y se sustituyen las anteriores por estas.

En la Figura 4.9 se muestra un diagrama que representa los estados internos del primer paso de la heurística: la detección del usuario en una parada de ómnibus. Este diagrama asume que el mecanismo de obtención de las paradas cercanas al usuario está en funcionamiento en forma independiente.

Esta primera fase de la heurística se basa fuertemente en el uso del acelerómetro para determinar si el usuario está quieto o en movimiento.

Para hacer uso del acelerómetro, al igual que con otros sensores de Android, hay que suscribirse a un manejador de sensores, indicando el sensor a utilizar. A partir de ese momento, llegan muestras con la frecuencia que se haya indicado. Para los parámetros usados en la implementación de esta heurística, los valores son recibidos con una frecuencia promedio de 5 Hz, aunque este valor depende del fabricante y del modelo del dispositivo. Finalmente, cuando no se desean recibir más mediciones del sensor, se debe cancelar la suscripción del manejador de sensores.

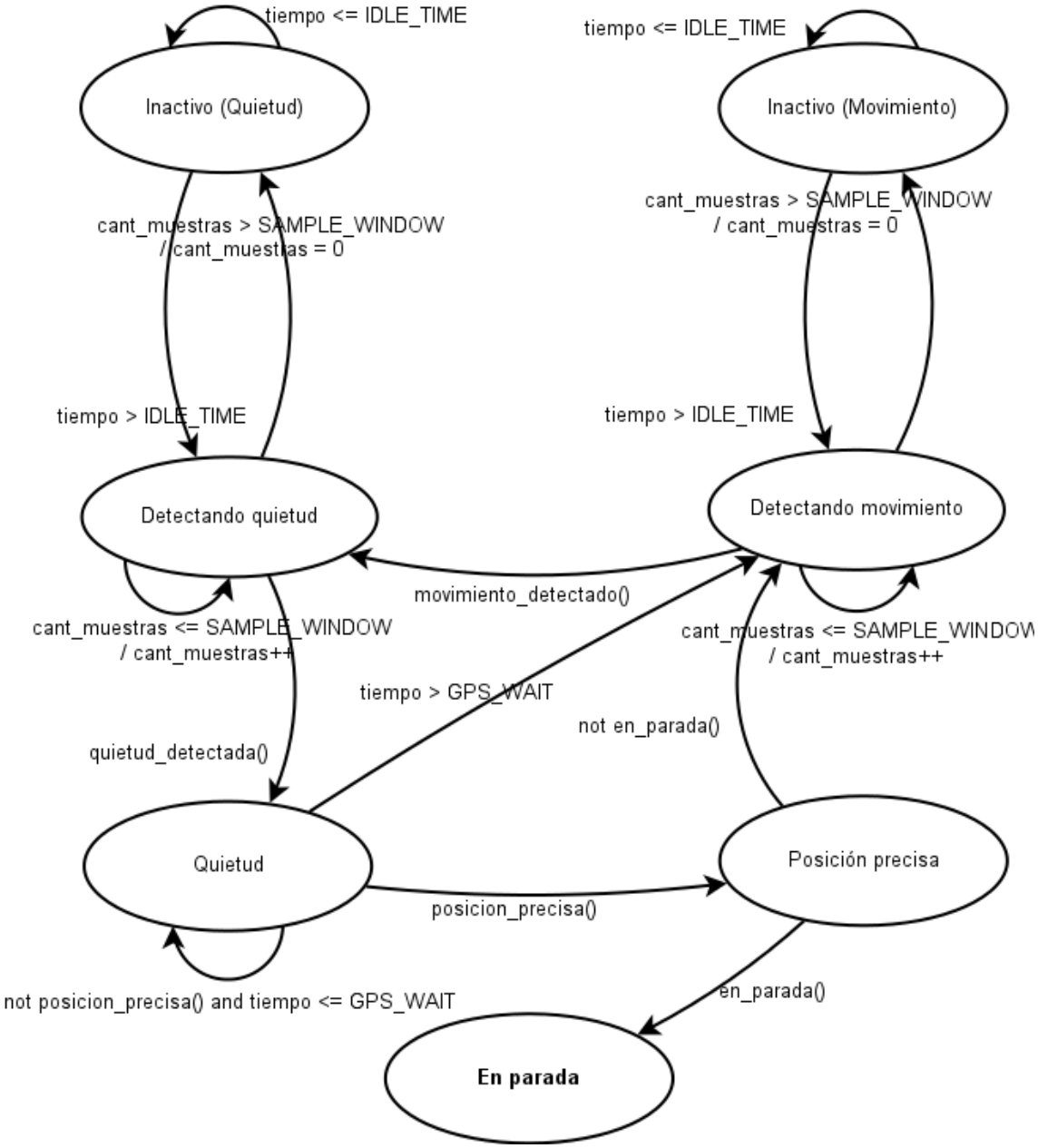


Figura 4.9: Detección de usuario en parada de ómnibus

Con el objetivo de consumir menos batería, y siguiendo las mejores prácticas de desarrollo en Android en cuanto a sensores [39], en los únicos estados en los que el sensor se encuentra prendido son «Detectando quietud» y «Detectando movimiento». La aplicación alterna entre los estados «Detectando» e «Inactivo» (en sus dos variantes) para tomar las muestras.

Se considera que un período de muestras del acelerómetro es el conjunto de mediciones recibidas entre la suscripción y la cancelación de la suscripción al sensor.

En la Figura 4.10 se puede ver un ejemplo ficticio de los valores retornados por el acelerómetro en el paso del tiempo. El valor mostrado es el valor absoluto de la magnitud de la suma vectorial de las tres aceleraciones retornadas por el acelerómetro: la aceleración en el eje X, en el eje Y y en el eje Z. La heurística se encuentra suscripta a las actualizaciones del acelerómetro sólo en los períodos de muestras.

Para cada período de muestras se realiza una evaluación para determinar si los valores del acelerómetro en el período se corresponden con un estado de quietud o de movimiento. Para concluir que el dispositivo se encuentra quieto en el período, estos valores no deben estar alejados del promedio para ese período particular; en caso de no poder determinar un estado de quietud se considera que el usuario estuvo en movimiento durante dicho período.

Por ejemplo, en el «Período de muestras 1» los valores retornados por el acelerómetro se alejan bastante del promedio en dicho período, por lo que la evaluación de la heurística determinaría en ese caso que el usuario está en movimiento. En el caso del «Período de muestras 2», los valores no varían demasiado y todos tienen diferencias pequeñas con respecto al promedio, por lo que la evaluación daría como resultado un período de quietud.

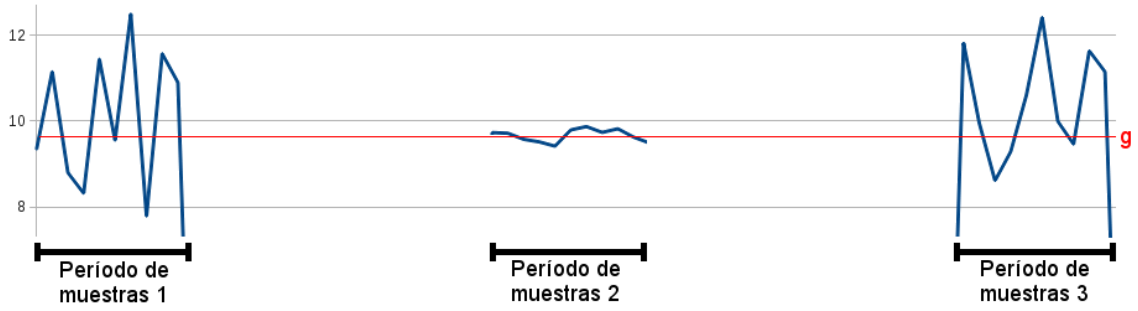


Figura 4.10: Períodos de muestras del acelerómetro

Para determinar si el usuario está quieto o en movimiento, la heurística considera las evaluaciones correspondientes a los últimos cinco períodos de muestras. La heurística determina que el usuario está quieto o en movimiento si un porcentaje dado de las evaluaciones consideradas se corresponde con períodos de quietud o movimiento, respectivamente. Esto quiere decir que si el porcentaje mínimo de períodos de quietud para considerar que el usuario está quieto es 75 por ciento, de las últimas cinco evaluaciones, al menos cuatro se deben corresponder con períodos de quietud.

Una vez detectado que el usuario está quieto se enciende el sensor de GPS para verificar si su posición es cercana a alguna de las paradas conocidas. Si se determina

que el usuario está a menos de una distancia determinada de alguna parada (considerando márgenes de error), la heurística infiere que el usuario está en esta parada. En caso contrario, se continúa tomando muestras con el acelerómetro, pero, para evitar que vuelva a detectarse el mismo punto innecesariamente, se requiere que la heurística detecte que el usuario estuvo en movimiento, para luego volver a buscar un estado de quietud. Esto es importante para evitar consultas al GPS innecesarias (que acarrean un gasto de batería mayor). Lo mismo ocurre si pasa un cierto tiempo sin que el dispositivo logre obtener una posición aceptable de GPS que pueda comparar con las posiciones de paradas conocidas.

Cuando la heurística infiere que el usuario está en una parada, se inicia la segunda fase de la heurística que tiene como objetivo detectar el momento en que el usuario inicia el viaje en ómnibus.

4.7.1.4. Implementación de la segunda fase: «Detección de inicio de viaje»

Se muestra en la Figura 4.11 un diagrama que representa los estados internos de la segunda fase de la heurística: la detección de inicio de viaje, es decir, el momento en que el usuario se subió al ómnibus.

Para detectar el momento en que el usuario se sube a un ómnibus (y por lo tanto inicia su viaje), se optó por implementar una heurística basada en la posición de GPS del usuario. Esta decisión se debió principalmente a la posibilidad de plantear una heurística simple en términos de esta posición, y debido a que el gasto de tener el sensor de GPS encendido es bajo dado el breve lapso en que debería estar activa esta heurística.

En el momento en que se inicia esta heurística, la aplicación obtiene desde el servidor la lista de líneas de ómnibus que deben parar por la parada detectada en el paso anterior.

Esta heurística define dos zonas circulares con centro en el punto donde se encuentra en la parada, con radios d_1y d_2. Cuando el usuario se aleja más de d_1 de la parada, se almacena el tiempo y posición en que ocurrió este evento. Mientras el usuario se encuentra con una distancia a la parada mayor a d_1y menor a d_2, la aplicación acumula la distancia recorrida dentro de esta zona (ya que la ruta seguida por el usuario en este punto puede no ser rectilínea). Cuando el usuario abandona la zona definida por d_2, se calcula la velocidad promedio llevada en el recorrido anterior.

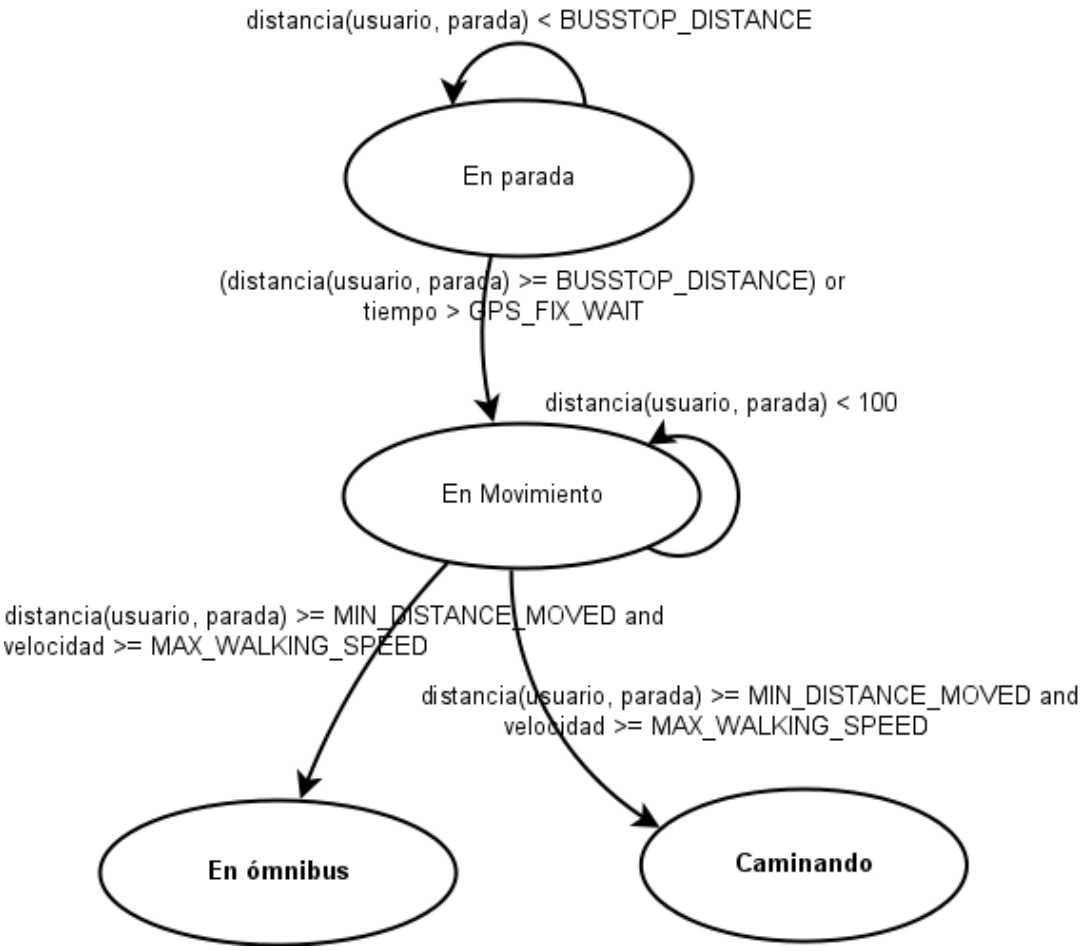


Figura 4.11: Detección de inicio de viaje

Si esta velocidad es menor a un cierto umbral, se infiere que el usuario probablemente se alejó caminando de la parada, y por lo tanto la aplicación debe volver al estado inicial (esperando detectar el momento en que el usuario llega a una parada).

En el caso contrario, se inicia una notificación al usuario preguntando si se encuentra en algún ómnibus, mostrándole la lista de líneas de ómnibus obtenida en el inicio de la ejecución para que el usuario seleccione en cuál se encuentra. También se provee la opción de ver todas las líneas, para manejar el caso en que la línea existe, pero no se conoce aún que la parada está asignada a la línea.

4.7.1.5. Verificación

Para verificar esta heurística se realizaron pruebas manualmente, ingresando información sobre paradas reales en la base de datos, y probando casos de uso intentando simular lo más precisamente posible el caso para el que fue diseñada la

heurística.

4.7.2. Descenso de ómnibus

Para detectar automáticamente el momento en que el usuario desciende de un ómnibus, se implementó una heurística ejecutada completamente en el teléfono basada en las posiciones de GPS obtenidas por éste.

Haciendo referencia al diagrama de estados de la aplicación, presentado en la Figura 4.4, esta heurística puede determinar que la aplicación pase del estado *ON_BUS* al estado *USER_WALKING*.

En este caso, hacer un uso intensivo del GPS no acarrea un costo adicional, ya que en el momento en que se ejecuta la heurística (el estado ON_BUS de la aplicación), el sensor de GPS ya se encuentra encendido.

Esta heurística es muy importante, ya que es común que un usuario indique su presencia en un ómnibus particular, y luego olvide indicar el momento en el que desciende del mismo. Por esta razón, en muchas ocasiones se muestran dentro de la aplicación ómnibus en posiciones incorrectas. Por ejemplo, si el usuario se bajó y camina hasta su destino, la aplicación mostrará (siempre que no exista la heurística aquí presentada) un ómnibus en la posición correspondiente al destino del usuario.

Tanto para el diseño como para la verificación de esta heurística se utilizaron las trazas reales obtenidas con las primeras versiones de la aplicación.

En la etapa de diseño, fueron fundamentales para encontrar patrones comunes en trazas donde el usuario olvidó indicar que la aplicación no debía publicar más información que ayudasen a detectar el momento en que éste debía haberlo hecho. Además, se analizaron también trazas «correctas» (donde el usuario había indicado correctamente ambos extremos del viaje), para verificar que la heurística a implementar no detectase falsos positivos en estos casos.

En la etapa de verificación, se usó el juego de datos como referencia, realizando una simulación de la ejecución de la heurística para verificar que ésta detectase correctamente el momento en que el usuario descendió del ómnibus.

4.7.2.1. Diseño

Del análisis de las trazas obtenidas, se observó que aquellas donde el usuario había descendido del ómnibus sin indicarlo en la aplicación, eran fácilmente diferenciables del resto. Las posiciones reportadas en los tramos en los que el usuario se encontraba caminando, luego de descender del ómnibus, están a una distancia menor entre sí

que las posiciones cuando el usuario estaba en el ómnibus. Como las posiciones son publicadas en intervalos constantes de tiempo, una distancia menor entre posiciones implica una velocidad menor del movimiento del usuario.

Estos datos empíricos se corresponden con que la velocidad de una persona caminando, que oscila entre los 3 y 4,5 km/h[2], es menor a la de un ómnibus, que puede alcanzar velocidades cercanas a 60 km/h en el transporte público.

El diseño de la heurística está basado en la observación presentada. Si el usuario se mueve a una velocidad inferior a una cierta velocidad de referencia a determinar, llamada a partir de ahora v_{MAX}, durante una cierta cantidad de muestras también a determinar, en adelante llamada n_{MIN}, la aplicación asumirá que el usuario se ha bajado del ómnibus.

Analizando las trazas, se encontró que la heurística tenía el riesgo de encontrar falsos positivos en casos en que el ómnibus se encontrase estacionario durante mucho tiempo, por ejemplo en una parada, en un semáforo, o en casos donde hay congestionamientos puntuales en el recorrido.

Por esta razón, se optó por agregar un parámetro más a la heurística, que estableciera que el usuario debía trasladarse una cierta distancia, en adelante llamada d_{MIN}, como mínimo para que se detectase el descenso por parte del usuario. Con esta modificación, se evita la detección de casos donde el usuario se mantuvo estacionario o con velocidades muy bajas durante más de n_{MIN} muestras, y esta baja velocidad se corresponde a que el ómnibus en el que viaja el usuario se encuentra esperando en una parada, o en un congestionamiento del tránsito.

Con esta última modificación, si un usuario descendiese del ómnibus para ir a un lugar cercano a la parada (con distancia menor a d_{MIN} de esta), y permanecer estacionario en esta ubicación, la heurística no inferiría un descenso del ómnibus, dado que el usuario no se alejó lo suficiente. Por esta razón, se optó por agregar un cuarto parámetro n'_{MIN} y lógica adicional a la heurística, de forma que ésta infiera que el usuario descendió del ómnibus, siempre que el usuario se mantenga con una velocidad inferior a v_{MAX} durante más de n'_{MIN} publicaciones.

En la Figura 4.12 se muestra un diagrama de la máquina de estados que modela lo descripto anteriormente. Todas las transiciones se realizan en el momento de recibir una nueva posición de GPS. En cada arista se muestran en su inicio las condiciones que deben cumplirse para transitar por ella, y en su fin las acciones realizadas en la transición.

[2] http://es.wikipedia.org/wiki/Peat%C3%B3n

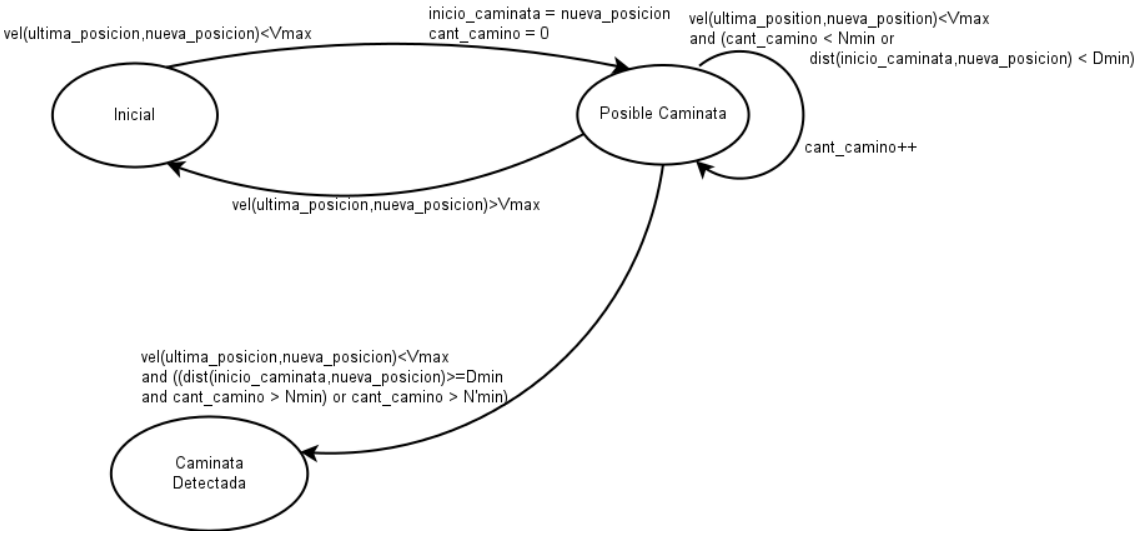


Figura 4.12: Diagrama de estados de la heurística de descenso de ómnibus

4.7.2.2. Verificación

Como ya se comentó, para la verificación y ajuste de la heurística se utilizaron los datos obtenidos a partir de las ejecuciones.

Como paso previo al ajuste y verificación de la heurística, se procedió a evaluar manualmente estas trazas, determinando si estas trazas se correspondían con situaciones donde el usuario seguramente había olvidado apagar la publicación de su posición, y en caso que hubiese olvidado hacerlo, establecer cuál es la posición más plausible para la parada donde el usuario había descendido del ómnibus.

Se optó por dividir el conjunto de trazas disponible en dos grupos: Uno (llamado en adelante grupo de calibración) usada para obtener valores apropiados para los parámetros v_{MAX}, n_{MIN} y d_{MIN}, y otro (en adelante grupo de verificación) para probar los valores encontrados y verificar la heurística.

Para la determinación de los valores se optó por realizar pruebas simples (posibles ya que la heurística es fácilmente evaluable), donde se ejecuta la heurística para todos las trazas pertenecientes al grupo de calibración, alterando el valor de los parámetros. Se establecieron estimando a partir de medidas cotas máximas y mínimas para estos parámetros, y dentro de los intervalos correspondientes se tomaron valores equiespaciados entre sí. La heurística se probó para cada combinación de estos valores, y se obtuvo la combinación de parámetros que mejor cubre al conjunto de calibración. Para determinar cuál era la mejor combinación, se observó en primer lugar la cantidad de detecciones correctas, y en caso de empate se eligió la combinación que en promedio detectaba el descenso del usuario a una distancia menor de

la parada de ómnibus.

Luego de estas pruebas, se determinó que los valores óptimos, para el juego de datos de calibración, son:

$$v_{MAX} = 2{,}3611 m/s$$

$$n_{MIN} = 8$$

$$d_{MIN} = 60m$$

Con estos valores para los parámetros de la heurística, se procedió a probarla contra el conjunto de verificación. Para cada punto se evaluó que se detectase correctamente si el usuario había indicado su descenso a tiempo o no, y que la detección se realizara a una distancia razonable de la parada correspondiente. El resultado, fue que la heurística clasificó correctamente el 86 % de los casos. En la Sección 6.2 se muestran casos reales de los usados para verificar esta heurística, ilustrando las situaciones que se encontraron.

4.7.3. Alerta de proximidad de destino

El caso de uso/heurística se inicia en el momento en que el usuario indica a la aplicación que se encuentra sobre un ómnibus.

La aplicación en este punto obtiene desde el servidor las paradas correspondientes a la línea a la que pertenece el ómnibus en el que se encuentra el usuario, y las muestra sobre el mapa junto con los ómnibus mostrados. El usuario puede indicar en qué parada de éstas se bajará.

Si el usuario presiona una parada en el mapa (podría nunca hacerlo), la aplicación muestra un diálogo preguntando a qué distancia de la parada el usuario desea ser notificado. De forma predeterminada esta distancia es de trescientos metros, pero el usuario puede modificarla a cualquier otra distancia, llamada en adelante d_{ALERTA}. La aplicación asume en ese momento que el usuario va a descender del ómnibus en la parada indicada.

La operación de establecer una parada destino puede ser repetida por el usuario tantas veces como se desee, para obedecer a posibles cambios en el destino al que quiera llegar el usuario, o correcciones sobre los datos ingresados. Siempre se muestra

en pantalla la parada seleccionada actualmente con algún distintivo visual, de forma que el usuario sepa cuál es el destino asumido por la aplicación.

Luego de seleccionarse una parada como destino, la aplicación establece dos alertas de proximidad:

- A la distancia d_{ALERTA} del punto en que se encuentra la parada. Cuando se dispare esta alerta se elevará una notificación al usuario para recordarle que debe bajar en ese lugar.

- A una distancia de cincuenta metros del punto en que se encuentra la parada. Cuando se dispare esta alerta la aplicación verificará que la distancia a la parada sea menor a cincuenta metros con una precisión menor a cincuenta metros, y en este punto asume que el usuario descendió del ómnibus, dejando de compartir la posición del ómnibus sin requerir interacción con el usuario.

4.8. Resumen

En este capítulo se presentó la solución implementada en este proyecto. Se inicia con la definición del alcance, y se incluye luego una comparación del sistema definido por este alcance contra las aplicaciones relevadas en el Capítulo 2.

Se pasa luego a mostrar el diseño propuesto para implementar la solución correspondiente al alcance definido, destacándose la definición del modelo de datos, la arquitectura propuesta, los mecanismos de comunicación, y el comportamiento de la aplicación, incluyendo las heurísticas de detección de eventos.

Durante el proyecto se implementó todo lo mencionado en este capítulo. En el Capítulo 6 se muestran los resultados obtenidos en el uso de la aplicación, indicando hasta qué punto la solución propuesta cumple con los requerimientos incluidos en el alcance.

Capítulo 5

Desarrollo del proyecto

Se presenta a continuación un resumen del desarrollo del presente proyecto desde el punto de vista de la gestión.

En primer lugar, se muestran los resultados obtenidos en las actividades orientadas al aprendizaje del sistema operativo Android y su SDK asociado, a modo de referencia para estimaciones en proyectos donde el equipo de trabajo parta de una situación similar.

Luego, se presenta un breve resumen del desarrollo del proyecto, su planificación y las desviaciones que sufrió a lo largo del tiempo.

Por último, en la Sección 5.3, se presentan actividades realizadas en el marco del proyecto, pero no relacionadas directamente al desarrollo del sistema.

5.1. Aprendizaje de Android

La situación original del equipo de trabajo en los inicios del proyecto era desigual en lo que respecta a conocimientos del sistema Android. Si bien todos los integrantes tenían conocimientos de usuario sobre la plataforma, solo uno de ellos tenía conocimientos a nivel de desarrollador. Estos conocimientos eran limitados al manejo de la interfaz de usuario y el *framework* provisto por la plataforma para la persistencia, pero fueron útiles en los inicios del proyecto.

Se muestran a continuación ciertas lecciones aprendidas sobre la programación en Android, desde diferentes perspectivas.

5.1.1. Programación en Java

Si bien en un bajo nivel la máquina virtual que ejecuta el código de aplicación en un dispositivo Android no es equivalente a una máquina virtual Java tradicional (clases compiladas con Java estándar no pueden correr directamente en Android), en la mayoría de los casos pueden considerarse equivalentes. Esto facilitó mucho el aprendizaje, ya que todos los integrantes del equipo tenían experiencia en el lenguaje.

5.1.2. Plataforma Android

Aunque el conocimiento de Java es una precondición para poder desarrollar aplicaciones en Android, es necesario poseer conocimientos complementarios para hacerlo. Principalmente, debe conocerse la arquitectura propuesta para las aplicaciones, los componentes básicos provistos por el sistema operativo, y cómo usarlos. Estos conocimientos son necesarios sin importar qué tipo de aplicación se desee construir. En este proyecto el aprendizaje se inició con la construcción de prototipos, por lo que se obtuvieron resultados de forma rápida sin tener aún conocimientos profundos sobre la plataforma. El costo en tiempo para llegar a un nivel suficiente de conocimiento sobre la plataforma fue entre dos y tres meses.

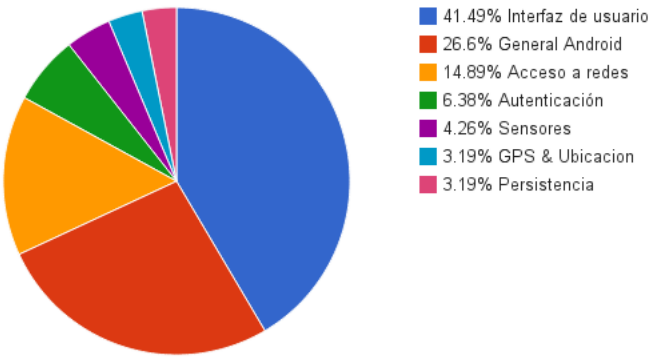


Figura 5.1: Interfaces integradas por tipo

En la Figura 5.1 se muestra la distribución de las interfaces según el área del sistema, que tuvieron que ser aprendidas por el equipo en el transcurso del proyecto. En total, la aplicación implementada se integra con 94 interfaces de terceros, en su mayoría propias de Android. El único área donde existen integraciones con bibliotecas externas a Android es «Acceso a redes», donde se utilizan bibliotecas

pertenecientes al proyecto Paho[1], implementación de MQTT, protocolo utilizado como se describe en la Sección 4.6.2.

5.1.3. Uso de sensores

En el caso particular de este proyecto, fue necesario aprender sobre aspectos avanzados en el desarrollo para Android. Esto se debió principalmente a la necesidad de interactuar con los sensores presentes en los dispositivos, que tienen APIs específicas para interactuar con cada uno. En este caso fue fundamental el aprendizaje a partir de la generación de prototipos desechables para validar tanto que la API se comporte como se espera, como las características físicas de los propios sensores. El tiempo necesario para el aprendizaje en esta área depende mucho de los sensores a usar y el nivel de integración de éstos con la aplicación a construir.

Si bien en la Figura 5.1 se muestra que la cantidad de interfaces usadas para la integración con los sensores es baja en comparación al total, cabe mencionar que el tiempo insumido en el aprendizaje de su uso es comparativamente alto. Esto se debe a que se deben comprender tanto aspectos de la interfaz desde un punto de vista de software, como características físicas de los sensores para poder interpretar correctamente su salida. Además, se deben evaluar las diferentes implementaciones de cada sensor que pueden existir en distintos dispositivos.

5.1.4. Documentación

Como fuente de documentación sobre Android se utilizó principalmente la página oficial de Google [14] que concentra toda esta información, complementándola ocasionalmente con información puntual obtenida de otras fuentes.

La documentación disponible sobre el sistema tiene una cobertura muy amplia en las áreas correspondientes al *framework* básico. En algunos temas avanzados no se encuentra documentación suficiente, sería útil contar con ejemplos de código sobre el uso de funcionalidades comunes.

El problema de fragmentación del sistema operativo tiene consecuencias sobre la documentación del mismo, tanto dentro de la documentación oficial (donde existen documentaciones distintas según la versión), como fuera de los canales oficiales. Muchas veces se encontraron soluciones a problemas que sólo eran válidas para versiones específicas del sistema operativo.

[1] http://www.eclipse.org/paho/

5.2. Desarrollo del sistema

Se presenta a continuación una breve reseña de cómo se desarrolló el proyecto en el tiempo, cuáles fueron sus etapas más importantes, y los hitos que se alcanzaron.

Se dividió esta sección en tres partes.

Primero, en la Sección 5.2.1, se comentan las actividades que fueron realizadas por el equipo antes de iniciar el desarrollo del sistema.

Luego, en la Sección 5.2.2, se describen las iteraciones en las que se dividió el desarrollo del sistema, indicando en cada una los componentes desarrollados.

En último lugar, en la Sección 5.2.3, se describen las liberaciones realizadas, a quiénes fueron dirigidas, y los resultados obtenidos de éstas.

5.2.1. Etapa inicial

Durante esta etapa el foco estuvo concentrado principalmente en la investigación de las tecnologías a usar para la resolución del problema. En este momento del proyecto se construyeron varios prototipos para validar las posibilidades provistas por la plataforma Android, principalmente en el acceso a los sensores. Además, se hicieron experimentos orientados a la búsqueda de mecanismos eficientes de comunicación entre el servidor y los clientes para la notificación de las posiciones de ómnibus. En particular, se investigó Twitter como una posible solución a este problema, por más detalles ver el Anexo A.

Las principales actividades realizadas fueron:

1. Construcción de prototipos en Android: se crearon varios prototipos validando el uso de varias funcionalidades provistas por el sistema. Se abordaron los siguientes aspectos del sistema: uso de Wi-Fi para detectar redes cercanas, uso de mecanismos para encontrar la ubicación del dispositivo, uso de conexión del dispositivo (en particular para publicar a Twitter, ver Anexo A), definición de tareas en segundo plano en Android, acceso al sensor de Bluetooth. Además, se investigaron las posibilidades en Android para integrar los mapas provistos por Google Maps a una aplicación.

2. Búsqueda de mecanismos de comunicación eficientes para la distribución de mensajes en Android: los resultados de esta búsqueda se encuentran detallados en el Anexo A.

3. Creación de un modelo genérico del sistema de transporte público (considerando sólo sistemas basados en ómnibus).

El resultado de esta etapa fue considerado positivo ya que se logró implementar prototipos validando las posibilidades de la plataforma. A partir de estas validaciones se definió la arquitectura a usar para el sistema.

5.2.2. Iteraciones

Luego de finalizada la etapa inicial, la aplicación y el servidor fueron construidos en 5 iteraciones. Estas iteraciones se corresponden con objetivos claros, establecidos antes de iniciar la iteración junto con el tutor. Las iteraciones, junto con los objetivos logrados en cada una, se detallan a continuación.

5.2.2.1. Primera iteración

Esta iteración tuvo como principal objetivo la creación de un prototipo no desechable cubriendo las funcionalidades básicas de la aplicación. Para lograr este objetivo, se puso especial énfasis en la implementación de la comunicación entre los dispositivos y el servidor usando MQTT, para verificar que el mecanismo de comunicación era posible.

Los casos de uso implementados fueron la publicación de una posición de ómnibus por parte del usuario, y la consulta de posiciones para una línea determinada.

Inicialmente se había planificado esta iteración con una longitud de un mes y medio, pero finalmente se atrasó casi un mes. Las causas del atraso fueron principalmente una subestimación del tiempo necesario para aprender Android (dado que en esta etapa el equipo no tenía aún mucha experiencia de desarrollo), y el tiempo necesario para integrar correctamente a Mosquitto. Hubo una dedicación importante a la corrección de errores luego de finalizar la implementación correspondiente a esta iteración, los cuales fueron introducidos debido a la baja experiencia del equipo en el desarrollo para Android.

Más allá del atraso, el resultado de esta primera iteración es positivo, ya que se logró un prototipo funcional con el caso de uso más crítico funcionando correctamente en la mayoría de los casos. Más adelante se realizaron muchos ajustes a lo desarrollado en esta versión, pero la arquitectura se mantuvo sin mayores cambios, por lo que esta iteración puede considerarse como exitosa.

5.2.2.2. Segunda iteración

En esta iteración se agregó el componente social a la aplicación. Este agregado no presentó problemas mayores en su desarrollo gracias a los conocimientos obtenidos en la primer iteración.

Además, se aprovechó este tiempo para incorporar a la aplicación muchas sugerencias recibidas de usuarios y docentes. Si bien esta iteración tuvo pocas actividades de implementación, se avanzó mucho en la planificación de lo que se implementaría en la siguiente.

5.2.2.3. Tercera iteración

La necesidad de planificar el desarrollo de esta tercera iteración fue causada por el impacto de los cambios a introducir en el sistema. Hubo un solo cambio pero con un gran impacto, que fue la mejora del protocolo usado para las comunicaciones para ser más eficiente en términos de CPU y ancho de banda consumido.

En el Anexo B se presenta cómo evolucionó el protocolo en el tiempo. En resumen, las modificaciones se centraron en la codificación de los mensajes enviados y un análisis de qué datos son estrictamente necesarios para cada mensaje intercambiado en el sistema, con la intención de ahorrar ancho de banda.

Además de la modificación del protocolo, se realizaron muchas mejoras en la aplicación con el objetivo de estabilizarla, eliminando errores cuya corrección se había postergado.

5.2.2.4. Cuarta iteración

En esta iteración el trabajo estuvo concentrado en la creación de una interfaz de administración en el servidor, para brindar herramientas que apoyen en el monitoreo del sistema, viendo en tiempo real los eventos que están ocurriendo.

En esta iteración se corrigieron también errores reportados por los usuarios de la aplicación, y se incorporaron algunas sugerencias provenientes tanto de usuarios como de docentes. De forma similar a lo ocurrido con la segunda iteración, se planificaron en esta iteración las actividades a realizar en la siguiente.

5.2.2.5. Quinta iteración

Durante esta iteración se diseñaron, implementaron, y verificaron las heurísticas de detección de eventos. Las heurísticas implementadas en este punto son las

descriptas en la Sección 4.7.

Para la heurística de descenso del ómnibus, como se menciona en la Sección 4.7.2, se pudo reutilizar el conocimiento existente en las trazas almacenadas durante las pruebas de versiones anteriores, simplificando la verificación.

En otros casos esto no fue posible, obligando al equipo a probar las heurísticas ejecutando los casos de uso de la aplicación que incluyen heurísticas (llegada a la parada de ómnibus, subida y descenso de ómnibus) de la misma forma que lo harían los usuarios. En una primera instancia este mecanismo fue usado para calibrar las heurísticas, y luego para verificarlas.

5.2.3. Liberaciones

En el transcurso del proyecto se realizaron tres liberaciones a grupos progresivamente mayores de usuarios finales. Estas liberaciones, realizadas en el fin de las iteraciones uno, cuatro, y cinco, tuvieron tres objetivos en común.

Por un lado obtener retroalimentación de usuarios en cuanto a la usabilidad de la aplicación, y los problemas que encontraron en su uso. En este sentido, se obtuvieron comentarios útiles que fueron en su mayoría incorporados a versiones posteriores de la aplicación.

Por otro lado, se intentó recabar información sobre posibles errores en distintos modelos y distintas versiones de Android. Dado el problema de fragmentación de este sistema operativo, las pruebas realizadas en los dispositivos a los que el equipo del proyecto tenía acceso no eran garantía del correcto funcionamiento de la aplicación en otros. En este frente, se encontró que en la mayoría de los casos la aplicación se comportó correctamente, aunque con variaciones en el comportamiento de los sensores. En casos puntuales, existieron usuarios que proveyeron datos de errores específicos a algunos modelos, que de otra forma no hubieran sido detectados.

Por último, se obtuvo información a través de estas liberaciones que sirvieron para diseñar y verificar la heurística de descenso del ómnibus.

Se presentan a continuación las liberaciones realizadas.

5.2.3.1. Liberación 1

Realizada el 15 de agosto de 2012 (fin de la primera iteración), a un grupo reducido de voluntarios conocido y elegido por el equipo. El volumen que accedió a la aplicación en esta etapa fue de aproximadamente quince personas.

A partir de esta liberación se obtuvieron datos que sirvieron a la mejora de la aplicación, principalmente en lo referente a la portabilidad de ésta a distintos modelos de dispositivos Android, ya que con esta liberación se amplió significativamente el conjunto de dispositivos donde fue probada la aplicación.

Además, se obtuvieron muchas sugerencias referentes a la usabilidad de la aplicación, en su mayoría incorporadas a la aplicación final.

Desde un punto de vista técnico, se verificó que la arquitectura propuesta podía resolver correctamente el problema.

5.2.3.2. Liberación 2

Realizada el 14 de noviembre de 2012, en el marco de la presentación a estudiantes de redes que se detalla en la Sección 5.3.3 (en el fin de la cuarta iteración). Simultáneamente, se publicitó la liberación en redes sociales. A través de estos medios, se llegó a un grupo de aproximadamente 70 personas.

Por la naturaleza de la liberación, no hubo una selección sobre quiénes instalaron la aplicación, ni hubo posibilidad de entregar capacitación a los usuarios. En su lugar, se optó por publicar, junto con la aplicación, una página web resumiendo su funcionamiento.

Si bien la aplicación contiene un mecanismo interno para enviar comentarios al equipo de trabajo, los usuarios no hicieron uso de éste. Esto, sumado al hecho de no conocer a los usuarios que probaron la aplicación, significó que la cantidad de comentarios fue mucho menor que en la liberación anterior.

No obstante, la información recabada a partir del uso de la aplicación en esta instancia fue importante a los efectos de diseñar las heurísticas implementadas en la quinta iteración, detalladas en la Sección 4.7.

Desde un punto de vista técnico, la liberación fue exitosa ya que el producto liberado cubrió prácticamente toda la funcionalidad buscada , salvo las heurísticas implementadas más adelante. Se validó que la versión definitiva del protocolo y el componente social funcionaban correctamente.

5.2.3.3. Liberación 3

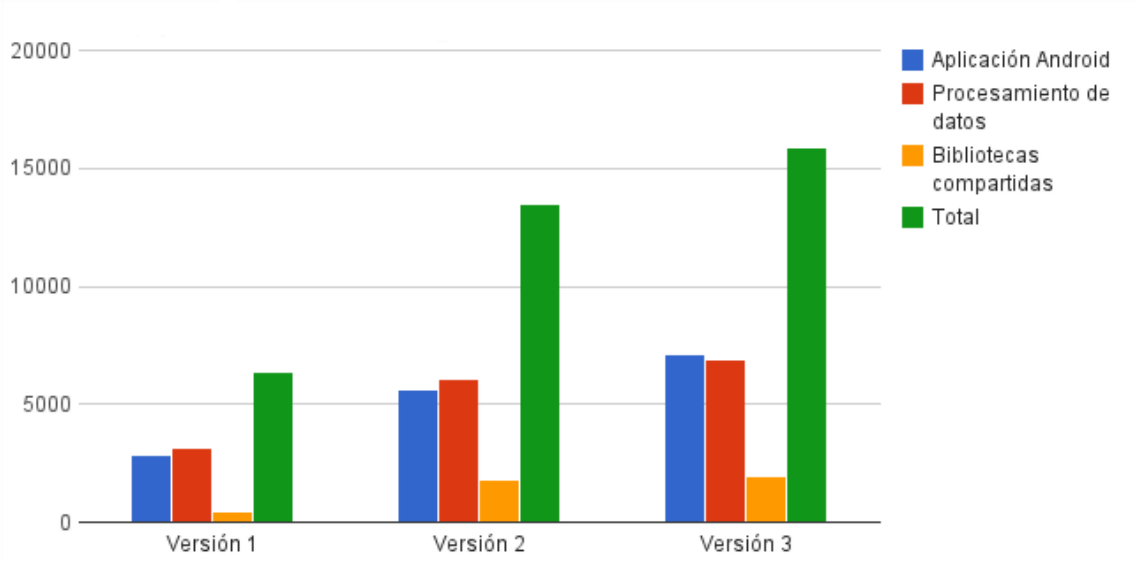
Realizada el 26 de febrero de 2013, a fines de la iteración 5, representa el estado final de la aplicación en términos de alcance. El mecanismo de liberación fue similar al de la anterior. Se actualizó la página web, y se notificó a los usuarios de la aplicación de la existencia de la nueva versión.

El mayor cambio introducido en esta liberación son las heurísticas presentadas en la Sección 4.7.

Esta liberación sirvió principalmente para la verificación del producto final.

5.2.3.4. Datos de las liberaciones

En la Figura 5.2 se muestra la cantidad de líneas de código que componen cada una de las liberaciones. Se presentan en forma separada las cantidades para la aplicación Android, para el componente de procesamiento de datos presente en servidor central, y para las bibliotecas compartidas.



	Aplicación Android	Procesamiento de datos	Bibliotecas compartidas	Total
Liberación 1	2839	3123	422	6384
Liberación 2	5622	6091	1802	13515
Liberación 3	7104	6867	1912	15883

Figura 5.2: Cantidad de líneas de código por liberación

5.3. Otras actividades realizadas

Se presentan a continuación actividades realizadas dentro del marco del proyecto no mencionadas hasta el momento. Estas actividades no están relacionadas directamente al desarrollo del sistema, por lo que se optó por describirlas por separado.

5.3.1. Presentación a representantes del proyecto CUDEN

El día 4 de octubre de 2012 se realizó una presentación a Javier Bustos, investigador de NicLabs (Chile) asociado al proyecto CUDEN. La valoración de la reunión fue positiva, el acta de la misma se encuentra en el Anexo D.

5.3.2. Presentación del proyecto en la octava edición de TRISTAN (*Triennal Symposium on transportation analysis*)

TRISTAN [35] es una conferencia internacional que provee un foro de alta calidad para la presentación de modelos matemáticos, metodologías, y resultados computacionales, y para el intercambio de ideas y discusiones científicas sobre aplicaciones y tecnologías avanzadas aplicadas al transporte. Se han realizado siete instancias de esta conferencia, y está programado que la próxima se realice en Chile en junio de este año.

El día 2 de noviembre de 2012 se realizó la postulación del proyecto a la octava edición de esta conferencia. Esta postulación consistió en la entrega de un *Extended Abstract* (adjunto en el Anexo G), la cual fue aceptada por la organización de la conferencia. Este proyecto será presentado en junio de este año en la octava conferencia TRISTAN.

5.3.3. Presentación a estudiantes de Redes de Computadoras

El día 15 de noviembre de 2012 se realizó una presentación a estudiantes de la asignatura Redes de Computadoras donde se presentó la aplicación y el proyecto, y se los motivó a que aportaran compartiendo sus datos cuando se encontrasen usando el transporte público. Esta presentación coincidió con el fin de la cuarta iteración, y por lo tanto con la segunda liberación.

Luego de la presentación se recibieron ideas por parte de los propios estudiantes sobre posibles mejoras, y se hizo también un pequeño debate sobre las posibilidades y desafíos encontrados en el desarrollo de esta aplicación que quedan como posibles trabajos futuros.

El material usado en la presentación se encuentra en el directorio Material/-Presentaciones del DVD de la entrega. El material consta de las diapositivas de la

presentación («20121115_Presentacion_Redes.ppt») y un video con una demostración de la aplicación («20121115_Demo.3gp»).

5.3.4. Presentación en *workshop* de CUDEN

Los días 4 y 5 de abril de 2013 se realizó en Montevideo un *workshop* de CUDEN, al cual asistieron representantes de Chile, Francia, Brasil y Uruguay. Se presentaron los avances de los proyectos de investigación realizados por los socios de CUDEN, y los próximos pasos a seguir.

Dentro de este encuentro se hizo una breve presentación de este trabajo de grado. Luego de la presentación, se produjo un intercambio de ideas entre los presentes. Los temas de mayor discusión fueron la escalabilidad del sistema para un uso masivo, así como la importancia de la privacidad de los usuarios.

Capítulo 6

Pruebas

Se presentan en este capítulo algunos de los resultados prácticos obtenidos en el uso de la aplicación. En primer lugar, se muestran datos obtenidos a partir de las publicaciones de los usuarios. Luego, se muestran resultados de la ejecución de la heurísticas de descenso del ómnibus.

A partir de las pruebas realizadas por los usuarios, se lograron registrar aproximadamente 56.000 posiciones de ómnibus. Algunas de éstas fueron consideradas erróneas, probablemente causadas por usos incorrectos de la aplicación, por ejemplo, usuarios que declararon estar sobre un ómnibus sin que ésto fuera cierto.

Se identificaron correctamente las posiciones de más de 375 ómnibus. Se presentan a continuación algunos recorridos almacenados, caracterizando diversas situaciones encontradas. Estas ejecuciones muestran cómo la aplicación se comportó en escenarios reales, siendo ejecutadas por diferentes dispositivos.

Las imágenes mostradas a continuación fueron obtenidas desde la interfaz web de administración incluida en el servidor central.

6.1. Evaluación de los datos obtenidos

El ómnibus correspondiente a la traza mostrada en la Figura 6.1 pertenecía a la línea 142 - Punta Gorda, y se registró el día 1 de febrero de 2012 entre las 18:25 y las 19:14.

En dicha figura se muestra el registro de una ejecución en la cual el sistema supo la posición del ómnibus durante todo su trayecto, con buena precisión. Esto permitió que todos los usuarios que consultaron la posición de ómnibus de esta línea en ese momento, visualizaran correctamente a ese ómnibus a lo largo de casi todo

Figura 6.1: GPS funcionando correctamente

su recorrido. Es importante observar también que en esta traza el usuario comenzó a compartir su ubicación en el inicio del viaje (el viaje inicia prácticamente en el origen de la línea), y deja de compartir en el momento en que desciende del ómnibus.

Figura 6.2: GPS funcionando en pequeños tramos

La traza de la Figura 6.2 se corresponde a un ómnibus de la línea 21 - Parque Roosevelt, y se registró el día 5 de diciembre de 2012 las 8:13 y las 8:45.

En esta figura se muestra el caso opuesto al anterior. En este caso, la recepción de GPS por parte del dispositivo no fue buena, registrándose solamente algunos tramos del recorrido. Si un usuario hubiera consultado la posición de ómnibus de esta línea, hubiera recibido información imprecisa, con actualizaciones menos frecuentes. En algunos casos, hubiera visto puntos alejados del recorrido real del ómnibus, como el punto resaltado en la figura.

La experiencia indica que los problemas observados en este caso pueden ser causados por varios motivos. Uno de ellos se debe a que el dispositivo no logró recibir la señal de los satélites de GPS durante buena parte del recorrido. Esto puede ser provocado por problemas geográficos o por otros obstáculos que impiden la llegada de la señal al dispositivo, por ejemplo, zonas dentro de la ciudad rodeadas de edificios, donde es más difícil que el GPS funcione correctamente.

En casos donde la aplicación no logra obtener una posición de GPS, continúa publicando la posición inferida a partir de la intensidad con que se recibe la señal de antenas celulares, o de Wi-Fi. Este tipo de ubicación tiene una precisión mucho menor, lo que podría explicar el punto resaltado en la figura.

Por otro lado, hay una posible causa asociada a la arquitectura propuesta por el sistema operativo Android para procesos que ejecutan en segundo plano. El sistema operativo, en situaciones de baja memoria, puede elegir detener servicios que no sean vitales para su funcionamiento. Cuando el sistema operativo vuelve a poseer recursos suficientes para la ejecución del proceso, lo reinicia. En este caso el GPS del dispositivo podría estar funcionando correctamente, pero al no estar activo el mecanismo de publicación de las posiciones obtenidas el servidor central no recibe esta información.

Figura 6.3: GPS funcionando correctamente a partir de un punto del recorrido.

En la Figura 6.3 se muestra un caso muy frecuente. En esta ejecución, el GPS tuvo una recepción muy buena a partir de un cierto punto del recorrido (salvo una interrupción puntual), pero tuvo una mala recepción en el inicio del mismo, como se puede ver en los tres puntos presentes en el lado derecho de la imagen.

Esta situación ocurre a menudo, y está asociada al tiempo que transcurre desde el inicio del viaje hasta que el dispositivo obtiene la primer ubicación usando GPS. Hasta ese momento, la aplicación muestra las posiciones obtenidas a través de las otras fuentes de posicionamiento provistas por Android.

Luego de obtener la primer posición, la recepción de GPS suele mantenerse hasta el fin del viaje, o hasta que alguno de los factores indicados anteriormente provoque su interrupción.

Otro problema relativo al sensor de GPS fue encontrado luego de la tercer liberación. En esta liberación, se dio un caso en el cual un usuario estaba reportando posiciones para un ómnibus, cuya fecha estaba un día adelantada. Se encontró que la causa de este problema es una falla ya conocida en algunos módulos de GPS

usados por teléfonos Samsung. Esta falla provocaba que en los reportes de posición del GPS, la fecha estuviera un día adelantada. La aplicación toma esta hora como la hora actual. Por lo tanto, la aplicación estaba enviando al servidor información incorrecta sobre la hora en que había sido tomada la posición del ómnibus, generando errores. Una vez diagnosticado el problema, se incorporaron mecanismos en el sistema para que esta falla no afecte su funcionamiento.

6.2. Heurística de descenso de ómnibus

En esta sección se presentan algunos casos que ilustran los resultados obtenidos relativos a la heurística de descenso de ómnibus incluida en la última versión de la aplicación. Como se explicó en el Capítulo 4, para esta heurística se tomaron en cuenta los datos obtenidos hasta ese momento, tanto para el diseño de la heurística como para su calibración y verificación.

A continuación se muestran algunos casos encontrados en la simulación realizada para verificar la heurística. Estos datos fueron recabados con la primer versión de la aplicación, y se usaron como entrada al algoritmo de la heurística, para observar en qué momento la heurística detecta que el usuario descendió del ómnibus.

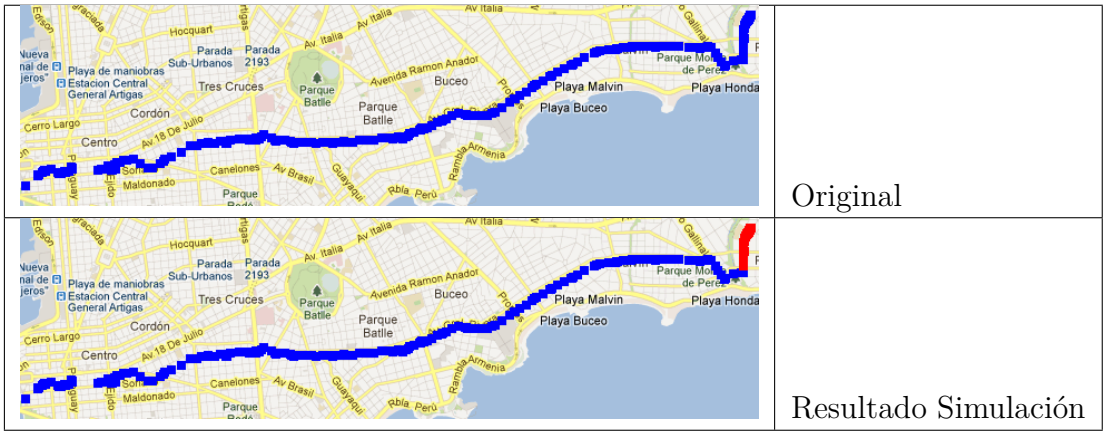


Figura 6.4: Detección correcta de descenso de ómnibus

En la Figura 6.4 se muestra un caso exitoso de la detección de descenso de ómnibus. La traza se corresponde con un ómnibus de la línea 142 - Punta Gorda, se registró el día 10 de enero del presente año entre las 18:40 y las 19:29.

En la traza original, puede verse que el usuario descendió del ómnibus pero no indicó que había descendido. En la simulación, puede verse que casi inmediatamente la aplicación detectó que el usuario no se encontraba en un ómnibus, por lo que este

caso puede considerarse exitoso. En la nueva versión de la aplicación, si se repitiera este caso, luego de publicar la primer posición indicada en rojo, el dispositivo dejaría de compartir su posición.

Figura 6.5: Detección incorrecta de descenso de ómnibus

Como ya se mencionó en la Sección 4.7.2.2, la heurística de descenso de ómnibus genera falsos positivos en algunos casos, como es el caso que se muestra en la Figura 6.5. La medida original muestra un recorrido similar al caso anterior, pero el resultado de la simulación muestra que la heurística detectó un descenso del usuario cerca del inicio del viaje. En este caso, a no ser que el usuario se percatase de la situación y comenzara a compartir la ubicación nuevamente, el dispositivo hubiera dejado de reportar la posición mucho antes de el fin del viaje.

Si bien en casos de este tipo la heurística provoca que se pierda mucha información, la cantidad de estos casos es poco significativa en el total.

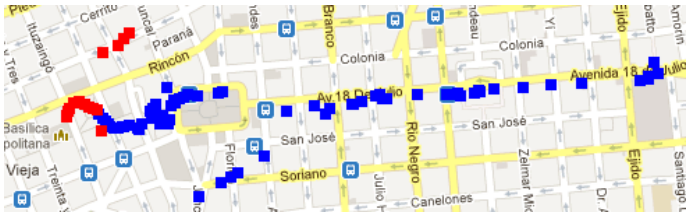


Figura 6.6: Detección tardía de descenso

En la Figura 6.6 se muestra otro caso en el que la heurística no tuvo el resultado esperado. En este caso,el usuario descendió del ómnibus en la terminal ubicada en la plaza independencia, y continuó a pie. La aplicación detectó que el usuario había descendido del ómnibus, pero esto ocurrió mucho más tarde de lo deseable. Si bien se

obtuvo finalmente una detección automática del descenso, mientras esto no ocurrió el sistema continuó mostrando al ómnibus como activo, moviéndose según la caminata del usuario.

6.3. Resumen

Se mostraron en este capítulo los resultados obtenidos en el uso de la aplicación desarrollada. En general, se verifica que la aplicación se comporta correctamente, cumpliendo con el objetivo planteado, aunque con problemas. Estos problemas, se originan en su totalidad en las limitaciones impuestas por el sensor de GPS presente en los dispositivos.

Más allá de los problemas presentados, se verificó que el sistema planteado es factible técnicamente, ya que la aplicación funciona correctamente en la mayoría de los casos.

Por otro lado, se mostraron casos de prueba tomados para la heurística de descenso de ómnibus, presentando representantes de las situaciones encontradas. Si bien existen casos donde la heurística falla, donde la heurística podría ser mejorada, se verifica que es factible construir una heurística de descenso de ómnibus basada en las posiciones de GPS.

Capítulo 7

Resultados obtenidos y conclusiones

Se presentan a continuación las conclusiones extraídas a partir del trabajo presentado, tanto las relativas al sistema construido, como relativas al uso de sensores en la plataforma Android.

7.1. Sistema construido

En este proyecto se implementó de forma satisfactoria un sistema colaborativo para dispositivos Android, que permite ubicar los ómnibus del sistema de transporte de una ciudad en tiempo real. Este sistema tiene dos componentes principales, una aplicación de usuario final, instalable en su dispositivo, y un servidor, que solo puede ser accedido directamente por un administrador y es responsable de centralizar la información existente en el sistema y de su distribución. Este sistema cubre todos los requerimientos descriptos en la Sección 4.1.

Se describe a continuación cómo se completaron los objetivos específicos del presente proyecto.

7.1.1. Intercambio de posiciones de ómnibus entre usuarios

Se implementó un mecanismo mediante el cual un usuario puede indicar al sistema que se encuentra en un ómnibus, y a partir de ese momento su dispositivo compartirá su ubicación, con lo que el sistema la conoce y pone a disposición del grupo de usuarios que la requiera.

7.1.2. Intercambio de información adicional

Se implementó un mecanismo, independiente al mencionado en el punto anterior, que los usuarios pueden usar para intercambiar información dentro del sistema. Este mecanismo permite el intercambio de mensajes de texto en un *chat* grupal. Podrían usar este sistema por ejemplo para intercambiar mensajes relativos a desvíos o retrasos de una línea específica.

7.1.3. Visualización de ómnibus en tiempo real

Se implementó un mapa que muestra en forma centralizada toda la información provista por los dispositivos de los usuarios. Este mapa puede ser accedido desde la consola web de administración del sistema.

También se implementó un mapa para los dispositivos que muestra los ómnibus de un conjunto acotado de líneas, indicado por el usuario.

La actualización de las posiciones conocidas de ómnibus por parte del sistema, se realizan cada cinco segundos. Esto implica que los usuarios visualizan en la aplicación, las posiciones reales que tenían los ómnibus cinco segundos atrás.

Esta funcionalidad fue probada en el caso de la ciudad de Montevideo con resultado exitoso, que puede ser visto en el Capítulo 6.

7.1.4. Funcionamiento en varias áreas

La solución diseñada permite la ejecución del sistema en varias áreas de forma simultánea. Esto se vio reflejado tanto en el modelo de datos como en las comunicaciones, ya que todo es relativo a un área específica.

Sin embargo, las liberaciones realizadas fueron en Montevideo, por lo que no se probó intensivamente esta funcionalidad.

7.1.5. Generación de alertas de proximidad

Este objetivo se cumplió parcialmente. Se implementó en la aplicación Android un mecanismo que permite al usuario indicar en qué parada desea descender del ómnibus. Una alarma de proximidad se activa a una cierta distancia de la parada, y alerta al usuario sobre la situación.

No se implementó en este proyecto una alarma de proximidad para el caso en que el usuario desea subirse a un ómnibus. Esta alarma alertaría al usuario cuando el ómnibus se encuentra próximo a la parada donde pretende subirse.

7.1.6. Detección de eventos

Se realizó una prueba de concepto de posibles mecanismos de detección de eventos relevantes al sistema, usando los datos provistos por los sensores del dispositivo.

Se implementó la detección de los eventos: llegada a una parada de ómnibus, inicio de viaje y descenso de un ómnibus, como se describe en la Sección 4.7.

Se verificó que es viable detectar eventos usando estos sensores, aunque las heurísticas desarrolladas no cubren todos los posibles casos, por lo que podrían ser mejoradas en trabajos futuros.

7.2. Uso de sensores

Uno de los objetivos propuestos era la investigación del uso de sensores en la plataforma Android.

En el transcurso del proyecto se investigaron las posibilidades en Android para usar los sensores de GPS y acelerómetro. También se investigó la posibilidad del uso de antenas como posibles sensores, tales como la antena GSM/3G o Wi-Fi.

Se verificó que Android es un sistema que permite fácilmente el acceso a la información provista por estos sensores, y es una buena plataforma para desarrollar aplicaciones que requieran fuerte uso de éstos.

Cabe notar sin embargo, que al ser los sensores dispositivos de hardware, los resultados obtenidos dependen fuertemente del modelo del dispositivo. Por ejemplo, pueden haber sensores con fallas que indiquen datos erróneos, o sensores que se comportan de formas levemente diferentes según el modelo y el fabricante.

Como segunda desventaja, es importante notar que si bien los sensores pueden usarse sin mayores problemas, aplicaciones que hagan un uso intensivo de estos provocarán un gran gasto de batería.

7.3. Conclusiones

Se concluye que el sistema completo, tal como se describió en el Capítulo 3, es técnicamente factible y podría ser instalado para un uso real. Como se mencionó previamente, el principal problema se da cuando los usuarios usan la aplicación para consultar la posición de los ómnibus y no obtienen la información esperada. Esto provoca que se vean desmotivados en su uso y por esta razón, es necesario incorporar al sistema un atractivo inicial que lleve a un uso masivo de la aplicación.

La falta de información al momento de consultar es principalmente causada por el enfoque completamente colaborativo tomado en este proyecto, alineado con sus objetivos iniciales. Este enfoque implica que los usuarios proveen tanto la información en tiempo real sobre los ómnibus como la información estática, como ubicación de paradas y recorridos de líneas de ómnibus.

Una posible mejora sería incorporar inicialmente al sistema información estática en las ciudades donde se pueda obtener esta información. Para el resto, los usuarios deberían aportar estos datos manualmente. En el caso de Montevideo la información estática está públicamente disponible en el sitio web de la Intendencia de Montevideo [1].

Incorporando esta información se contaría con más y mejores datos como entrada para las heurísticas de detección de eventos y se podrían agregar nuevas funcionalidades atractivas para el usuario. Por ejemplo, se podría mostrar la ubicación teórica de los ómnibus según sus recorridos y horarios, en el caso de que no exista información en tiempo real provista por los usuarios.

El sistema seguiría teniendo un carácter colaborativo, puesto que los usuarios continuarían compartiendo información para la construcción del mapa en tiempo real de la flota de transporte, así como otro tipo de información usando el componente social.

Otro enfoque completamente distinto sería abandonar el principio de la colaboración entre usuarios, presente en los objetivos iniciales del proyecto, e incorporar, además de la información estática, información en tiempo real sobre la flota de transporte provista por empresas de transporte o el gobierno. Esto permitiría contar con información confiable y precisa sobre la flota de transporte en todo momento.

Si bien este enfoque tendría excelentes resultados, estaría restringido a un conjunto menor de ciudades, aquellas que cuentan con una infraestructura que provea la ubicación en tiempo real de la flota de transporte. Dada esta restricción, el enfoque colaborativo sigue siendo interesante puesto que se puede aplicar a cualquier ciudad.

[1] http://www.montevideo.gub.uy/node/13757

Capítulo 8

Trabajo futuro

En este capítulo se presentan mejoras posibles al sistema creado así como extensiones que permitirían proveer una mejor experiencia al usuario durante la ejecución de la aplicación.

8.1. Explotación de los datos obtenidos

Podrían crearse algoritmos que trabajen sobre los datos recolectados por el sistema, que los analicen para extraer información útil. Por ejemplo, podría inferirse a partir de un conjunto de trazas (con imprecisiones, y «ruido» en general) cuál es el recorrido real para una línea de ómnibus determinada, cuáles son sus horarios, o cuál es el tiempo que demora en promedio entre dos puntos cualesquiera del recorrido.

Ésto podría ser útil para ser luego incorporado en la aplicación, apoyando funcionalidades como:

- Determinación del tiempo de espera para un ómnibus particular: dada la línea a la que pertenece y la distancia de éste a la parada donde se encuentra el usuario.

- Predicción de posiciones de ómnibus sobre los que ya no se tienen datos: sabiendo la última posición conocida para un ómnibus, estimar la posición actual.

- Predicción de posición de ómnibus basada en horarios: interpolar dónde debería haber un ómnibus en el momento actual a partir de los horarios conocidos para la línea correspondiente.

8.2. Componente social

La aplicación construida dentro de este proyecto incluye un componente social, el cual se limita a salas de conversaciones para cada línea de ómnibus y para el área a la que pertenecen, en donde los usuarios pueden interactuar entre ellos.

En versiones futuras, la aplicación podría incorporar otros mecanismos de interacción entre los usuarios, a través de otros medios. Sería interesante extender la aplicación para, por ejemplo, incorporar fotos o videos dentro del contenido compartido por los usuarios.

Se podrían explotar también las comunidades generadas dentro del sistema para mejorar la construcción colaborativa de los datos. Por ejemplo, se podrían agregar opciones para denunciar información incorrecta, o implementar sistemas de votación para que la aplicación acepte información de ciertos usuarios como correcta.

Otra opción, es brindar a los usuarios la posibilidad de enviar información extra junto con la publicación de su posición, para indicar el estado en el que está el ómnibus. Por ejemplo, podrían advertir que un ómnibus se encuentra desviado, que se ha averiado, o que se encuentra lleno. Esta información se podría categorizar para representar el estado del ómnibus. Los usuarios que consulten por estos ómnibus además de ver su posición también conocerían su estado.

8.3. Perfil de usuario

En la versión actual del sistema se almacena muy poca información sobre las costumbres de los usuarios. En un trabajo futuro, podría analizarse qué información es interesante almacenar sobre un usuario a los efectos de aprender su comportamiento, y así más adelante proveerle información relevante según estos comportamientos.

Por ejemplo, si se detecta un patrón similar a «Todos los días de Lunes a Viernes, el usuario toma un ómnibus de la línea Y a las nueve de la mañana» la aplicación podría automáticamente mostrar la posición de los ómnibus de dicha línea un tiempo antes de esa hora, para que el usuario vea por dónde vienen estos ómnibus.

Esta funcionalidad tiene el desafío de encontrar una implementación que maneje la privacidad del usuario correctamente.

8.4. Interfaz

En el presente trabajo, el desarrollo estuvo muy centrado en la infraestructura necesaria para el funcionamiento de la aplicación, y el manejo de sensores en el dispositivo. Un trabajo futuro podría mejorar la aplicación en términos de su interfaz gráfica, investigando patrones de diseño para dispositivos inteligentes, e implementando a partir de éstos una interfaz que mejore la usabilidad de la aplicación.

8.5. Aplicación web del servidor

Este trabajo incluyó el desarrollo de un una aplicación de administración en el servidor, con funcionalidades básicas. Se cuenta con un mapa que permite ver en tiempo real la posición de los ómnibus, junto con registros de la información enviada entre los dispositivos conectados y el servidor.

Esta aplicación podría ser mejorada para incluir información estadística, para poder ver la actividad por línea de ómnibus, por área, por paradas, etc.

La interfaz actual está centrada en un usuario administrador, pero podría ser interesante dejar algunas de estas funcionalidades públicas para que cualquier persona pudiera consultar la información recolectada dentro del sistema.

8.6. Heurísticas

Las heurísticas de este proyecto fueron desarrolladas como prueba de concepto de lo que se puede lograr con la información obtenida de los sensores en un dispositivo Android. En proyectos futuros, estas heurísticas podrían mejorarse para detectar correctamente una mayor cantidad de casos.

Estas heurísticas podrían ser una mejora de las actuales, o basarse en conceptos completamente distintos a los actuales.

Por ejemplo, para casos de ómnibus con redes Wi-Fi internas, podría definirse una heurística que detectase la subida a un ómnibus si se capta una de estas redes durante un cierto período de tiempo.

Otra posibilidad, es buscar alternativas similares a la desarrollada en el trabajo «*How Long to Wait?: Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing*» [13], en el cual se detecta el momento en el que el usuario sube a un ómnibus, escuchando a través del micrófono del dispositivo, el sonido emitido por la máquina expendedora de boletos.

Desde otro punto de vista, podrían investigarse en trabajos futuros formas de implementar heurísticas que consuman menos batería, sin sacrificar eficacia en la detección.

8.7. Transmisión de datos al servidor

Si el usuario no tiene conexión de datos (3G) se podría implementar un protocolo oportunista que vaya guardando la traza de posiciones del ómnibus y la envíe la próxima vez que tenga conexión. Ésto serviría para obtener datos estadísticos de tiempos y recorridos, no para ver en tiempo real por dónde viene el ómnibus.

También se podrían desarrollar mecanismos para comunicar mediante Bluetooth (u otros medios) con algún otro dispositivo que se encuentre dentro del mismo ómnibus y que éste, que sí tiene conexión de datos, envíe los datos por él.

8.8. Permitir el desarrollo de servicios sobre la infraestructura

Exponer servicios para aplicaciones externas de forma que, por ejemplo, éstas puedan consultar la información de la posición actual de los ómnibus.

Una aplicación de terceros podría acceder a los datos del sistema y realizar procesamiento sobre éstos con algún fin particular, como realizar explotación de los datos como se menciona en la Sección 8.1.

Anexo A

Distribución de mensajes en Android

Como se mencionó en la Subsección 3.1.1, uno de los principales problemas a resolver es la entrega eficiente de mensajes a grupos de usuarios. Se buscó una solución que permitiese el manejo de grupo de usuarios de una forma sencilla, tanto su creación en forma dinámica como la entrega de mensajes a los integrantes de los grupos.

En este anexo se presentan las alternativas evaluadas, cuál es el mecanismo seleccionado y las razones que llevaron a esta decisión.

A.1. Twitter

El primer método evaluado fue Twitter, tras observar que el comportamiento asociado a los *hashtags* de esta plataforma podría modelar los grupos de usuarios.

En Twitter, los usuarios puede subscribirse a uno o varios *hashtags.* Cuando un usuario se subscribe a un *hashtag* específico, Twitter le envía una notificación por cada *tweet* que contenga dicho *hashtag.* La definición de un *hashtag* es dinámica, es creado cuando se envía el primer mensaje usándolo.

La idea de usar los *hashtags* para modelar los grupos de usuarios consiste en asociar a cada línea de ómnibus un *hashtag* mediante una traducción automática. Cuando un usuario selecciona una línea de ómnibus para consultar la posición de los ómnibus, internamente la aplicación se subscribiría al *hashtag* de la línea. El servidor publicaría en este mismo *hashtag* las últimas posiciones conocidas, evitando distribuir esta información a cada dispositivo.

Con esto, se lograría bajar la carga del servidor, ya que publicaría la información una única vez, y Twitter se encargaría de su distribución a los dispositivos.

Twitter posee millones de usuarios en todo el planeta y soporta una gran carga con muy buen rendimiento, lo cual aseguraría no contar con problemas en la distribución de los mensajes. Se disminuiría el procesamiento del servidor, bajando sus requerimientos de hardware.

A.1.1. Prototipo implementado

Debido al gran potencial de Twitter como solución para la distribución de mensajería se realizó un prototipo intentando mitigar de forma temprana problemas en la arquitectura, la integración con Android o limitaciones de Twitter.

Se encontró la biblioteca Twitter4j[15] una biblioteca no oficial hecha en Java de la API de Twitter, la cual tiene encapsulado el consumo de los servicios brindados por Twitter facilitando la integración con un proyecto Java.

Luego se investigó el protocolo de autenticación OAuth de Twitter[16], y se realizaron prototipos utilizando las bibliotecas API REST[17] y Stream[18] de Twitter. El servicio de autenticación no presenta mayores dificultades y es similar al de otros servidores como Facebook o Google.

Usando la API Stream se obtuvieron resultados muy alentadores, ya que tras la publicación de un mensaje dentro de un cierto *hashtag*, en pocos segundos este mensaje llegaba correctamente a todos los clientes subscriptos a dicho *hashtag*.

Sin embargo, se encontró que Twitter tiene un límite de mil *tweets* por usuario por día[19], con lo cual es inviable implementar la solución de transporte sobre esta plataforma, ya que es esperable que se supere esta cota de mensajes por día.

A.2. Cloud to device messaging (C2DM)

C2DM es un servicio gratuito provisto por Google, que simplifica el envío de mensajes desde un servidor a aplicaciones instaladas en dispositivos con Android [20]. Este servicio está disponible sólo en versiones de Android posteriores a la 2.2.

C2DM no soporta enviar mensajes a más de un dispositivo simultáneamente. Esto implica que si el servidor debe enviar mensajes a un grupo de usuarios, debe hacerlo de forma individual para cada uno de los integrantes[21]. Otra limitación encontrada del servicio es que tiene cotas de envío por dispositivo y por aplicación, similar a la limitación que tiene Twitter.

A.3. Message Queue Telemetry Transport (MQTT)

MQTT[27] es un protocolo desarrollado por IBM en 1999, que a primera vista pareció estar muy bien adaptado a las necesidades de la aplicación desarrollada. Dentro de su documentación es descripto como: *"MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium."*[24]

Este protocolo está basado en el uso de *topics* a los que los clientes se subscriben. Por ejemplo, podría haber un *topic* para cada línea de ómnibus, donde se subscribirían todos los clientes que desean recibir actualizaciones sobre las instancias conocidas de ómnibus de esa línea.

El actor principal de este protocolo es el *message broker*, que es quien mantiene la información sobre los *topics* (los *topics* se crean dinámicamente en tiempo de ejecución), y cuáles dispositivos están registrados en cada *topic*. El servidor se conecta a este *message broker* para enviar mensajes a un determinado *topic* (sin saber si existen clientes escuchando este), y los dispositivos se registran a los *topics* de su interés. El *message broker* se encarga de en el momento de recibir un mensaje del servidor enviarlo a los clientes que corresponda. Los dispositivos podrían enviar mensajes al servidor usando el mismo mecanismo, usando *topics* a los que esté subscripto el servidor. A los efectos del *message broker*, el servidor es un cliente con las mismas características que los dispositivos.

Este protocolo ya es usado por otras aplicaciones Android, existen mucho material en Internet sobre cómo integrarlo a aplicaciones hechas en este sistema operativo[26, 28]. Se implementó un prototipo sencillo en Android para validar que el sistema podía ser utilizado para nuestros objetivos.

Existen *message brokers* públicos (no deberían usarse para producción), o pueden también instalarse localmente. El servidor local usado fue Mosquitto[25].

Este protocolo es usado por Facebook en su aplicación de mensajería instantánea para Android.[29]

A.4. Deacon

Deacon[30] es un software gratuito y de código abierto que implementa un cliente del servidor web Meteor[31]. Dicho servidor corre sobre Linux y está implementado en Perl. Corre como un demonio que tiene 2 procesos, uno que escucha un puerto para los controladores de eventos y otro por suscriptores de canales. Los controladores de eventos, son clientes que se conectan al puerto de control del servidor y utilizan los comandos del protocolo Meteor para insertar eventos en los canales, como por ejemplo "ADDMESSAGE channelName messagetext" para enviar un determinado mensaje en un canal específico. Los controladores de eventos también pueden consultar el estado de dichos canales. Los suscriptores son clientes que se conectan al puerto de suscripción y utilizan pedidos HTTP para solicitar la suscripción a uno o varios canales.

Deacon es una biblioteca de notificaciones *push* implementada puramente en Java. Puede usarse tanto en ambientes Android como en ambientes tradicionales de escritorio. La ventaja de usar Deacon es que la interacción con Meteor se realiza en un nivel más alto de abstracción, encapsulando las llamadas HTTP.

A.5. Comparación

Como se vio anteriormente, el mecanismo basado en *hashtags* de Twitter no es viable como solución al problema por su limitación de 1000 *tweets* por día. Por otra parte, C2DM tampoco es una solución válida dado que, al igual que Twitter, posee cotas de uso, y además no ofrece facilidades para el manejo de grupos de usuarios.

Por lo tanto, se realizó una comparación únicamente entre MQTT y Deacon, llegando a los siguientes resultados:

- Ambas soluciones descriptas usan un servidor de *message broker*: en el caso de MQTT se puede elegir hacerlo con Mosquitto (o desarrollar un servidor que implemente el protocolo de MQTT), y en el caso de Deacon se usa el Meteor Server.

- MQTT está pensando para dispositivos con poco ancho de banda y que usan redes no confiables, mientras que Meteor Server está pensado para aplicaciones web.

- En ambos casos, existen bibliotecas que actúan como clientes (suscriptores).

- Existen implementaciones en la industria de aplicaciones Android usando MQTT, mientras que para Meteor Server no se logró encontrar casos de éxito.

A.6. Mecanismo seleccionado

Tomando como base las alternativas mostradas en la sección anterior, se decidió usar MQTT con Mosquitto. La razón principal por la que se tomó esta decisión es que MQTT fue diseñado para ser un mecanismo de transporte de mensajería liviano, en escenarios con dispositivos con poco ancho de banda y redes no confiables, lo cual coincide exactamente con la situación en que se van a encontrar los clientes que usen la aplicación. Además, la solución es viable en términos de escalabilidad ya que existen aplicaciones basadas en este protocolo con altos niveles de carga.

A.7. GCM

Luego de seleccionar el mecanismo de entrega de mensajes, durante la implementación de la solución, C2DM fue dado de baja por Google, siendo sustituido por Google Cloud Messaging (GCM)[22]. Dado el avance del proyecto, modificar la decisión en ese momento era poco viable.

De todas formas se evaluó la nueva alternativa, y se encontró que en este nuevo servicio se levantan las dos restricciones menores encontradas en C2DM. GCM no posee cotas de cantidad de mensajes a enviar, y permite enviar mensajes a varios dispositivos con una única llamada. Si bien el poder enviar el mismo mensaje a varios dispositivos con una única llamada es positivo en términos de uso de ancho de banda del servidor, los dispositivos deben ser enumerados en la llamada. Por lo tanto, el servidor debería mantener actualizada la composición de los grupos de usuarios. Dada esta restricción, el mecanismo seleccionado MQTT sigue siendo una mejor solución para el problema ya que posee un mejor manejo de grupos de usuarios.

Anexo B

Evolución de la sintaxis del protocolo

Durante el desarrollo del proyecto el protocolo tuvo modificaciones. En general, estas modificaciones estuvieron orientadas a la mejora en su rendimiento. En este anexo se describe los cambios que sufrió el protocolo, junto con los efectos de realizarlos.

B.1. Versiones del protocolo

Como se mencionó en la Sección 4.6.3.4, el protocolo fue desarrollado en dos etapas. A continuación, se muestra la sintaxis de las versiones resultantes de estas etapas.

B.1.1. Versión 0.1

Esta versión fue desarrollada en el inicio del proyecto, cuando el foco era conseguir implementar las funcionalidades básicas. Se priorizó en ese momento el modelado de la red de transporte, el aprendizaje dentro de la plataforma Android, y validar que fuera factible la arquitectura. Por estas razones se optó por un mecanismo de serialización sencillo de implementar, aunque fuera evidente que no era óptimo (tanto en ancho de banda consumido como en tiempo de procesamiento) ni extensible.

B.1.1.1. Sintaxis genérica de los mensajes

Todos los mensajes siguen un mismo formato, sin importar el *topic* en que sean publicados, el sentido en el que se publiquen, o el propósito del mensaje. Este formato establece que:

1. Todos los mensajes consisten de una cadena de caracteres *human readable*

2. Todos los mensajes cumplen con la siguiente estructura mostrada en la Figura B.1 donde <Message_Type> es un identificador del tipo de mensaje siendo enviado (único por *topic*), y <Message_JSON_Format> es la serialización en JSON de un *datatype* con la información correspondiente al mensaje. La cantidad de este último componente depende del MessageType del mensaje.

```
MessageType=<Message_Type>(;<Message_JSON_Format>)*
```

Figura B.1: Sintaxis genérica de mensajes

Se eligió esta codificación por dos razones:

1. Los mensajes enviados son *human readable* para poder monitorear los *topics* usados y ver (y entender) el tráfico que circula por estos.

2. Los datos se serializan usando JSON por dos razones. La primera es que permite hacer cambios fácilmente en el protocolo en tiempo de prototipación, ya que se puede cambiar el contenido de los mensajes cambiando la clase serializada. La segunda es que la información serializada no ocupa mucho más espacio que la información original (comparada por ejemplo, a una serialización basada en XML).

B.1.1.2. Sintaxis según el tipo de mensaje

A continuación, se muestra la sintaxis concreta para cada tipo de mensaje definido en la Sección 4.6.3. Es decir, se mostrará cómo se instancian cada uno de los tipos de mensajes identificados en la sección anterior. En las siguientes secciones, "JSON(DataTypeClass)" será una forma de abreviar el concepto "cadena de caracteres resultado de serializar un objeto de clase DataTypeClass a JSON".

B.1.1.3. AreaListSolicitation

Mensajes de este tipo son enviados por un cliente cuando requiere que se le entregue la lista de áreas disponibles en el sistema. El mensaje no lleva parámetros, por lo que el mensaje tiene la forma indicada en la Figura B.2.

```
MessageType=AREA_LIST_SOLICITATION
```

Figura B.2: Formato de mensaje AreaListSolicitation

Al no llevar ningún parámetro, así serán todos los mensajes de este tipo enviados dentro de la aplicación.

B.1.1.4. BusListSolicitation

Mensajes de este tipo son enviados por un cliente cuando requiere que se le entregue la lista de líneas de ómnibus disponibles en el sistema para un área dada. El mensaje es enviado a un *topic* MQTT determinado por el área, por lo cual no es necesario enviar información específica sobre el área en el mensaje mismo. Por esta razón el mensaje no lleva parámetros, por lo que el mensaje tiene la forma indicada en la Figura B.3.

```
MessageType=BUS_LIST_SOLICITATION
```

Figura B.3: Formato de mensaje BusListSolicitation

Al no llevar ningún parámetro, así serán todos los mensajes de este tipo enviados dentro de la aplicación.

B.1.1.5. AreaListNotification

Mensajes de este tipo no tienen un contexto definido (es decir, devuelve información que es global al sistema). La definición de la codificación de este mensaje es el indicado en la Figura B.6.

```
MessageType=AREALISTNOTIFICATION;JSON(Set<AreaDataType>)
```

Figura B.4: Formato de mensaje AreaListNotification

Donde JSON(Set<AreaDataType>) es la serialización del conjunto que contiene todas las áreas definidas en el sistema. La Figura B.5 contiene un ejemplo en ejecución de un mensaje de tipo AreaListNotification.

```
MessageType=AREALISTNOTIFICATION;[{"id":2,"name":"Buenos
   Aires"},{"id":1,"name":"Montevideo"},{"id":3,"name":"Nueva
    York"}]
```

Figura B.5: Ejemplo de mensaje AreaListNotification

En este caso se indica que existen tres áreas, “Montevideo”, “Buenos Aires” y “Nueva York”, junto con los identificadores de estas áreas en la base de datos.

B.1.1.6. BusListNotification

Mensajes de este tipo son enviados dentro del contexto de un área particular. Como contenido, tendrá todas las líneas de ómnibus definidas para esta. La implementación de este mensaje tiene la forma mostrada en la Figura B.6.

```
MessageType=BUSLISTNOTIFICATION;JSON(Set<BusLineDataType>)
```

Figura B.6: Formato de mensaje BusListNotification

Donde JSON(Set<BusLineDataType>) es la serialización del conjunto que contiene todas las líneas de ómnibus definidas en el área. En la Figura B.7 se muestra un ejemplo de mensaje de tipo BusListNotification.

```
MessageType=BUS_LIST_NOTIFICATION;[{"id":1,"number":"117","
   destination":"PZA. INDEPENDENCIA","area":{"id":1,"name":"
   Montevideo"}},{"id":2,"number":"117","destination":"PTA.
   CARRETAS","area":{"id":1,"name":"Montevideo"}},{"id":3,"
   number":"60","destination":"PORTONES","area":{"id":1,"name
   ":"Montevideo"}},{"id":4,"number":"60","destination":"
   CIUDAD VIEJA","area":{"id":1,"name":"Montevideo"}},{"id
   ":5,"number":"D10","destination":"GEANT","area":{"id":1,"
   name":"Montevideo"}},{"id":6,"number":"D10","destination
   ":"CIUDAD VIEJA","area":{"id":1,"name":"Montevideo"}},{"id
   ":7,"number":"104","destination":"GEANT","area":{"id":1,"
   name":"Montevideo"}},{"id":8,"number":"104","destination
   ":"PZA. INDEPENDENCIA","area":{"id":1,"name":"Montevideo
   "}},{"id":10,"number":"64","destination":"PZA.
   INDEPENDENCIA","area":{"id":1,"name":"Montevideo"}},{"id
   ":12,"number":"64","destination":"CIUDAD VIEJA","area":{"
   id":1,"name":"Montevideo"}},{"id":13,"number":"64","
   destination":"PUENTE CARRASCO","area":{"id":1,"name":"
   Montevideo"}}]
```

Figura B.7: Ejemplo de mensaje BusListNotification

B.1.1.7. BusNotification

Mensajes de este tipo son enviados dentro del contexto de una línea de ómnibus particular. Para esta línea, se informa mediante esta notificación la posición de todos los ómnibus conocidos por el sistema que pertenecen a esa línea. El formato concreto de este mensaje es el mostrado en la Figura B.8.

```
MessageType=BUS_NOTIFICATION;JSON(Set<BusDataType>)
```

Figura B.8: Formato de mensaje BusNotification

Un ejemplo de mensaje de este tipo es presentado en la Figura B.9

```
MessageType=BUS_NOTIFICATION;[{"id":92,"busInformation":{"id
    ":1,"number":"117","destination":"PZA. INDEPENDENCIA","
    area":{"id":1,"name":"Montevideo"}},"lastSeenTime":"Aug
    5, 2012 7:15:21 PM","status":"","position":{"latitude
    ":-34.8894382,"longitude":-56.1005805,"accuracy
    ":2104.0},"active":true}]
```

Figura B.9: Ejemplo de mensaje BusNotification

B.1.1.8. UserBusNotification

Mensajes de este tipo son enviados por un dispositivo cuando quiere indicar al servidor que el usuario al que pertenece se encuentra en un ómnibus, junto con la posición de este ómnibus y el estado del mismo. El formato concreto de este mensaje es el mostrado en la Figura B.10.

```
MessageType=USER_BUS_NOTIFICATION;JSON(UserDataType);JSON(
    BusDataType)
```

Figura B.10: Formato de mensaje UserBusNotification

Y una posible instanciación de este mensaje es el presente en la Figura B.11.

```
MessageType=USER_BUS_NOTIFICATION;{"area":{"name":"Montevideo
    ","id":1},"email":"","temporalId":"7c1c7fde-bca1-4f02-b83d
    -a5133f78ba7b","id":0};{"status":"","busInformation":{"
    area":{"name":"Montevideo","id":1},"destination":"PZA.
    INDEPENDENCIA","number":"117","id":1},"position":{"
    accuracy":2104.0,"latitude":-33.3334447,"longitude
    ":-57.876338},"lastSeenTime":"Aug 5, 2012 7:23:57 PM","id
    ":0,"active":true}
```

Figura B.11: Ejemplo de mensaje UserBusNotification

En este mensaje es importante observar el uso del *ID temporal* como forma de identificar al usuario dentro de la aplicación.

B.1.2. Versión 1.0

La segunda versión del protocolo optimiza la primera, minimizando la cantidad de campos enviados en cada mensaje y creando una codificación propia basada en cadenas de bytes. Además, se considera la posibilidad de tener clientes con distintas versiones del protocolo en funcionamiento.

B.1.2.1. Análisis de datos enviados en mensajes

Se analizó la semántica de cada tipo de mensaje, identificando los campos estrictamente necesarios para cada uno.

B.1.2.2. AreaListSolicitation

Mensajes de este tipo son enviados por un cliente cuando requiere que se le entregue la lista de áreas disponibles en el sistema. El mensaje debe contener información acerca de:

- **Identificador**: usado por el servidor para saber quién es el cliente que envía el mensaje.

B.1.2.3. AreaListNotification

Mensajes de este tipo son enviados desde el servidor, cuando algún cliente requirió que se le devuelva la lista de áreas disponibles en el sistema. El mensaje debe contener información acerca de:

- **Identificador**: usado por el cliente para verificar que el mensaje haya sido enviado por el servidor.
- **Datos**: lista de áreas disponibles en el sistema.

B.1.2.4. BusListSolicitation

Mensajes de este tipo son enviados por un cliente cuando quiere recibir la lista de líneas de ómnibus disponibles en el sistema para un área dada. El mensaje es enviado a un *topic* MQTT determinado por el área, por lo cual no es necesario enviar información específica sobre el área en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador**: usado por el servidor para saber quién es el cliente que envía el mensaje.

B.1.2.5. BusListNotification

Mensajes de este tipo son enviados por el servidor a el (o los) cliente(s) que pidieron la lista de ómnibus para un área dada. Al igual que el tipo de mensaje BusListSolicitation, el mensaje es enviado a un *topic* MQTT determinado por el área, por lo cual no es necesario enviar información específica sobre el área en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador**: usado por el cliente para verificar que el mensaje haya sido enviado por el servidor.
- **Datos**: lista de líneas de ómnibus para un área dada.

B.1.2.6. UserBusNotification

Mensajes de este tipo son enviados por un dispositivo informando al servidor que el usuario se encuentra en un ómnibus. Contiene la posición del usuario y el estado del ómnibus percibido por el usuario. Dicho mensaje contiene información acerca de:

- **Identificador:** usado por el servidor para saber quién es el cliente que envía el mensaje.
- **Datos**: ómnibus del usuario, posición (latitud y longitud), precisión de la misma, *timestamp* indicando cuándo se tomó la posición, y el estado del ómnibus.

B.1.2.7. BusNotification

Mensajes de este tipo son enviados por el servidor informando sobre los ómnibus en circulación para una línea de ómnibus - destino particular, dentro de un área dada. El mensaje es enviado a un *topic* MQTT determinado por el área y la línea de ómnibus, por lo cual no es necesario enviar información específica sobre el área o la línea en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador**: usado por el cliente para verificar que el mensaje haya sido enviado por el servidor.

- **Datos**: lista de ómnibus en circulación, indicando para cada uno de ellos la posición (latitud y longitud), la precisión y un *timestamp* indicando la última vez que fueron vistos.

B.1.2.8. SocialChannelSolicitation

Mensajes de este tipo son enviados por un cliente cuando quiere recibir la lista de grupos sociales disponibles en el sistema para un área dada. El mensaje es enviado a un *topic* MQTT determinado por el área, por lo cual no es necesario enviar información específica sobre el área en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador:** usado por el servidor para saber quién es el cliente que envía el mensaje.

B.1.2.9. SocialChannelNotification

Mensajes de este tipo son enviados por el servidor informando sobre los grupos sociales disponibles para un área dada. El mensaje es enviado a un *topic* MQTT determinado por el área, por lo cual no es necesario enviar información específica sobre el área en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador**: usado por el cliente para verificar que el mensaje haya sido enviado por el servidor.
- **Datos**: lista de grupos sociales disponibles dentro del área, indicando para cada uno de ellos una descripción, *topic* MQTT donde publica el servidor, *topic* MQTT donde publica el cliente, tipo de grupo social (por ejemplo: ómnibus, línea de ómnibus, área).

B.1.2.10. SocialUserMessage

Mensajes de este tipo son enviados por un cliente cuando quiere enviar un mensaje a un determinado grupo social. El mensaje es enviado a un *topic* MQTT determinado por el grupo social, por lo cual no es necesario enviar información específica sobre dicho grupo en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador:** usado por el servidor para saber quién es el cliente que envía el mensaje.

- **Datos**: mensaje del usuario para el grupo social en cuestión.

B.1.2.11. SocialServerMessage

Mensajes de este tipo son enviados por el servidor a los clientes que están en un grupo social dado. El mensaje es enviado a un *topic* MQTT determinado por el grupo social, por lo cual no es necesario enviar información específica sobre dicho grupo en el mensaje mismo. Dicho mensaje contiene información acerca de:

- **Identificador**: usado por el cliente para verificar que el mensaje haya sido enviado por el servidor.
- **Datos**: identificador del mensaje (generado por el servidor), sobrenombre del usuario emisor, *timestamp* indicando cuándo el mensaje fue enviado, y el mensaje escrito por el usuario.

B.1.2.12. UserBusStopNotification

Mensajes de este tipo son enviados por un dispositivo informando al servidor que el usuario se encuentra en una parada de ómnibus. Contiene la posición del usuario y (opcionalmente) una descripción de la parada por parte el usuario. Dicho mensaje contiene información acerca de:

- **Identificador:** usado por el servidor para saber quién es el cliente que envía el mensaje.
- **Datos**: posición (latitud y longitud), precisión de la misma y una descripción de la parada.

B.1.2.13. Tipos de datos básicos

Luego del análisis de los datos enviados en mensajes, se identificaron los tipos de datos básicos involucrados en el protocolo, y se trabajó en la definición de una codificación estándar para los mismos, tanto para simplificar el diseño del protocolo como la codificación y decodificación de todos los mensajes. El objetivo de la codificación es obtener la representación en *arrays* de bytes para cada tipo de datos, para luego ser incluidos dentro de los mensajes enviados a través de MQTT.

Se observó que en muchos mensajes intervienen *timestamps*. En la primer versión del protocolo la serialización en JSON convertía estos objetos a cadenas de caracteres tomando en cuenta el horario local, lo que genera problemas si se consideran las

diferencias de zonas horarias, horario de verano, etc. Para eliminar estos problemas se optó por manejar dentro del protocolo los *timestamps* de otra forma en esta nueva versión.

A partir de esta versión, los *timestamps* se envían usando números de tipo "long", representando la cantidad de milisegundos desde el 1 de enero de 1970 a las 00:00.000 en horario GMT.

Los tipos de datos básicos identificados fueron: byte (y byte[]), UUID, int, double, long, String, MessageType, SocialChannelType. La representación de estos tipos de datos se realiza como se indica a continuación.

En tipos numéricos que ocupen más de un byte, se usa el estándar *Big Endian*.

B.1.2.14. Byte y byte[]

No es necesario realizar ninguna conversión.

B.1.2.15. UUID

Es codificado con los 16 bytes que lo representan.

B.1.2.16. int, long

Son codificados en palabras de 4 y 8 bytes respectivamente, usando el formato *big-endian*.

B.1.2.17. double

Es codificado con *binary64* perteneciente al estándar IEEE-754.

B.1.2.18. MessageType

Es codificado con un único byte, realizando la conversión de acuerdo a el Cuadro B.1:

Message Type	Valor
AREA_LIST_SOLICITATION	0x01
AREA_LIST_NOTIFICATION	0x02
BUS_LIST_SOLICITATION	0x03
BUS_LIST_NOTIFICATION	0x04
SOCIALCHANNEL_LIST_SOLICITATION	0x05
SOCIALCHANNEL_LIST_NOTIFICATION	0x06
USER_BUS_NOTIFICATION	0x07
BUS_NOTIFICATION	0x08
USER_SOCIAL_MESSAGE	0x09
SOCIAL_MESSAGE	0x0A
USER_BUS_STOP_NOTIFICATION	0x0B

Cuadro B.1: Tabla de conversión para MessageType

B.1.2.19. SocialChannelType

Los valores de este tipo son codificados con un único byte, según el Cuadro B.2:

SocialChannelType	Value
GLOBAL	0x01
AREA	0x02
BUS_NUMBER	0x03
BUS_LINE	0x04
BUS_INSTANCE	0x05

Cuadro B.2: Tabla de conversión para SocialChannelType

Algunos de estos valores no son usados por la aplicación, pero se incluyen para poder ser usados en un futuro si fuera necesario.

B.1.2.20. String

Para codificar este tipo de datos se encontraron dos problemas.

El primero fue que en el pasaje de Strings a arreglos de bytes es importante establecer la codificación de caracteres usada en la representación. Se optó por la codificación UTF-8 en este caso.

El segundo fue que por su naturaleza los valores de este tipo de datos no tienen una dimensión definida. En última instancia muchos valores transmitidos dentro del protocolo son persistidos en la base de datos del servidor en campos de tipo *CHAR* con tamaño acotado, por lo se podría haber limitado el tamaño de los strings a un cierto valor. Esto no fue hecho para no trasladar la limitación a la definición del protocolo y poder, por ejemplo, cambiar solamente la definición en la base de datos para alterar el tamaño máximo permitido para estos campos.

Por esta razón, es necesario agregar alguna estructura a la representación que permita encontrar dónde termina un string dada la posición donde comienza. Para resolver este problema fueron evaluadas dos posibilidades:

1. La primera es agregar información en el inicio de la representación del String indicando su longitud en bytes. Se consideró en un principio que esta no era una buena opción, ya que en última instancia se estaría limitando la cantidad de bytes a usar por el String a la cota máxima representable en el campo que contuviese a la longitud del mensaje.

2. La segunda es agregar un símbolo especial (el byte 0x00) al fin del *array* de bytes con la información del String. Este símbolo no aparece en textos codificados en UTF-8, por lo que no es necesario agregar técnicas de *byte stuffing* para considerar posibles apariciones de este símbolo dentro de los datos.

Finalmente se optó por la primer opción, usando dos bytes al inicio de la codificación para expresar el largo de la cadena de bytes que representa al String. Si bien existen limitaciones al tamaño de la cadena (que puede tener, como máximo, un largo de 65536 bytes), se eligió esta representación para aumentar la eficiencia y simplificar la decodificación de los mensajes en los clientes y en el servidor.

Usando esta codificación, la cadena de caracteres “Hola” quedaría representada como se ve en la Figura B.12.

0x00	0x04	0x48 (H)	0x6F (o)	0x6C (l)	0x61 (a)

Figura B.12: Ejemplo de codificación

B.1.2.21. Implementación de estas conversiones en Java.

Durante la investigación de posibles formas de codificar cada tipo de datos, se encontró la clase *ByteBuffer* que ya realiza muchas de estas conversiones de forma

nativa. Se extiende la funcionalidad de esta clase para agregar métodos que permitan leer y escribir al *buffer* datos con los tipos UUID, MessageType, SocialChannelType y String (los únicos no soportados nativamente por *ByteBuffer*).

B.1.2.22. Sintaxis genérica de los mensajes

	Campo	**Bytes**
Cabezal	Versión	1
	Tipo de Mensaje	1
	Identificador	16
	Firma	20
Datos	Datos	dependiente del tipo

Cuadro B.3: Sintaxis genérica de los mensajes

Los mensajes enviados usando este protocolo, tienen el formato definido en el Cuadro B.3. Los campos mencionados en este cuadro tienen los siguientes significados:

- **Versión**: Número de versión del protocolo. (1 byte)
- **Tipo de mensaje**. (1 byte)
- **Identificador**: UUID del Origen del mensaje, para los usuarios tendrá el valor de su *ID temporal* asignado por el servidor y en caso de que sea el servidor tendrá el valor 0. (16 bytes)
- **Firma**: Espacio reservado para incluir una firma en el mensaje. (20 bytes)
- **Datos**: Los datos dependen del tipo de mensajes por lo que tiene largo variable.

B.1.2.23. Sintaxis según el tipo de mensaje

Se presentan a continuación los formatos para el campo "datos" de cada uno de los tipos de mensajes. Todos los campos en esta descripción serán codificados según la pauta dada en la sección anterior para codificación de tipos de datos básicos. En la elección de la codificación no se tomaron mayores reparos más allá de la agrupación y el orden de campos que frecuentemente aparecen juntos (por ejemplo, latitud y longitud), con el objetivo de mantener un patrón en la forma de acceder a estos.

En las siguientes descripciones el orden en que aparecen los campos es el orden en que están dentro de los paquetes.

B.1.2.24. BusListSolicitation, SocialChannelListSolicitation, AreaListSolicitation

Mensajes de este tipo no requieren enviar información extra al servidor, por lo que su sección de datos tendrá largo cero.

B.1.2.25. AreaListNotification

En mensajes de este tipo se debe enviar la lista de áreas existentes en el sistema. Para enviar cada instancia, se usa la codificación mostrada en el Cuadro B.4

Campo	**Bytes**
AreaId	4
Name	strlen(name)+2

Cuadro B.4: Codificación propuesta para un área individual

Para enviar el conjunto deben concatenarse las cadenas de bytes de cada área, por lo que se envía un paquete con el formato definido en el Cuadro B.5

Campo	**Bytes**	
AreaId	4	Instancia 0
Name	strlen(name)+2	
...		
AreaId	4	Instancia n
Name	strlen(name)+2	

Cuadro B.5: Formato de mensaje para mensajes de tipo AreaListNotification

B.1.2.26. BusListNotification

En mensajes de este tipo se debe enviar la lista de ómnibus existentes en el sistema para el área instanciada en el *topic* usado. Para enviar cada instancia, se usa la codificación mostrada en el Cuadro B.6

Campo	Bytes
BusLineId	4
BusNumber	strlen(BusNumber)+2
BusDestination	strlen(BusDestination)+2

Cuadro B.6: Codificación propuesta para un ómnibus individual

En este caso también se representa un conjunto como la concatenación de las cadenas de bytes que representan a cada instancia. Por lo tanto, la representación es la presente en el Cuadro B.7.

Campo	Bytes	
BusLineId	4	Instancia 0
BusNumber	strlen(BusNumber)+2	
BusDestination	strlen(BusDestination)+2	
...		
BusLineId	4	Instancia n
BusNumber	strlen(BusNumber)+2	
BusDestination	strlen(BusDestination)+2	

Cuadro B.7: Formato de mensaje para mensajes de tipo BusListNotification

B.1.2.27. SocialChannelListNotification

En mensajes de este tipo el conjunto a enviar son los grupos sociales definidos en el sistema. La estructura elegida para representar las instancias de esta entidad es la presentada en el Cuadro B.8.

Campo	Bytes
Description	strlen(Description)+2
ServerChannel	strlen(ServerChannel)+2
ClientChannel	strlen(ClientChannel)+2
Type	1

Cuadro B.8: Codificación propuesta para grupo social

Por lo que la estructura final (la colección de elementos de esta entidad), es la mostrada en el Cuadro B.9

Campo	**Bytes**	
Description	strlen(Description)+2	Instancia 0
ServerChannel	strlen(ServerChannel)+2	
ClientChannel	strlen(ClientChannel)+2	
Type	1	
...		
Description	strlen(Description)+2	Instancia n
ServerChannel	strlen(ServerChannel)+2	
ClientChannel	strlen(ClientChannel)+2	
Type	1	

Cuadro B.9: Formato para mensajes de tipo SocialChannelNotification

B.1.2.28. UserBusNotification

En mensajes de este tipo, el formato utilizado es el mostrado en el Cuadro B.10.

Campo	**Bytes**
BusLineId	4
Latitude	8
Longitude	8
Accuracy	8
Timestamp	8
StatusArray	4
BusLineNumber	strlen(BusLineNumber)+2
BusLineDestination	strlen(BusLineDestination)+2

Cuadro B.10: Formato de mensaje para mensajes de tipo UserBusNotification

En este caso se optó por dejar al final los campos BusNumber y BusDestination ya que estos son sólo usados en los casos en que el campo BusLineId es cero, y por lo tanto es necesario que se de de alta un nuevo ómnibus con los datos especificados (esta situación es muy poco frecuente).

B.1.2.29. BusNotification

En mensajes de este tipo el servidor debe enviar en el mismo mensaje la información de todos los ómnibus existentes en el sistema para una línea dada. La codificación para una instancia particular es la siguiente presentada en el Cuadro B.11.

Campo	**Bytes**
BusId	4
StatusArray	4
Latitude	8
Longitude	8
Accuracy	8
Timestamp	8

Cuadro B.11: Codificación propuesta para ómnibus

Para codificar el conjunto se concatenan las cadenas de bytes de todas las instancias a enviar, como se muestra en el Cuadro B.12.

Campo	**Bytes**	
BusId	4	Instancia 0
StatusArray	4	
Latitude	8	
Longitude	8	
Accuracy	8	
Timestamp	8	
...		
BusId	4	Instancia n
StatusArray	4	
Latitude	8	
Longitude	8	
Accuracy	8	
Timestamp	8	

Cuadro B.12: Formato de mensaje para mensajes de tipo BusNotification

B.1.2.30. SocialUserMessage

El formato a usar por mensajes de este tipo es el mostrado en el Cuadro B.13.

Campo	Bytes
Message	strlen(Message)+2

Cuadro B.13: Formato de mensaje para mensajes de tipo SocialUserMessage

B.1.2.31. SocialServerMessage

El formato a usar en mensajes de este tipo es el mostrado en el Cuadro B.14.

Campo	Bytes
id	4
Nickname	strlen(nickname)+2
Timestamp	8

Cuadro B.14: Formato de mensaje para mensajes de tipo SocialServerMessage

B.1.2.32. UserBusStopNotification

En mensajes de este tipo el formato del mensaje a enviar es el mostrado en el Cuadro B.15.

Campo	Bytes
BusStopId	4
Description	strlen(Description) + 2
Latitude	8
Longitude	8
Accuracy	8

Cuadro B.15: Formato de mensaje para mensajes de tipo UserBusStopNotification

B.2. Comparación de las versiones del protocolo

Se realizó una comparación entre las versiones del protocolo para conocer el impacto de los cambios, en lo que respecta al tamaño de los mensajes enviados.

Para hacer esto, se tomaron en cuenta mensajes tipo generados en la primera versión del protocolo, y se calculó el tamaño de estos mensajes codificados con la nueva versión. Se presenta en esta sección la comparación del tamaño de los mensajes para cada tipo de mensaje enviados con ambas versiones del protocolo. No se presentan mensajes relativos al componente social del sistema, ni al reporte de paradas por parte de usuarios, ya que estos mensajes no existían en la primer versión del protocolo.

B.2.1. Consideraciones previas

En el funcionamiento de la aplicación estos mensajes siempre van acompañados de un cabezal MQTT con información sobre su destino. Para simplificar el análisis se eliminó este factor ya que es una constante independiente de la codificación utilizada.

Además, se va a realizar una simplificación en todo el análisis al asumir que el tamaño en bytes de toda cadena de caracteres enviada es igual al largo de la cadena. Esto no es necesariamente así, ya que de existir caracteres fuera del conjunto ASCII estos insumirían más de un carácter, pero se eligió hacer ésto ya que la codificación de caracteres usada para este tipo de datos es igual en ambas versiones del protocolo (siempre es UTF-8), por lo que ambas versiones se verían afectadas por igual, y además, porque la ocurrencia de estos caracteres no es significativa dentro del conjunto de cadenas de caracteres a enviar.

B.2.2. Tamaño del cabezal

En la segunda versión del protocolo se estableció la existencia de un cabezal constante a enviar en todos los mensajes sin importar su tipo. Este cabezal tiene como tamaño (a partir de la definición en el Cuadro B.3):

$$t_{CABEZAL} = 38bytes$$

B.2.3. BusListSolicitation, AreaSolicitation

Los mensajes de este tipo no precisan información adicional más allá de lo incluido en el cabezal del protocolo por lo que su tamaño en la segunda versión del protocolo es igual al tamaño del cabezal, 38 bytes.

En la primer versión del protocolo, tenían las codificaciones mostradas en las Figuras B.3 y B.2, con tamaños de 27 y 28 bytes respectivamente.

B.2.4. AreaListNotification

El mensaje analizado fue el mostrado en la Figura B.5. En la codificación de éste, se insumen 126 bytes (asumiendo un byte por caracter).

A partir de la definición (en el Cuadro B.5) del formato de mensaje para este tipo de mensaje, se obtiene que el tamaño de estos mensajes (incluyendo el cabezal), para la segunda versión del protocolo, está dado por la fórmula:

$$\sum strlen(name_i) + 6 * n + t_{CABEZAL}$$

Por lo que, instanciando las variables con los valores presentes en el mensaje mostardo anteriormente, se obtiene que:

$$6 * 3 + 38 + 12 + 10 + 10 = 88bytes$$

B.2.5. BusListNotification

El mensaje analizado fue el presente en la Figura B.7, cuyo tamaño es 1014 bytes (asumiendo un byte por caracter).

A partir de la definición (en el Cuadro B.7) del formato de mensaje para este tipo de mensaje, se obtiene que el tamaño en bytes de estos mensajes (incluyendo el cabezal), para la segunda versión del protocolo, está dado por la fórmula:

$$\sum strlen(BusName_i) + \sum strlen(BusNumber_i) + 8 * n + t_{CABEZAL}$$

Por lo que, instanciando las variables con los valores presentes en el mensaje mostardo anteriormente, se obtiene que:

$$8 * 11 + 38 + 28 + 136 = 290bytes$$

B.2.6. UserBusNotification

El mensaje de este tipo analizado fue el mostrado en la Figura B.11, cuyo tamaño es 406 bytes.

A partir de la definición (en el Cuadro B.10) del formato de mensaje para este tipo de mensaje, se obtiene que el tamaño de estos mensajes (incluyendo el cabezal), para la segunda versión del protocolo, está dado por la fórmula:

$$44 + strlen(BusNumber) + strlen(BusDestination) + t_{CABEZAL}$$

Por lo que, instanciando las variables con los valores presentes en el mensaje mostardo anteriormente, se obtiene que:

$$44 + 38 = 82bytes$$

Para este tipo de mensaje, se da el caso particular donde el mensaje puede contener los datos BusNumber y BusDestination. Esto ocurre en el momento en que el usuario se sube a un ómnibus de una línea no conocida al sistema, por lo que debe indicar explícitamente su destino y número. Este no es el caso del mensaje mostrado en la Figura B.11, en el que se indica una línea ya existente. El caso donde se especifican destino y número de línea de ómnibus es muy particular y poco frecuente, por lo que no se incluye en este análisis.

Excluyendo el caso mencionado en el párrafo anterior, se produce en este mensaje una mejora importante, que es pasar de un mensaje de largo variable a un mensaje de largo constante. Esto fue valorado como positivo, ya que todas las instancias de este mensaje que sean enviadas tendrán un largo de 82 bytes, sin importar (como sí importaba en la codificación anterior) el largo de algunos campos a enviar (por ejemplo, número y destino de la línea, o nombre del área).

B.2.7. BusNotification

El mensaje de este tipo analizado fue el presente en la Figura B.9, cuyo tamaño es 297 bytes.

A partir de la definición (en el Cuadro B.12) del formato de mensaje para este tipo de mensaje, se obtiene que el tamaño de estos mensajes (incluyendo el cabezal), para la segunda versión del protocolo, está dado por la fórmula:

$$40 * CantidadOmnibusReportados + t_{CABEZAL}$$

Por lo que, instanciando las variables con los valores presentes en el mensaje mostardo anteriormente, se obtiene que:

$$40 + 38 = 78bytes$$

Nuevamente en este mensaje se puede apreciar la ventaja de que el largo es constante para una cantidad de ómnibus reportados dada, y no depende del largo de atributos con tamaño variable.

B.2.8. Conclusiones

La información de las secciones anteriores se ve resumida en el Cuadro B.16. Puede observarse que los primeros dos mensajes poseen un tamaño mayor en la segunda versión que en la primera, mientras que en el resto de los casos se ocupa menos espacio en la nueva versión, por lo que en principio el resultado es exitoso en la mayoría de los casos.

Tipo de mensaje	Primera versión (bytes)	Segunda versión (bytes)
BusListSolicitation	27	38
AreaListSolicitation	28	38
AreaListNotification	126	88
BusListNotification	1014	290
UserBusNotification	406	82
BusNotification	297	78

Cuadro B.16: Comparación de tamaños de mensajes entre versiones

Si bien en el caso de los mensajes de tipo BusListSolicitation, y AreaListSolicitation se experimenta una desmejora en el tamaño del paquete, es importante considerar que los paquetes enviados en la segunda versión poseen más información que en la primera. En la segunda versión se incluyen:

1. Número de versión de protocolo: fundamental para asegurar que futuras versiones puedan coexistir con la actual

2. Identificador del usuario: permite identificar el usuario que emitió el mensaje

3. Firma: permitiría (actualmente no implementado) autenticar que el usuario identificado en el punto anterior sea efectivamente el emisor del mensaje.

Por otro lado, mensajes de estos tipos son enviados poco frecuentemente en el sistema: ambos son enviados una única vez en el momento que el usuario inicia la

aplicación. Considerando lo anterior el resultado obtenido es bueno, ya que se incluyó esta información con un bajo impacto en el uso de los recursos de red hecho por el sistema.

Es importante destacar, dentro de los resultados mostrados en el Cuadro B.16, que los mensajes de tipo UserBusNotification y BusNotification vieron su tamaño reducido a un veinte y veintiséis por ciento del tamaño original. Este resultado es importante, ya que estos mensajes son los mensajes más enviados por la aplicación en funcionamiento (el primero para enviar la posición del ómnibus en que se encuentra el usuario, el segundo para enviar las posiciones de ómnibus conocidas para una línea dada), por lo que el consumo total de la aplicación será afectado en forma muy positiva por este cambio.

En el análisis anterior se consideró sólo el tamaño de paquete. Además de este factor, la nueva codificación tiene otra ventaja que es la mejora en la performance de la decodificación. En la primera versión del protocolo, la codificación en JSON estaba hecha usando una biblioteca Java trabajando sobre *reflection* que serializa o deserializa cualquier objeto a su representación JSON. En la nueva versión, la codificación es estática y su tiempo de ejecución es lineal respecto al tamaño del mensaje. A diferencia de la primer solución, no hay costo de *parsing* o de correspondencia entre los atributos JSON y los atributos alto nivel de las clases en este caso, por lo que es de esperar que la decodificación sea más eficiente.

Como última conclusión, vale mencionar que la primer codificación fue útil, ya que permitió desarrollar de forma ágil el sistema sin prestar demasiada atención al protocolo, y validar rápidamente que la arquitectura de comunicaciones del sistema es factible. Fue positiva la experiencia de prototipar el protocolo usando una codificación en JSON y luego optimizarlo usando una codificación directamente en bytes cuando éste se encontraba estable.

Anexo C

Autenticación de los usuarios en el sistema

Se eligió para la autenticación utilizar las herramientas provistas como parte del sistema operativo Android en lo que refiere al uso de las cuentas almacenadas en el sistema. El usuario deberá elegir (en la primera ejecución de la aplicación), una de las cuentas de Google configuradas en el dispositivo, que pasará a ser la utilizada por la aplicación. Si el usuario cambiase de dispositivo, pero mantuviera la cuenta de Google, entonces el sistema reconocerá que se trata del mismo usuario.

C.1. Acceso a las cuentas Google en Android

Para realizar el acceso a las cuentas de Google en Android, se utilizó el mecanismo implementado por la API Account Manager [32]. Dentro de la aplicación, es usado para obtener la lista de cuentas definidas en el sistema de tipo Google, y luego, dada una cuenta, obtener un *token* OAuth correspondiente a esta, con acceso a consultar cuál es la dirección de email del usuario. Esta dirección será la usada por el sistema para identificar al usuario.

En la obtención de *tokens* OAuth la aplicación debe indicar al servicio de Google cuál es el alcance que deberá tener dicho *token* [33]. Se realizó un relevamiento de los alcances disponibles, y se optó por el que da acceso el email del la cuenta de Google del usuario, lo cual es el permiso mínimo y suficiente para las necesidades planteadas.

C.2. Autenticación con el servidor

Luego de obtener el *token* OAuth correspondiente al usuario dentro del dispositivo, se procede a registrar/autenticar al usuario en el servidor. Llamamos a esta operación con este nombre ya que en ambos casos el comportamiento en el teléfono es idéntico, siendo el servidor quién tiene algunas acciones diferentes dependiendo del caso. Para el usuario, no existe un caso de uso de registro en la aplicación por primera vez, aunque sí deberá configurar la aplicación en su primera ejecución cuando la instale en un dispositivo nuevo.

Para autenticar al usuario se llama a un servicio HTTP en el servidor incluyendo como parámetro el *token* OAuth obtenido en el dispositivo. El servidor verifica con Google que el *token* sea válido, garantizando que quien invoca al servicio es quien dice ser, ya que para poder obtener el *token* el usuario debe haber ingresado en algún punto sus credenciales Google. Si bien existen escenarios donde un atacante podría haber obtenido *tokens* OAuth válidos para usuarios ajenos para usarlos con la aplicación, se tomó la decisión de ignorar estos escenarios, ya que estos casos son propios de la seguridad en la obtención del *token* para el usuario, que está siendo "delegada" a Google.

Anexo D

Acta reunión 04/10/2012

D.1. Información de la reunión

- Fecha de la reunión: Jueves 04 de Octubre del 2012
- Hora comienzo: 18:30 hs
- Duración: 1 hora 30 minutos
- Participantes: Eduardo Grampín, Javier Bustos, Ricardo Rezzano, Leandro Scasso, Ignacio Tisnés, Vicente Acosta y Juan Pablo Revello.

D.2. Orden del día

1. Trabajos relacionados al CUDEN en Uruguay
2. Presentación del proyecto
3. Espacio para intercambio de ideas

D.3. Temas tratados

D.3.1. Trabajos relacionados al CUDEN en Uruguay

El objetivo principal de la reunión fue que Javier Bustos, quien participa del proyecto CUDEN como integrante de uno de los socios, NicLabs de Chile, tome contacto con las actividades relacionadas a CUDEN que se están desarrollando en Uruguay. Entre ellas se encuentra nuestro proyecto de grado “Redes cooperativas

móviles y su aplicación al transporte público" y la tesis de maestría de Leandro Scasso, que hará uso de los datos generados por este proyecto.

D.3.2. Presentación del proyecto

Se aprovechó la ocasión para presentarle también a Eduardo Grampín el proyecto, dado que es otro de los participantes del CUDEN en Uruguay. Se presentó la idea general del proyecto, cómo se enmarca en la temática y los objetivos del CUDEN así como la motivación para su realización y el problema particular al que nos enfrentamos.

D.3.3. Espacio para intercambio de ideas

Luego de la presentación tanto Eduardo como Javier valoraron positivamente el avance hasta ese momento del proyecto. Además, aportaron ideas que podrían contribuir al proyecto.

En particular, Javier comentó que tuvo la experiencia de desarrollar un sistema de similares características en Chile pero contando con datos brindados por la infraestructura misma de la red de transportes. Tomando esta idea nos sugirió que, dado que está disponible información detallada públicamente, se aproveche para poder brindar un mejor servicio a los usuarios, y poder tener un mayor volumen de datos que mejore las heurísticas de explotación de datos que se puedan utilizar.

Anexo E

Versiones de Android

Liberaciones del sistema operativo Android en el tiempo.

- Versión 1.0: La primer liberación fue en Setiembre del 2008, pero no fue utilizada en ningún dispositivo comercial.
- Versión 1.1: Pequeña actualización, liberada en Febrero del 2009, fue la primer liberación utilizada en un dispositivo comercial, el T-mobile G1.
- Versión 1.5: Gran liberación, disponible a partir de Abril del 2009, fue la primer liberación utilizada por varios fabricantes. Se le llamó «Cupcake».
- Versión 1.6: Liberada en Setiembre del 2009, junto con la versión 1.6 de su SDK. Se le llamó «Donut».
- Versión 2.0: Liberada en Octubre del 2009, junto con la v2.0 de su SDK, pocos dispositivos se liberaron con esta versión. Se le llamó «Eclair».
- Versión 2.1: Actualización, liberada en Enero del 2010, junto con la v2.1 del SDK, mucho más popular que la versión anterior. Al igual que la versión anterior, se le llamó «Eclair».
- Versión 2.2: Liberada en Mayo del 2010, junto con la v2.2 del SDK. Se le llamó «Froyo (The Frozen Yogurt)».
- Versión 2.3: Liberada en Diciembre del 2010, junto con la v2.3 del SDK. Ha tenido una serie de actualizaciones y correcciones desde su primer liberación, la ultima versión es la 2.3.7 liberada en Setiembre del 2011. Se le llamó «Gingerbread».

- Versión 3.0: Liberada en Febrero del 2011, junto con la v3.0 del SDK. Orientada a tabletas. Se le llamó «Honeycomb».
- Versión 3.1: Liberada en Junio del 2011, junto con la v3.1 de su SDK, incluye nuevas funcionalidades orientadas a tabletas respecto a la versión anterior. También se le llamó «Honeycomb».
- Versión 3.2: Liberada en Julio del 2011, junto con la v3.2 de su SDK, incluye mejoras y nuevas funcionalidades orientadas a tabletas respecto a la versión anterior. También se le llamó «Honeycomb».
- Versión 4.0: Liberada en Octubre del 2011, junto con la v4.0 de su SDK, combina las versiones 3.x y 2.x orientadas a tabletas y teléfonos respectivamente, en una sola versión. Se le llamó «Ice-Cream Sandwich».
- Versión 4.1: Liberada en Julio del 2012, junto con la v4.1 de su SDK, incluye varias mejoras a nivel de interfaz y nuevas funcionalidades. Se le llamó «Jelly Bean».
- Versión 4.2: Liberada en Octubre del 2012, junto con la v4.2 de su SDK, incluye varias mejoras a nivel de interfaz y nuevas funcionalidades. Se le llamó, al igual que en su versión anterior «Jelly Bean».

Son un total de 14 versiones en aproximadamente 49 meses.

Anexo F

Modelo físico de la base de datos

En la Figura F.1 se muestra un diagrama del esquema de la base de datos. Por más información sobre las entidades representadas, ver la Sección 4.3. A continuación se describe el propósito de cada tabla:

- **User:** representa a los usuarios y su información básica.
- **Area:** representa el área donde se encuentran el usuario y las líneas de ómnibus. Puede ser una ciudad, un país, región (el área metropolitana de Montevideo).
- **BusLine:** representa una línea de Ómnibus, tal como está definido en la Sección 4.3.1.2. Ej: 64 PLAZA INDEPENDENCIA.
- **BusStop:** representa las paradas de Ómnibus.
- **BusLineBusStop:** Almacena la relación entre una línea de ómnibus y las paradas que componen su recorrido. Además de definir el conjunto de paradas para una línea, tiene un orden asociado para representar el recorrido de estas.
- **BusLinePathPoint:** representa determinados puntos del mapa por donde pasa una línea de Ómnibus dada. No son todos los puntos por los cuales pasa el ómnibus, sino los necesarios para poder representar el recorrido. Notar que estos puntos pueden o no corresponderse a paradas. La razón por la que esta tabla fue introducida fue para manejar casos donde el recorrido real de una línea difiere mucho del resultado de concatenar los caminos en línea recta entre las paradas definidas en la tabla anterior.
- **Bus:** representa una instancia de una línea de ómnibus con datos tales como última posición y tiempo registrados.

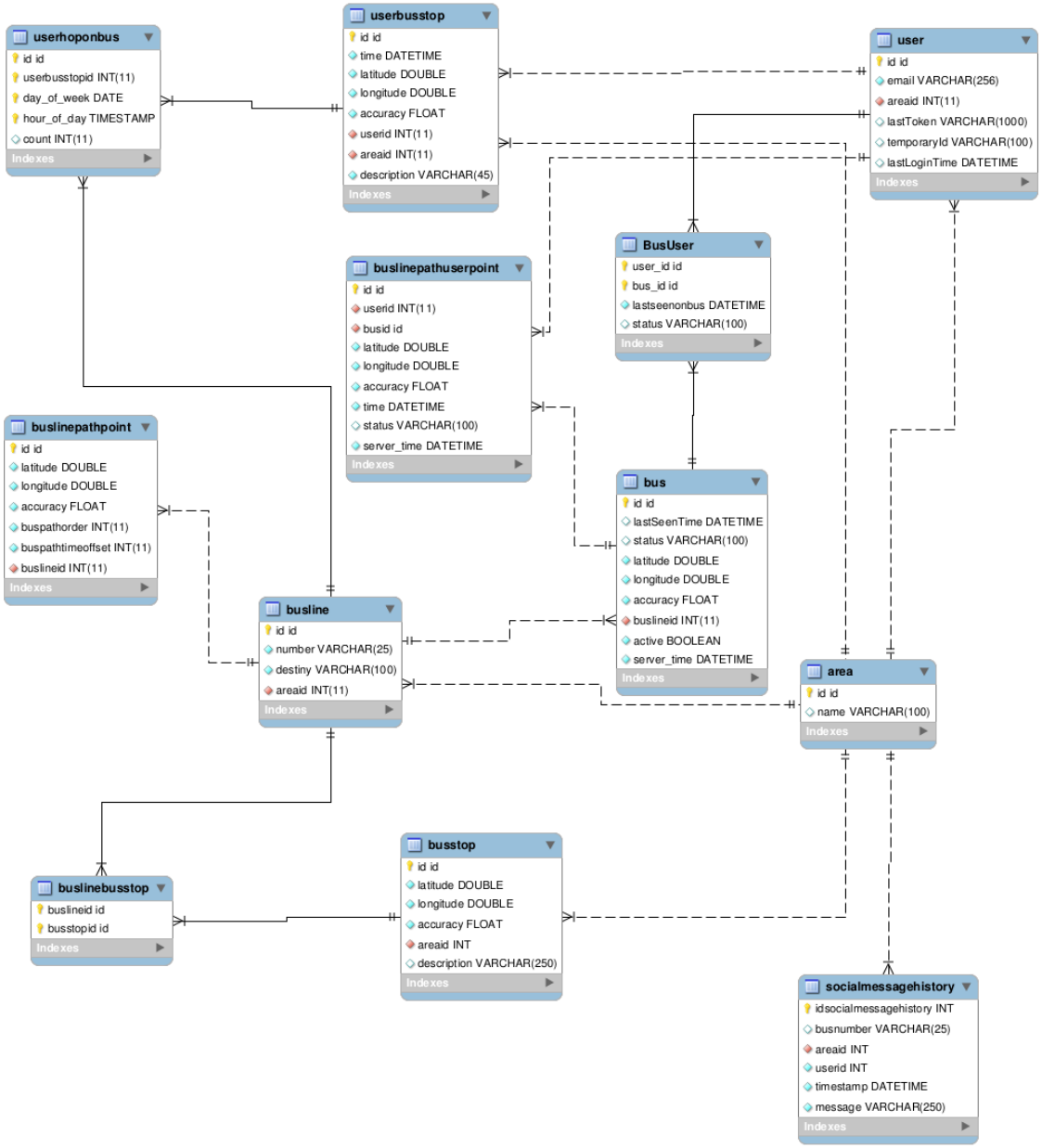


Figura F.1: Esquema de la base de datos

- **UserBusStop:** representa las paradas dadas de alta por los usuarios.
- **BusLinePathUserPoint:** representa los puntos por los cuales se mueve el usuario en una línea de ómnibus dada. La información guardada en esta tabla actúa como un log del usuario y es guardada (al menos) hasta su procesamiento para inferencia de recorridos y de tiempo de los mismos.
- **BusUser:** representa los usuarios que van en una instancia de ómnibus dada, e información que brinda el usuario sobre el ómbibus tales como el estado. Cuando un usuario informa de una posición nueva o una parada nueva, puede consultarse en esta tabla para saber cuál es el ómnibus en el que se encuentra el usuario. Esta tabla fue pensada para encontrar de una forma rápida cuál es el ómnibus en el que se encuentra el usuario.
- **UserHopOnBus:** representa el hecho de que un usuario se sube a una línea de ómnibus a determinado día/hora fijos. Puede servir para conocer los hábitos del usuario y brindarle información sobre los ómnibus que se toma más a menudo. Esta tabla se incluyó previendo posibles desarrollos futuros, pero no fue usada por el sistema desarrollado finalmente.
- **SocialMessageHistory:** Almacena los mensajes enviados dentro del subsistema social. De esta forma, los mensajes podrían ser enviados en el caso en que algún dispositivo perdió momentáneamente la conexión.

Anexo G

Extended abstract presentado a TRISTAN

En este anexo se presenta el *extended abstract* presentado en el marco de la conferencia TRISTAN, como se explicó en la Sección 5.3.2. Se optó por eliminar el formato original requerido, incluyendo únicamente el contenido concreto del documento. Este es presentado en inglés, su idioma original.

El documento original se incluye en el material adjunto en el archivo TRISTAN.pdf

G.1. Introduction

ARTBus is a distributed and collaborative solution, that aims the problem of real time, best effort, transport fleet tracking without any support from transport companies or goverments.

It relies on the sensors and the connectivity provided by smart devices owned by users of the public transport system.

The solution provides users with social tools to interact and create bonds with others sharing their path daily.

This project is based on a grade thesis work, coordinated the CUDEN (Collaborative Centric User-Device Networking) project, whose *"main objective is to provide support for ad-hoc cooperative communities between various devices to facilitate collaborative activities by extending users' physical experience and capabilities. (...) To this end, we will develop data processing and context modeling techniques to analyze users' social and surrounding environment information provided by various devices*

ranging from enhanced sensors to common products as smart-phones."[36]

The Android operating system, developed and sponsored by Google is based on the GNU/Linux kernel. Android was chosen as the development platform for this project, given that it is open source, contains a powerful development API, has been one of the fastest growing platforms lately[37, 38] and is the most used smart phone platform in South America.
Having chosen this platform the solution was developed using publicly available services such as Google Maps and Google Accounts. This project is work-in-progress at the time of the writing.

G.2. The solution

The solution's main function is to allow the user to know the position of the buses he is interested on using the information provided by other users. Optionally, more information could be provided regarding the status of said buses and the transportation system in general.

Despite having information available about the transportation system in our city, a decision was made not to use any of it for route gathering. It will be used for contrasting gathered information with real one. This decision was made so that the same solution would be deployed in a different city and it would be able to model the local transport network relying solely on the information provided by the users to determine the available bus lines, bus stops, etc.

The next sections explain the architecture and the communication scheme used by the solution.

G.2.1. The architecture

As a device centric solution, developed as an Android application, it can be explained by the roles of the devices. One of the roles is mainly a querying role where the user can aquire the information known by the system about the state of the transportation network. This information is, in turn, provided by these same devices acting under the second role, where the devices publish information in the system about information known to it (for example, the bus it is currently on and its position). Both roles can be active simultaneously in each device.

Another component of the solution is a server with the responsibilities of handling all information and summarizing the data provided by the users.

In addition to this, the solution contains a social component which enables user to exchange messages among them serving both an informative function (letting other people know about a specific situation in the transport system), and an entertainment one.

The basic idea is that this solution offers relevant information about the transit system and entertainment to the user in exchange of CPU time, battery, mobile bandwith, and sharing the information known to its device. Also, the social services are an extra reason for the users to give us their resources. We expect to provide powerful, attractive and innovative social services.

G.2.2. Communications

Users will select which information they are interested in (for example the position of all buses known belonging to a certain bus line). As many users will be expecting the same information, groups were defined according to that interest. The server will then send multicast-like messages to those groups according to their interests.

A custom protocol was made to allow this, which allows the server to send multicast-like messages to all users subscribed to events from a certain entity (for example, a bus line). This increases performance significantly because the processing is done as determined by the server, and then the result is delivered to all clients simultaneously. By using these multicast-like messages, a clean and efficient solution was acquired.

At the moment the server executes two components which handle the business logic of the solution and the delivery of messages to subscribed clients, but they can be easily separated in different servers to improve the scalability of the solution.

All communications between the bussiness logic component and the devices goes through the message delivery component. The latter manages the subscription to events defining topics dinamically in a way that is transparent to the end user. Thus, the message delivery component is responsible for ensuring that each client receives all and only the information requested by it.

G.3. Current status and roadmap

Currently, a second version of the solution has been completed, still in a prototype stage, including most of the core functionalities. One of the main objectives in this

phase is to develop an intuitive user interface so as to enhance the user experience. The corresponding Android application is on the proccess of being released as a free application for anyone in Montevideo. The first target group will be students from our University.

Up to this point, the solution relies heavily on the user to input relevant events manually. For example, it is required for each user to indicate when it is standing at a bus stop, gets on a bus, or gets off it. This stage is in part neccessary due to the self-learning property of the application, but the final objective is to include heuristics in the application that will detect these events, based on the information known about the area and the user, and using the data provided by the sensors in the device.

This objective is an important one, as it will enable the solution to function without requiring user interaction throughout the cycle. The user would be able to use the application's information as needed, but no conscious decision would be needed to share the information known to them (for example, the bus they are on). Developing these heuristics is our objective for the coming months.

A masters thesis is being developed also, whose objective is to take all the raw data gathered by this application and construct from it a model of the area's transportation system, overcoming various problems related to this data's quality. The final plan would be to use this information as an input for new heuristics.

Bibliografía

[1] Google. «Nexus 4 Tech Specs». http://www.google.com/nexus/4/specs/ disponible en 20121125Nexus4TechSpecs.html. Consultado el 25/11/2012.

[2] Qualcomm. «Snapdragon™ Processors». http://www.qualcomm.com.au/products/snapdragon disponible en Paginas_Web_Consultadas/20121124SnapdragonProcessorsQualcomm.html. Consultado el 25/11/2012.

[3] Gartner. «Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth». http://www.gartner.com/it/page.jsp?id=1924314 disponible en 20121124Gartner.htm. Consultado el 24/11/2012.

[4] Gartner. «Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012». http://www.gartner.com/newsroom/id/2335616 disponible en 20130223_GartnerSaysWorldwideMobilePhoneSalesDeclined1.7Percentin2012.htm. Consultado el 24/11/2012

[5] Intendencia de Montevideo. «Como ir». http://www.montevideo.gub.uy/mvdtv/mvd-te-dice/como-ir-0. Consultado el 24/11/2012.

[6] Gabriel Yordi. «STM Montevideo». http://stm.matungos.com.uy/ disponible en 20121030STMMontevideo.html. Consultado el 30/10/2012.

[7] Diego Rostagnol. «GXbus». http://gxbus.com.uy/ disponible en 20121030GXBus.html. Consultado el 30/10/2012.

[8] Google - Android developers. «General Transit Feed Specification Reference». https://developers.google.com/transit/gtfs/reference?hl=es-419 disponible en 20121124GeneralTransitFeedSpecificationReference-Transit\T1\textemdashGoogleDevelopers.html. Consultado el 24/11/2012.

[9] Google. «Google Transit Partner Program». http://maps.google.com/help/maps/transit/partners/ disponible en 20121124GoogleTransitPartnerProgram.html. Consultado el 30/10/2012.

[10] Movistar. «iBus». http://www.movistar.com.uy/Particulares/Servicios/LlamadasySMS/iBusCutcsa.aspx disponible en 20121030iBusSMSal933.html. Consultado el 30/10/2012.

[11] University of Washington. «OneBusAway». http://onebusaway.org/ disponible en 20121030OneBusAway.html. Consultado el 30/10/2012.

[12] Brian Ferris, Kari Watkins, Alan Borning. «OneBusAway: A Transit Traveller Information System». *Proceedings of Mobicase 2009*. http://wiki.onebusaway.org/bin/download/Main/Research/mobicase-2009.pdf disponible en 20121124mobicase-2009.pdf. Consultado el 24/11/2012.

[13] Pengfei Zhou, Yuanqing Zheng, Mo Li. «How Long to Wait?: Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing». http://www3.ntu.edu.sg/home/limo/papers/sys012fp.pdf disponible en 20121125sys012fp.pdf. Consultado el 25/11/2012.

[14] Google - Android developers. http://developer.android.com/develop/index.html disponible en 20130503_DevelopAndroidDevelopers.htm. Consultado el 25/11/2012.

[15] Twitter4J. «A Java library for the Twitter API». http://twitter4j.org/en/index.jsp disponible en 20121126_Twitter4J-AJavalibraryfortheTwitterAPI.html. Consultado el 26/11/2012.

[16] Twitter - Twitter developers. «OAuth». https://dev.twitter.com/docs/auth/oauth disponible en 20121126_OAuthTwitterDevelopers.html. Consultado el 26/11/2012.

[17] Twitter - Twitter developers. https://dev.twitter.com/docs/api disponible en 20130503_TheTwitterRESTAPITwitterDevelopers.htm. Consultado el 26/11/2012.

[18] Twitter - Twitter developers. «The Streaming API». https://dev.twitter.com/docs/streaming-apis disponible en 20121126_TheStreamingAPIsTwitterDevelopers.html. Consultado el 26/11/2012.

[19] Twitter - Centro de ayuda. «Sobre los Límites de Twitter (Actualizaciones, API, MD y Seguimiento)». http://support.twitter.com/articles/344781-sobre-los-limites-en-twitter-actualizaciones-dms-api-seguidores disponible en 20121126_SobrelosLímitesdeTwitter(Actualizaciones,API,MDySeguimiento).html. Consultado el 26/11/2012.

[20] Google - Android developers. «Android Cloud to Device Messaging Framework (C2DM)». https://developers.google.com/android/c2dm/ disponible en 20121126_AndroidCloudtoDeviceMessagingFramework-Android\T1\textemdashGoogleDevelopers.html. Consultado el 26/11/2012.

[21] Google Groups. «Parallel connections from application server». https://groups.google.com/forum/?fromgroups=#!topic/android-c2dm/3bYpDQaM3CM disponible en 20121126_Parallelconnectionsfromapplicationserver-GoogleGroups.html. Consultado el 26/11/2012.

[22] Google - Android developers. «Google Cloud Messaging for Android». http://developer.android.com/guide/google/gcm/index.html disponible en 20130408_GoogleCloudMessagingforAndroidAndroidDevelopers.html. Consultado el 08/04/2103.

[23] IBM - Developer works. «MQ Telemetry Transport (MQTT) V3.1 Protocol Specification». http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html disponible en 20130408_MQTelemetryTransport(MQTT)V3.1ProtocolSpecification.htm. Consultado el 08/04/2103.

[24] MQTT. «MQ Telemetry Transport». http://mqtt.org disponible en 20130409_MQTTMQTelemetryTransport.htm. Consultado el 09/04/2103.

[25] «Mosquitto, An Open Source MQTT v3.1 Broker.» http://mosquitto.org disponible en 20130409_AnOpenSourceMQTTv3.1Broker.htm. Consultado el 09/04/2103.

[26] Dale Lane. «Using MQTT in Android mobile applications». http://dalelane.co.uk/blog/?p=1599 disponible en 20130410_UsingMQTTinAndroidmobileapplicationsdalelane.htm. Consultado el 10/04/2103.

[27] MQTT,. «MQTT V3.1 Protocol Specification». http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html disponible en 20130409_MQTTV3.1ProtocolSpecification.htm. Consultado el 09/04/2103.

[28] Tokudu. «How to Implement Push Notifications for Android». http://tokudu.com/2010/how-to-implement-push-notifications-for-android/ disponible en 20130409_HowtoImplementPushNotificationsforAndroid.htm. Consultado el 09/04/2103.

[29] Facebook. «Building Facebook Messenger». https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920 disponible en 20130409_BuildingFacebookMessenger.htm. Consultado el 09/04/2103.

[30] Deacon. «The Deacon Project». http://deacon.daverea.com/ disponible en 20130409_TheDeaconProject.htm. Consultado el 09/04/2103.

[31] «Meteor, An HTTP server for the 2.0 Web». http://meteorserver.org/ disponible en 20130409_Meteor.htm. Consultado el 09/04/2103.

[32] Google - Android developers. «AccountManager». http://developer.android.com/reference/android/accounts/AccountManager.html disponible en 20130409_AccountManagerAndroidDevelopers.htm. Consultado el 09/04/2103.

[33] Google - Google's Internet Identity Research. «Email Display Scope». https://sites.google.com/site/oauthgoog/Home/emaildisplayscope disponible en 20130409_EmailDisplayScope-Google'sInternetIdentityResearch.htm. Consultado el 09/04/2103.

[34] FAQoid. «Android Timeline and Versions». http://faqoid.com/advisor/android-versions.php disponible en 20130409_AndroidTimeline,VersionsandChanges.htm. Consultado el 09/04/2103.

[35] Tristan. *http://www.tristan.cl/* disponible en 20130220_Tristan.html. Consultado el 09/04/2013.

[36] STIC-Amsud. «Collaborative Centric User-Device Networking». http://www-npa.lip6.fr/cuden/ disponible en 20130409_CUDENCollaborativeCentricUser-DeviceNetworking.htm. Consultado el 09/04/2013.

[37] StatCounter Global Stats. «Mobile OS (Operating System) Percentage Market Share Worldwide». http://stats.areppim.com/stats/stats_mobiosxtime.htm disponible en 20120504_stats_mobiosxtime.htm consultado el 04/05/2012.

[38] StatCounter Global Stats. «Mobile OS (Operating System) Percentage Market Share South America». http://stats.areppim.com/stats/stats_mobiosxtime_sam.htm disponible en 20120504_stats_mobiosxtime_sam.htm consultado el 04/05/2012.

[39] Google - Android developers. «Sensors Overview». http://developer.android.com/guide/topics/sensors/sensors_overview.html, disponible en 20130501_SensorsOverviewAndroidDevelopers.htm. Consultado el 01/05/2013.

[40] Google - Android developers. «Motion Sensors». http://developer.android.com/guide/topics/sensors/sensors_motion.html, disponible en 20130424_MotionSensorsAndroidDevelopers.html. Consultado el 24/04/2013.

[41] Google - Android developers. «Application Fundamentals». http://developer.android.com/guide/components/fundamentals.html, disponible en 20130427_ApplicationFundamentalsAndroidDevelopers.htm. Consultado el 01/05/2013.

[42] Google - Android developers. «Optimizing Battery Life». http://developer.android.com/training/monitoring-device-state/index.html, disponible en 20130501_OptimizingBatteryLifeAndroidDevelopers.htm. Consultado el 01/05/2013.