UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Solución de modelado para el posicionamiento continuo de entregas en el marco de Repartos a Domicilio
Autores: Iglesias Marchese, Mariano – Barrena Gualtieri, Hugo Daniel
Director: Dra. Cecilia Challiol
Codirector: Dra. Silvia Gordillo
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

Gracias al avance tecnológico y masificación del uso de teléfonos inteligentes vista en los últimos años, se han desarrollado una vasta cantidad de aplicaciones que hacen uso y sacan provecho de lo que el hardware provee. Como consecuencia de esto, los servicios basados en el posicionamiento en tiempo real están tomando cada vez mayor relevancia, generando cambios tanto en los mercados como en los hábitos de la gente. Las aplicaciones móviles basadas en posicionamiento van desde el transporte de productos, pasando por el ocio, la oferta de elementos de interés por cercanía, hasta el simple uso de una aplicación GPS con el fin de saber la posición de un individuo.

En esta tesina, se propone un modelo para el posicionamiento continuo de entregas en el marco de Repartos a Domicilio. En base a este, se implementa un prototipo funcional, donde se puede apreciar como los usuarios involucrados en el sistema pueden observar desde sus respectivas perspectivas el estado actual de las ordenes de los pedidos realizados por los clientes junto con su posición geográfica actual.

Para el posicionamiento de cada una de estas órdenes, se ha usado la posición del repartidor que las transporta, obtenida a través del GPS del dispositivo móvil de este.

Palabras Claves

Seguimiento continuo de entregas de pedidos, Aplicaciones Móviles basadas en Posicionamiento, Modelado, GPS, Movilidad

Trabajos Realizados

Se analizaron distintos sistemas de seguimiento en tiempo real utilizando dispositivos móviles presentes en el mercado actual.

Se propuso una solución de modelado genérica, la cual se puede extender para realizar el seguimiento continuo de productos con otras características, por ejemplo, reparto de correspondencia.

Además, se desarrolló un prototipo para un dominio específico, como es el seguimiento de repartos de órdenes de pedidos de un negocio de comida. A partir del mismo se mostró su funcionamiento con un ejemplo concreto.

Conclusiones

Se presentó un modelo flexible y adaptable, el cual se modelo usando distintos patrones de diseño.

En base al modelo propuesto, se implementó un prototipo, el cual se focaliza en brindar servicios de seguimiento continuo de entregas de pedidos provistos por un negocio de comidas. Este prototipo permitió observar que la posibilidad de uso de estos sistemas muchas veces puede estar limitado a la infraestructura de la red móvil con la que se cuente. Por ejemplo, en este caso, es crítico el envío de la posición GPS del repartidor.

Trabajos Futuros

- *Explorar distintos dominios de uso, para poder generar así extensiones concretas de los conceptos necesarios en el modelo propuesto.*
- *Definir otras estrategias para el cálculo de recorrido, por ejemplo, que las entregas tengan una prioridad.*
- *Considerar algún tipo de comunicación por medio de mensajes entre el negocio de repartos y los repartidores.*
- *Una posible incorporación al prototipo podría ser informar sobre cuantos pedidos faltan entregar antes de que le llegue el turno al del usuario junto al tiempo estimado de entrega.*

Fecha de la presentación: Diciembre de 2017

ÍNDICE GERENAL

1. Introducción

En este capítulo se describe la motivación que dio origen a esta tesina. Se especificarán los objetivos de la misma como así también se detallará la estructura definida a lo largo del documento.

1.1 Motivación

Desde el comienzo de la venta masiva de teléfonos inteligentes hace menos de 10 años[1], se dotó a la población con dispositivos de computo de propósito general de gran poder lo que generó la proliferación de aplicaciones móviles que dan solución a diferentes situaciones de la vida cotidiana; como puede ser el conocer en qué lugar estamos posicionados, sin contar con rústicos y costosos equipos de propósito específico, siendo estos la única opción previo a esta revolución [Chadil et al., 2008]. Los teléfonos inteligentes han evolucionado de tal manera que hoy en día casi no existe dispositivo inteligente que no cuente con sistemas de posicionamiento global (GPS[2]) o posicionamiento asistido (GPS-A[3]), esto generó que se desarrollen una amplia gama de aplicaciones que utilizan la posición del usuario para proveer algún tipo de servicio.

Como consecuencia de esto, los servicios basados en el posicionamiento en tiempo real o diferido de personas u objetos están tomando cada vez mayor relevancia, generando cambios tanto en los mercados como en los hábitos de la gente. Las aplicaciones móviles basadas en posicionamiento van desde el transporte de productos o personas, pasando por el ocio, la oferta de elementos de interés por cercanía, hasta el simple uso de una aplicación GPS con el fin de saber la posición de un individuo.

Existen varias publicaciones en las que se analiza y se destaca las ventajas, y las consideraciones éticas de realizar el seguimiento de personas en diferentes contextos, como lo pueden ser el rastreo de padres a hijos [Bhatia and Hilal, 2013], o el seguimiento de personas con demencia o alzhéimer [Landau and Werner, 20012]. Es decir, el número de dominios en el que las aplicaciones usan la posición del usuario es muy grande, así como también los servicios que se pueden brindar a través de estas [Emmanouilidis et al., 2013] y las implicancias que conlleva.

El ejemplo más común, de una aplicación que utiliza la posición del usuario es *Google Maps*[4], que entre sus servicios brinda la posibilidad de saber precisamente en donde estamos, buscar un destino e indicarnos como llegar, esta aplicación no solo permite ubicar destinos geográficos, sino que también brinda la posibilidad de encontrar lugares de interés

[1] El iphone 1 salió a la venta el 29 de junio de 2007 en Estados Unidos.
[2] Es un sistema de radionavegación basado en satélites desarrollado y controlado por el Departamento de Defensa de Estados Unidos de América que permite a cualquier usuario saber su localización, velocidad y altura, las 24 horas del día, bajo cualquier condición atmosférica y en cualquier punto del globo terrestre.
[3] El GPS Asistido es un sistema de posicionamiento por satélite que al iniciarse, recoge las coordenadas de las antenas para teléfonos móviles, o de WIFI para ubicar los satélites de posicionamiento global de una forma más rápida y eficiente.
[4] Página de *Google Maps*: https://maps.google.com.ar (Ultimo acceso 11/10/2016)

como puede ser un local de ropa. Otro ejemplo de aplicación es *Uber*[5], la cual permite contratar transporte y saber dónde se encuentra en tiempo real cada auto relacionado a dicho servicio. Se muestran los autos más cercanos a la posición actual del usuario, el cual puede ver como dichos autos se van moviendo en tiempo real y en base a esto decidir si contratar uno de estos para realizar un viaje.

Al igual que los casos mencionados, a nivel mundial existen miles de aplicaciones de propósito específico, orientados al seguimiento, que proveen servicios de posicionamiento como lo pueden ser *Glympse* [Glympse] (para compartir la posición con amigos, familiarizares y colegas), *Corvus GPS* [CorvusGPS] (servicio de tracking para vehículos desde el celular), *Pathshare GPS location sharing* [Pathshare] (compartir la posición con quienes el usuario elija), *Nereus* [*Nereus*] (tracking y auditoría de vehículos automotores, permite enviar imágenes y eventos a través de la red de telefonía móvil a un centro de monitoreo con un Smartphone), *Moby Simple Location Sharing*[6] (compartir recorridos, incorporando, por ejemplo, fotos).

A pesar de la variada gama de servicios, sigue siendo un mercado que cuenta con varios aspectos sin cubrir, donde uno de estos es el seguimiento continuo del posicionamiento de las entregas realizadas mediante repartos a domicilio (delivery). Estas entregas pueden ser alimentos o cualquier otro objeto que tenga que ser llevado al domicilio de un cliente. Generalmente, el repartidor tiene varias entregas por realizar simultáneamente, y el cliente no cuenta con un dato precisión de la posición actual de la entrega. Más aun, para los clientes sería interesante saber cuánto va a tardar la misma en arribar a su domicilio. Una muestra de esto puede verse en el sistema de seguimiento de envíos del *Correo Argentino*[7]. En este sistema se va conociendo la situación actual de un determinado envío, sin embargo, cuando el mismo se encuentra en manos del cartero no se puede determinar el recorrido que realizará, no se sabe en tiempo real en donde se encuentra el pedido, ni cuanto falta para que el envío sea entregado en el domicilio. Algo similar ocurre en el caso de reparto de alimentos, supongamos una cadena de supermercados que realiza entregas de pedidos, en este caso si bien se sabe que el pedido llegara en una franja horaria al domicilio del interesado, no se sabe cuánto podría faltarle o cuál es el recorrido del reparto. Esta información seria de sumo interés para los usuarios, por ejemplo, para saber si su pedido está en el primer orden de entrega o al final y permitirle de esta manera gestionar más eficientemente su tiempo. Otra clara ventaja seria la que obtendría quien gestione las entregas, pudiendo tener información precisa de en qué lugar se encuentra cada entrega, así como también, en qué lugar se encuentra cada uno de sus repartidores. Esta es la motivación principal de esta tesina, donde se busca brindar una solución de modelado para el seguimiento continuo del posicionamiento de las entregas realizadas mediante repartos a domicilio.

Es importante destacar que los ejemplos mencionados, no cuentan con un modelo subyacente de dominio público, sino que son aplicaciones creadas especialmente para un fin determinado. Esta tesina también hará un aporte en este aspecto, dado que varios de las

[5] Página de *Uber*: https://www.uber.com/es-US (Ultimo acceso 11/10/2016)
[6] Página de *Moby Simple Location Sharing*: http://mo.by/ (Ultimo acceso 17/10/2016)
[7] Página de *Seguimiento de Envíos del Correo Argentino*: http://www.correoargentino.com.ar/pagina/tt-seguimiento-de-envios (Ultimo acceso 11/10/2016)

características de la solución de modelado podrían ser útil para los distintos ejemplos de seguimiento mencionados.

1.2 Objetivo

Como se mencionó anteriormente, el objetivo principal de esta tesina es diseñar una solución de modelado para el seguimiento continuo del posicionamiento de las entregas realizadas mediante repartos a domicilio. Esta solución de modelado será flexible y adaptable a nuevos requerimientos [Weyns et al., 2015]. Cabe mencionar que para la solución de modelado se utilizaron distintos patrones de diseño [Gamma et al, 1995].

Una las características principales que tiene en cuenta la solución de modelado, es proveer el comportamiento para brindar soporte al seguimiento continuo del posicionamiento de las entregas. Dado que dichas entregas son realizadas por un repartidor, se toma la posición actual del mismo para poder proveer información de las entregas que realizará cada repartidor. De esta manera, cada entrega va a poder tener un posicionamiento continuo en tiempo real.

Otro de los objetivos que tiene esta tesina es implementar un prototipo funcional a partir del modelo propuesto. El prototipo se focaliza en brindar servicios de seguimiento continuo de entregas de pedidos provistos por un local de comidas. El prototipo contará de tres partes, una plataforma web que se encargara de la lógica de negocio y servirá para mostrar el recorrido de los repartidores en un mapa, una aplicación móvil que representara a cada repartidor y permitirá recolectar su información a medida que este se va moviendo, por último, una API que brindara acceso a parte de la lógica de negocio y servirá de interfaz de comunicación entre la plataforma web y la aplicación móvil.

El prototipo permitirá mostrar en el sitio web el recorrido en tiempo real de un repartidor que utilice la aplicación móvil. De esta manera, se podrá apreciar el posicionamiento actual de las entregas asociadas de este repartidor. A medida que el repartidor se vaya moviendo las posiciones de las entregas que faltan entregar también se actualizar.

1.3 Estructura del proyecto

A continuación, se describe la estructura de la tesina, y que aborda cada capítulo de la misma.

El Capítulo 2 se hace un análisis y resumen sobre algunos sistemas existentes que proveen servicios de posicionamiento en tiempo real. Además, se detallan los aspectos más relevantes de las tecnologías que sirvieron de base para el desarrollo del prototipo.

El Capítulo 3 se describe la problemática a resolver, y en base a la misma se detallan los requerimientos a contemplar especificando a partir de los mismos distintos casos de uso. Luego a partir de los mismos se proponer una solución de modelado. Se presentan diagramas de secuencia que se corresponden a los casos de uso especificados.

La arquitectura del prototipo funcional y su implementación se presenta en el Capítulo 4. Se detalla tanto la aplicación Web como la aplicación Móvil que componen dicho prototipo. Como así también la API que fue creado para el mismo. Se usan documenta con pantallas la visualización que recibirá cada uno de los actores que interactúan con el prototipo, *clientes*, *repartidor* y los *negocios*.

Posteriormente, en el Capítulo 5, se muestra con un ejemplo cómo funciona el prototipo desarrollado. Se cuentan con información precargada, la cual es detallada para comprender así los ejemplos que se muestran. En particular, se hace hincapié en la creación de un reparto, y como luego este se lleva a cabo por un repartidor, haciendo foco además en la información que reciben los clientes en cada momento.

Por último, en el Capítulo 6 se detallan las conclusiones de la tesina realizada como así también se mencionan posibles trabajos futuros que surgen de la misma, no solo a nivel del modelo propuesto sino también para el prototipo desarrollado.

2. Trabajos relacionados

En este capítulo se presentarán distintos sistemas existentes en diferentes dominios que realizan el seguimiento en tiempo real utilizando algún dispositivo móvil. Además, se presentan distintas tecnologías que sirvieron de base para el planteo de esta tesina.

2.1 Sistemas de seguimiento en tiempo real

En esta sección se presentarán distintos sistemas existentes que realizan el seguimiento en tiempo real utilizando algún dispositivo móvil.

- *CorvusGPS [CorvusGPS]*

  CorvusGPS posee un sistema de seguimiento de flotas en tiempo real, dispone de una aplicación para dispositivos móviles con sistema operativo Android, denominada "*EverTrak App*" que reporta la posición de estos usuarios entre otras funciones. *CorvusGPS* cuenta además con una plataforma web, una aplicación desktop (*CorvusGPS Desktop Map*) y móvil (*CorvusGPS Mobile Map*) para administrar usuarios y realizar el seguimiento en tiempo real de los mismos. Por otro lado, estos dispositivos poseen GPS preparados para dicha tarea. En la Figura 2.1 se puede apreciar el logo de *CorvusGPS*.

Figura 2.1: Logo de CorvusGPS [CorvusGPS]

A continuación, se detallarán cada uno de los servicios provistos por *CorvusGPS*.

- *EverTrak App* [CorvusGPS]

  Es una aplicación móvil que permite realizar el rastreo en tiempo real de la persona que la tenga instalada la aplicación en el celular inteligente. Además de la posición, esta aplicación reporta información del estado del celular (nivel de carga de la batería, nivel de señal, fecha y hora de última carga, etc.). Está desarrollada para permitir personalizar que opciones desea ver el usuario y cuales desea ocultar. Estas se pueden apreciar en la Figura 2.2.

  Cabe mencionar que el objetivo de *EverTrack* es lograr un transporte más eficiente, económico, y ecológico, mejorar la comunicación en el emprendimiento, y realizar una comunicaciones más flexible y conveniente.

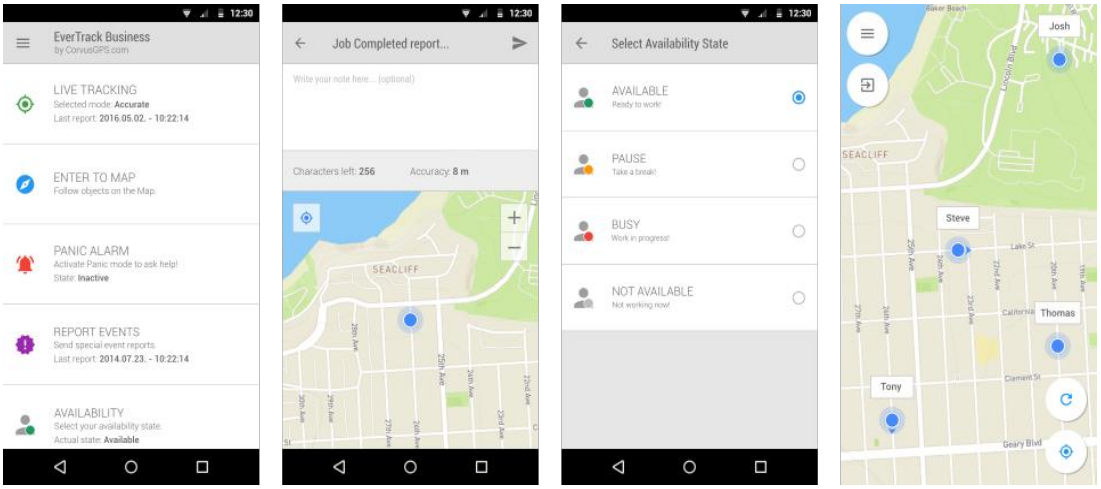

Figura 2.2: Aplicación *EverTrack* [CorvusGPS].

Entre las funcionalidades provistas por *EverTrak*, está disponible la opción de realizar envíos de eventos y notificaciones, o informes de estados del usuario, como parte de esto dispone de un botón de pánico que le permite reportar al centro de repartos y a sus compañeros cualquier emergencia o problema, así como también permite indicar que se ha completado un trabajo o que ocurrió un problema en la entrega. Estas diferentes opciones se pueden apreciar en la Figura 2.3.

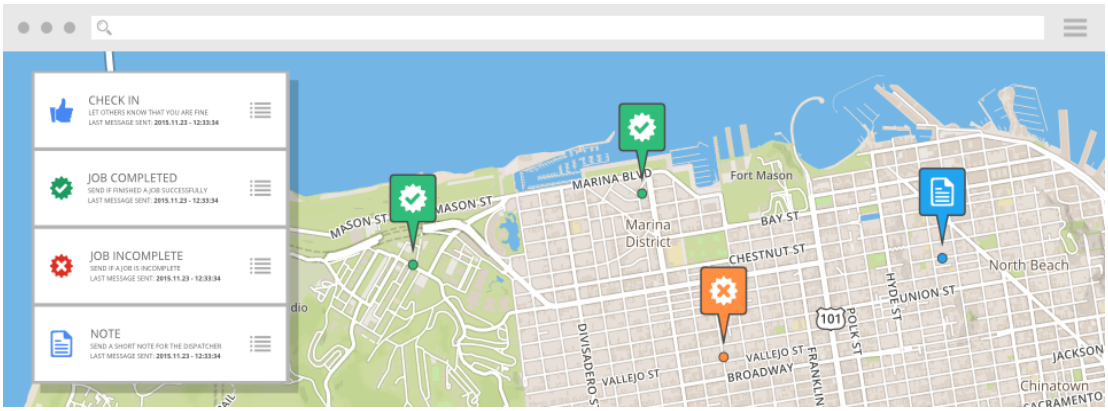

Figura 2.3: Notificaciones visibles desde la plataforma web de *CorvusGPS* [CorvusGPS].

En la Figura 2.4 se puede apreciar la opción de pánico tanto en el celular como así también como se visualiza en el mapa.

Tanto en las Figuras 2.3 como 2.4 se puede observar las distintas iconografías utilizadas por la plataforma web de *CorvusGPS*.

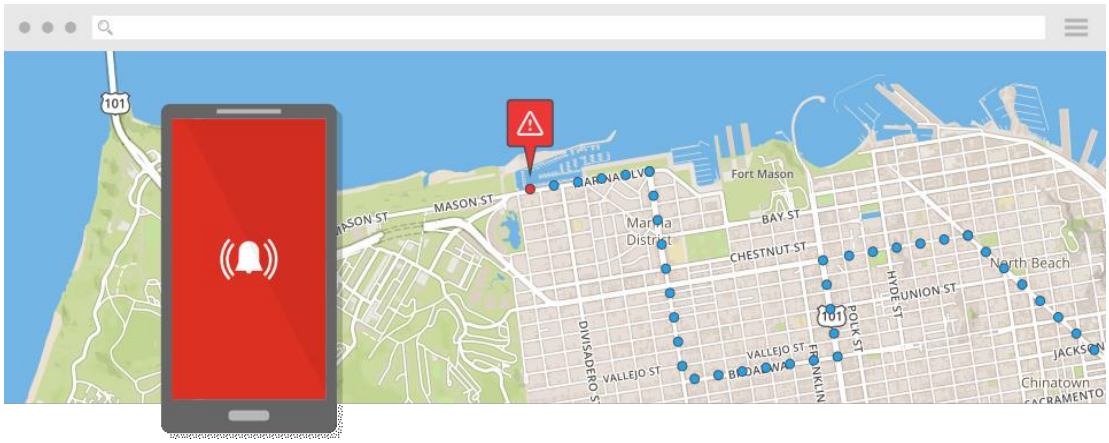

Figura 2.4: Aviso de activación del botón de pánico, visto desde la plataforma web de *CorvusGPS* [CorvusGPS].

Cada miembro del equipo que utilice *EverTrack* puede seleccionar su estado (*Disponible, Pausa, Ocupado, No Disponible*), con el fin de informar quién está trabajando. Cada estado tiene una iconografía distinta como se puede apreciar en la Figura 2.5.

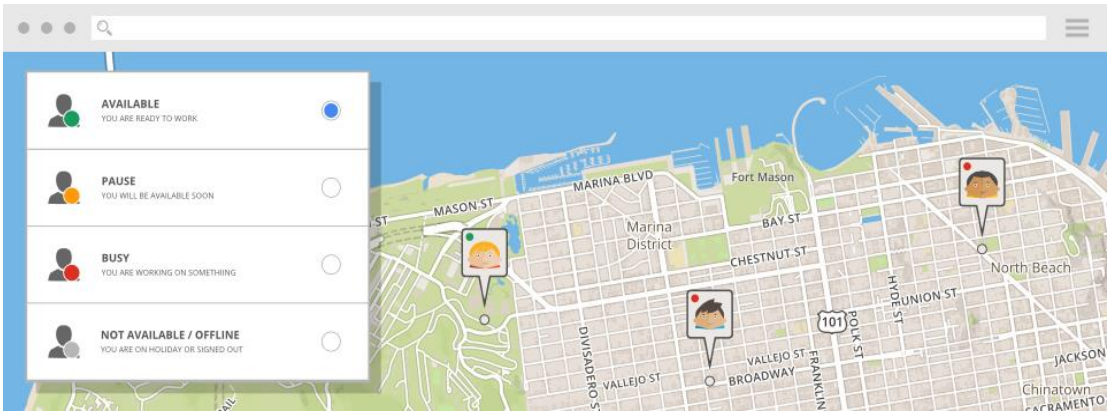

Figura 2.5: Estados visibles desde la plataforma web de CorvusGPS [CorvusGPS].

En comparación con los dispositivos de rastreo en tiempo real tradicionales, entre las ventajas que ofrece *EverTrack* están la carencia de instalación de algún equipo de rastreo, el no tener que contratar ningún nuevo servicio de telefonía celular, ya que puedes seguir utilizando el que dispones.

Instalado en un celular de gama media/alta, *EverTrack* provee mayor precisión de datos que un dispositivo de rastreo tradicional, también aprovecha los beneficios de elegir entre un conjunto grande de redes de datos (WIFI, 2g, 3g y 4g) provista por la telefonía móvil que suele ser de mejor cobertura y ancho de banda ya que la mayoría de los dispositivos convencionales utilizan las tecnologías más antiguas que en algunos países comienzan a ser discontinuadas.

EverTrack no pierde la información del usuario si no se puede conectar con los servidores de *CorvusGPS*, ya que almacena temporalmente la información, para enviarla cuando la conexión sea recuperada.

EverTrack asegura poder enviar información en no más de 30 segundos (algo que parece difícil de probar, más aún con las actuales redes de datos de argentina). Esta aplicación es utilizada para rastrear:

- Autos, Combis, Camiones, Motos, Bicicletas.
- Repartidores.
- Servicios de patrullas de seguridad.
- Camiones de comida o de helados.
- Trabajadores.

➢ *Visualización de distintos datos de los usuarios* [CorvusGPS]

Para ver el recorrido de un usuario, *CorvusGPS* tiene varias aplicaciones (web, móvil y de escritorio). Éstas también permiten gestionar los usuarios administradores como aquellos usuarios que serán "*rastreables*". Los usuarios "*rastreables*" son usuarios que cuentan con la aplicación *EverTrak* en su celular inteligente o bien, usuarios que cuentan con dispositivos GPS dedicados[8] que sean compatibles con *CorvusGPS*.

Los usuarios administradores podrán seguir (rastrear) a los usuarios "*rastreables*" individuales que dieron de alta o bien a un grupo de usuarios, ya que se pueden definir y asociar a distintos usuarios administradores un conjunto de usuarios "*rastreables* para su seguimiento". Estos usuarios podrán ser rastreados en tiempo real en un mapa, así como también se podrá visualizar toda la información que reporten además de su posición (información del dispositivo que usen para reportar su posición, eventos que hayan enviado, estado, etc.).

En la Figura 2.7 se puede apreciar a distintos usuarios posicionados sobre el mapa, como así también un conteiner identificado con el número 235. En este caso se está accediendo a la plataforma web de *CorvusGPS.* En el caso de cliquear sobre el icono de un usuario se puede ver información del mismo como se muestra en la Figura 2.8.

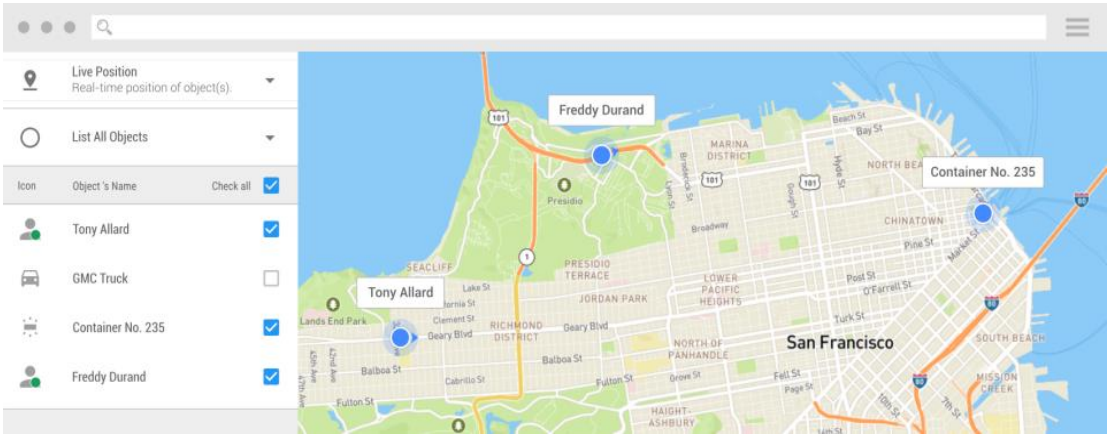

Figura 2.7: Visualizando usuarios desde la plataforma web de CorvusGPS [CorvusGPS].

[8] Más adelante se hará hincapié en estos dispositivos dedicados.

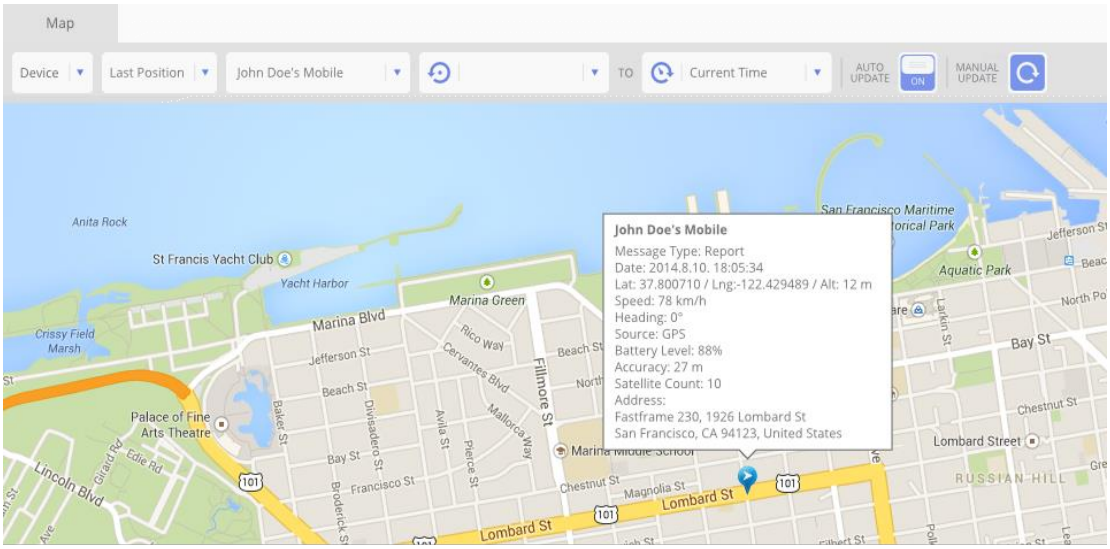

Figura 2.8: Información del dispositivo, visible desde la plataforma web de CorvusGPS [CorvusGPS].

Como se mencionó anteriormente, *CorvusGPS* permite crear grupos para saber la última posición de un conjunto de dispositivos al mismo tiempo, se pueden crear tantos grupos como sea necesario, pueden ser grupos privados y públicos. También permite compartir a través de un enlace estas posiciones en tiempo real a cualquier usuario no registrado, el cual solo podrá visualizar el mapa junto a las posiciones de los usuarios del grupo. Una persona que está observando al resto, también puede ser rastreada al mismo tiempo si tiene instalada la aplicación *EverTrack*. En la Figura 2.9 se puede visualizar a un grupo y la posición de las personas que lo integran.

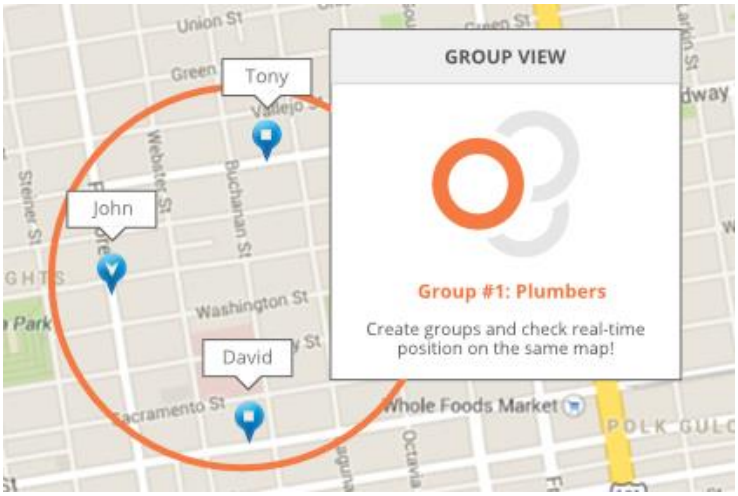

Figura 2.9: Grupo visible desde la plataforma web de *CorvusGPS* [CorvusGPS].

CorvusGPS permite visualizar un panel con una lista de eventos con la información más importante de cada dispositivo como se puede apreciar en la Figura 2.10.

Figura 2.10: Lista de eventos visible desde la plataforma web de *CorvusGPS* [CorvusGPS].

Cabe mencionar que el sistema envía notificaciones de los eventos más importantes (por ejemplo, aviso de ayuda proveniente del botón de pánico) para que el usuario sea notificado lo más rápido posible. Un ejemplo de esto se puede visualizar en Figura 2.11.

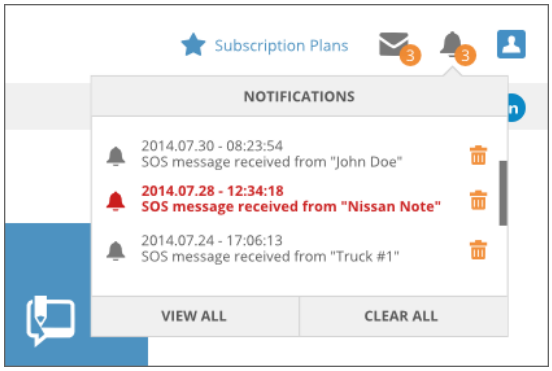

Figura 2.11: Notificaciones recibidas en *CorvusGPS* [CorvusGPS].

También cuenta con un sistema interno de mensajería que permite al repartidor intercambiar mensajes con su empleador, como se puede apreciar en la Figura 2.12.

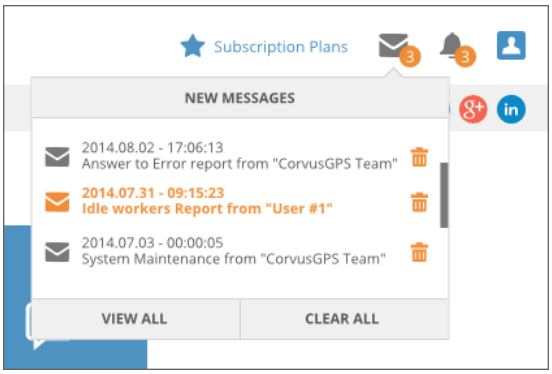

Figura 2.12: Mensajes recibidos en *CorvusGPS* [CorvusGPS].

Además, con *CorvusGPS* se pueden visualizar seguimientos pasados realizados en las últimas 5 semanas, por otro lado, se puede cambiar el proveedor de mapas, por ejemplo, *de Google a Open Street Maps*, como se puede apreciar en la Figura 2.13. Una vez seleccionado el mapa, se puede elegir tipo de visualización (satelital, calle o hibrido) como se puede apreciar también en dicha figura.

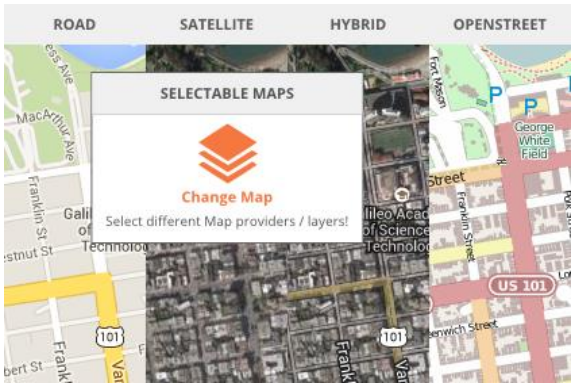

Figura 2.13: Diferentes vistas de mapas de *CorvusGPS* [CorvusGPS].

Adicionalmente, es posible ver el historial de un recorrido con colores según la velocidad del dispositivo, junto a la orientación del mismo e incluso si estuvo detenido. Un ejemplo de esto se puede apreciar en la Figura 2.14.

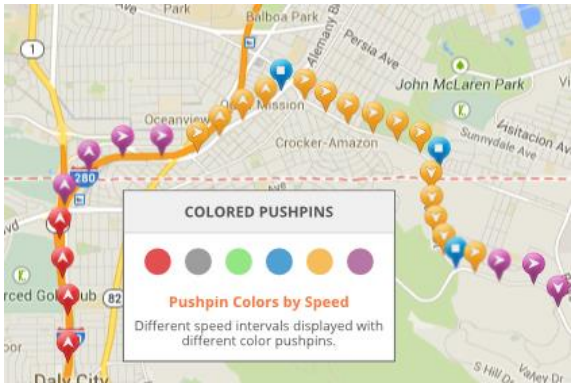

Figura 2.14: Historial visible desde *CorvusGPS* [CorvusGPS].

CorvusGPS cuenta con actualización automática de posiciones, en la que cada 5 segundos se encarga de refrescar en el mapa las posiciones de los dispositivos que se están rastreando como se puede apreciar en la Figura 2.15.

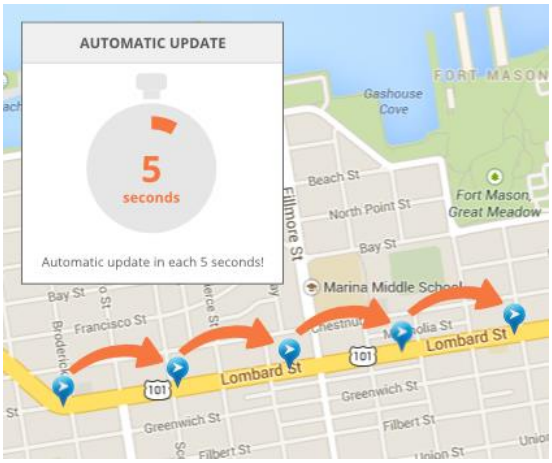

Figura 2.15: Refresco automático de la posición visible desde *CorvusGPS* [CorvusGPS].

Cabe mencionar que toda la información de *CorvusGPS* se encuentra almacenada de una "*Cloud"* segura, utiliza conexión HTTPS (con certificado SSL). El mapa móvil tiene limitaciones, por lo que para un seguimiento detallado recomiendan utilizar la web o la aplicación de escritorio.

CorvusGPS cuanta con una versión personal y otra empresarial, que se diferencian en las opciones que brinda el sistema, frecuencia máxima con la que el sistema reporta la posición, número de licencias, cantidad de días que se mantiene el historial de posiciones.

- *Dispositivos de seguimiento GPS tradicionales* [CorvusGPS]

Como se mencionó anteriormente *CorvusGPS*, no solo permite rastrear los dispositivos, sino que también permite rastrear dispositivos tradicionales de seguimiento GPS.

En caso de activos móviles, como contenedores, equipamiento o maquinaria de construcción, se dificulta la utilización de la aplicación *EverTrack*, por lo que en estos casos es crucial utilizar dispositivos GPS tradicionales.

CorvusGPS cuenta con un sistema que permite gestionar una flota de dispositivos GPS tradicionales, el cual cuenta con soporte para una amplia variedad de los dispositivos más populares del mercado, dándole al cliente la posibilidad de no depender de un celular inteligente y así poder utilizar un dispositivo dedicado y diseñado específicamente para el rastreo. En la Figura 2.16 se pueden apreciar algunos ejemplos de dispositivos de rastreo específicos.

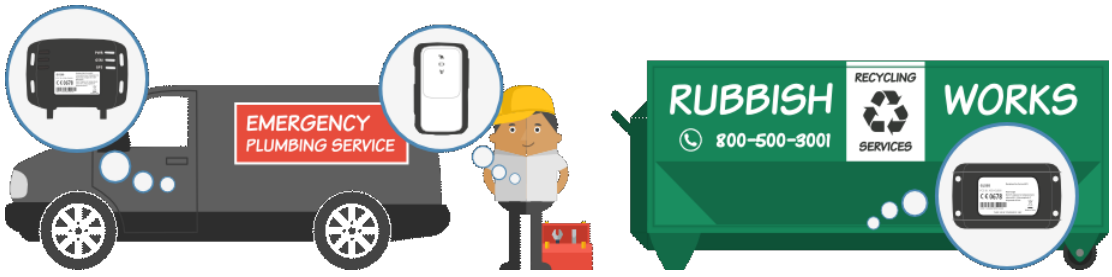

Figura 2.16: Algunos de los dispositivos de rastreo específicos de *CorvusGPS* [CorvusGPS].

CorvusGPS muestra la información que pueda obtener del dispositivo, por ejemplo, si el dispositivo está montado en un vehículo, podría mostrar el nivel de combustible, velocidad, revoluciones por minuto, alarma de temperatura o de mal funcionamiento. Existen incluso dispositivos que cuentan con una batería recargable cuyo fin es tener una larga duración y sensores que permiten medir el movimiento de donde se encuentra montado, por lo que la información de posicionamiento es mostrada solamente en cuando el activo se desplaza (por ejemplo, un contenedor de residuos), logrando de este modo que la batería dure con carga entre 1 y 3 años, dependiendo del nivel de uso de la misma.

- *Glympse [Glympse]*

  Es una de las empresas de geo-localización móvil más antigua, fundada en 2008 por ex empleados de Microsoft. *Glympse* provee una forma rápida y simple para compartir información de GPS del celular inteligente en tiempo real con quien se desee. El logo de *Glympse* se puede apreciar en el Figura 2.17.

Figura 2.17: Logo de *Glympse* [Glympse].

Glympse cuenta con un sitio web donde se puede monitorear el recorrido de los usuarios y una aplicación móvil para quien desee ser "*rastreado*". Desde la web, no es necesario crear una cuenta para usar el servicio, eliminando el *loggin* y *password*. Por otro lado, una vez instalada la aplicación móvil, simplemente se debe seleccionar uno o varios contactos, elegir la duración que se desea que esté disponible su ubicación y luego compartirla. Una vez compartida la ubicación, se puede extender el tiempo en la que se la va a compartir, también se puede *pausar* o *cancelar* el compartir dicha posición. En la Figura 2.18 se puede apreciar un ejemplo con distintos usuarios que compartieron su posición.

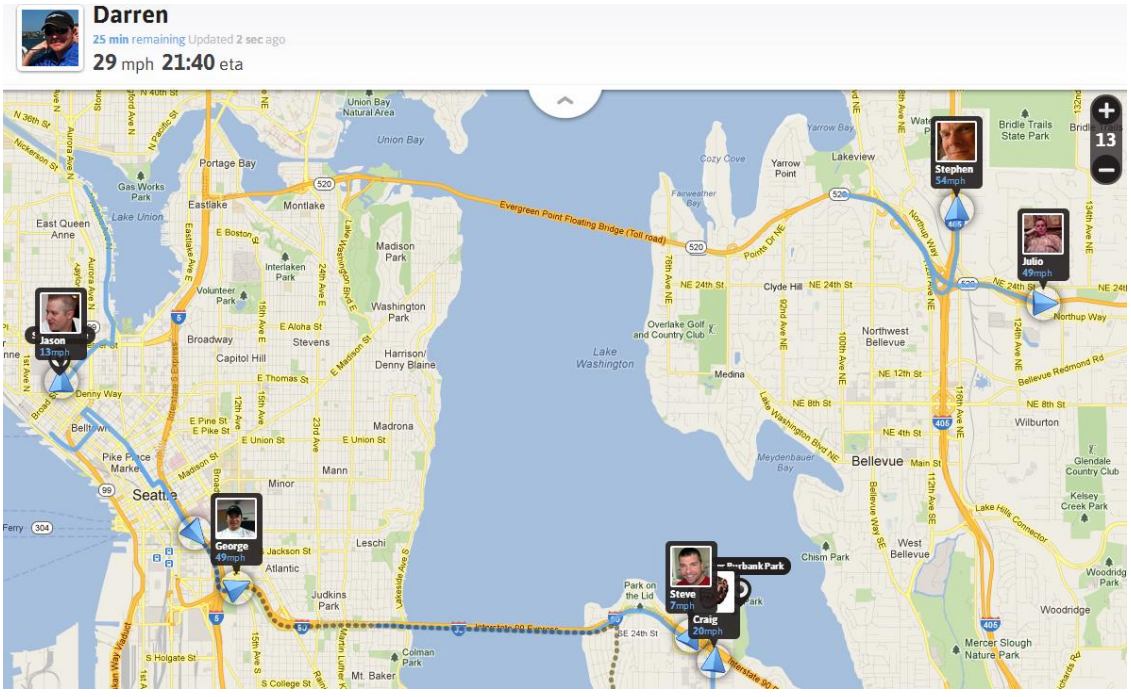

Figura 2.18: Plataforma web de *Glympse* [Glympse].

Glympse también permite solicitar la posición a uno de nuestros contactos. Cualquier persona con una conexión a internet, puede recibir la URL única mediante mensaje o email con la cual alguien está compartiendo su posición en tiempo real, que puede ser vista en un navegador o en la aplicación móvil. Dicha posición se irá sumando a las

posiciones previas, dibujando el trayecto realizado en un mapa. Un detalle interesante es que muestra cuantos segundos transcurrieron desde que se recibió la última posición, esto se puede apreciar en la Figura 2.19.

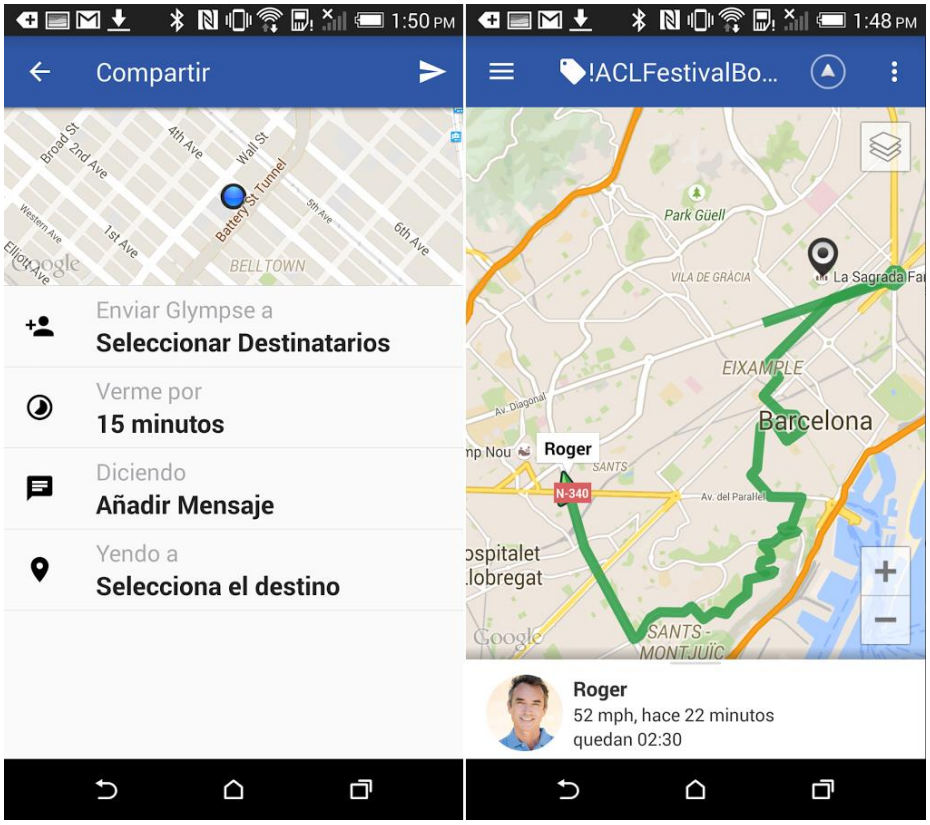

Figura 2.19: Aplicación *Glympse* [Glympse].

El sistema *Glympse*, posee tres opciones de uso:

- *Uso libre y social*, permite definir grupos de personas y un lapso de tiempo durante el cual se va a compartir la posición en tiempo real. Permite marcar un destino y personalizar un mensaje. Es multiplataforma (celular móvil, tablet, pc, reloj inteligente). Permite vincular la aplicación con cuentas de *Facebook*, *Twitter* o *Evernote*.
- *Uso comercial*, se basa en que las esperas generan estrés, por lo que mediante esté servicio se le puede facilitar a los clientes de quien los contrate un mapa con actualizaciones de tiempo real para sus productos o personas.
- *Integración con otras compañías*. Esta integración se realiza mediante el uso de una SDK de *Glympse* en algún sistema que otra compañía esté diseñando, puede ser tanto en una aplicación web u otro tipo de aplicación diseñadas para *Android*, *Windows Phone*, *Blackberry OS* o iOS. Algunas de las asociaciones con las que cuenta *Glympse* son con: BMW, Mercedes-Benz, Land Rover, Ford, WV, Volvo, Samsung, GARMIN, entre otras.

En la Figura 2.20 se puede apreciar la configuración de *Glympse* para autos. Mientras que en la Figura 2.21 se puede apreciar cómo se visualiza el mapa desde un auto, donde se está rastrando a un usuario especifico.

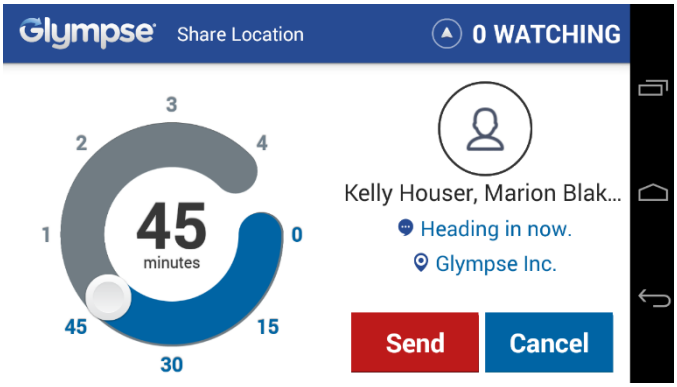

Figura 2.20: Configuración de *Glympse* para autos [Glympse].

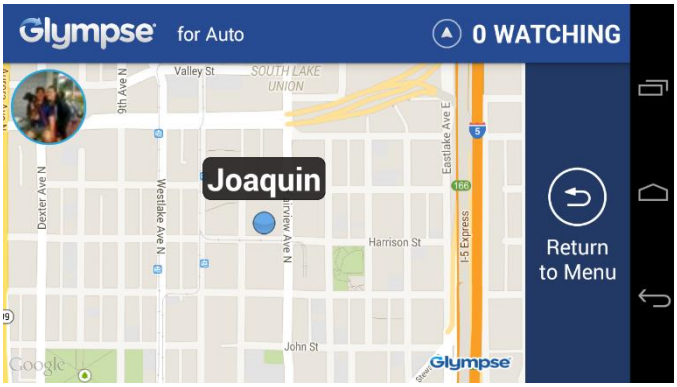

Figura 2.21: Vista de *Glympse* para autos [Glympse].

- *Pathshare GPS Location Sharing [Pathshare]*

  A diferencia de las tecnologías mencionadas anteriores *Pathshare* está pensada para un uso orientado a lo social, permitiendo compartir el posicionamiento de tu dispositivo móvil con un grupo de personas a través de una aplicación instalada en el mismo, posibilitando ver en un determinado momento la posición de aquellos que también te estén compartiendo su posición, sin hacer diferenciación entre usuarios.

  Pathshare permite crear un grupo, con un tiempo de duración del mismo, una vez creado el grupo permite invitar a unirse al grupo a otras personas. Estas personas reciben un enlace que al seleccionarlo las envía a la aplicación móvil de *Pathshare* o al navegador web del dispositivo móvil, y a partir de ese momento se suma al grupo, logrando que tanto su posición como la de la persona que creó el grupo sean visualizada en un mapa. Esto se puede apreciar en la Figura 2.22.

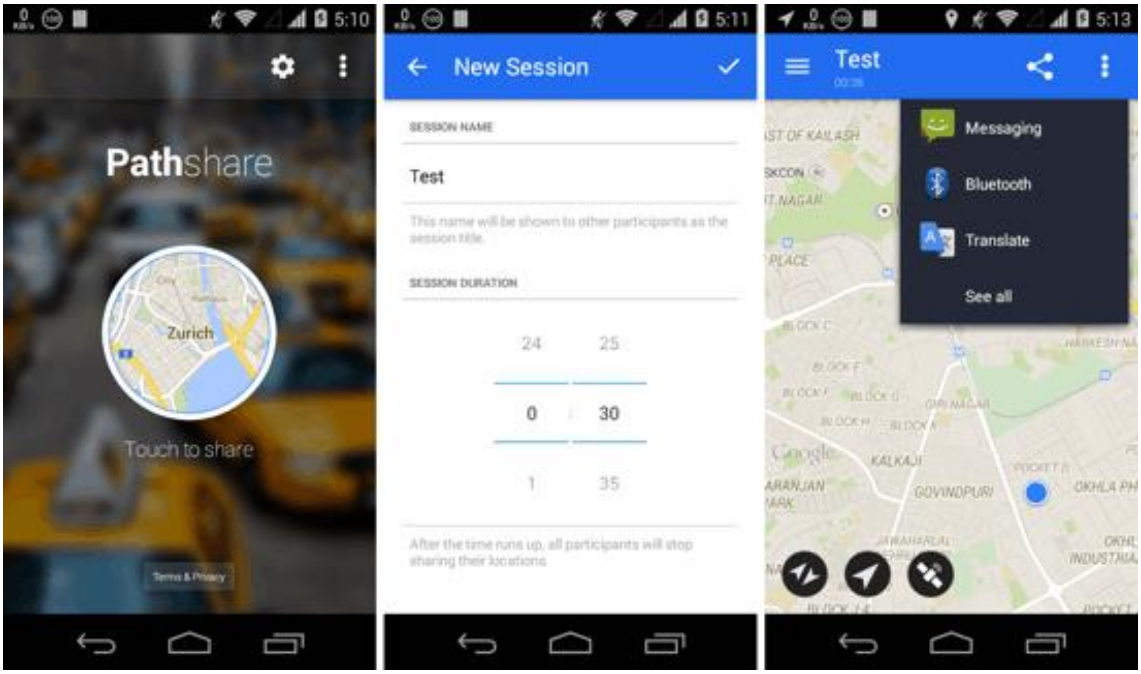

Figura 2.22: Aplicación *Pathshare* [Pathshare].

Pathshare puede establecer el mejor camino para llegar desde donde uno está hasta donde se encuentra otro de los usuarios (ver Figura 2.23), cuenta con notificaciones que avisan cuando una de las personas que comparten posición con el usuario se encuentran cerca, conoce en que se están desplazando las personas y enviar mensajes de información al resto de las personas que están compartiendo su posición en un momento dado. Todos los tipos de notificaciones que provee pueden ser realizadas por voz, siendo esto de importante ayuda para gente con capacidad visual disminuida.

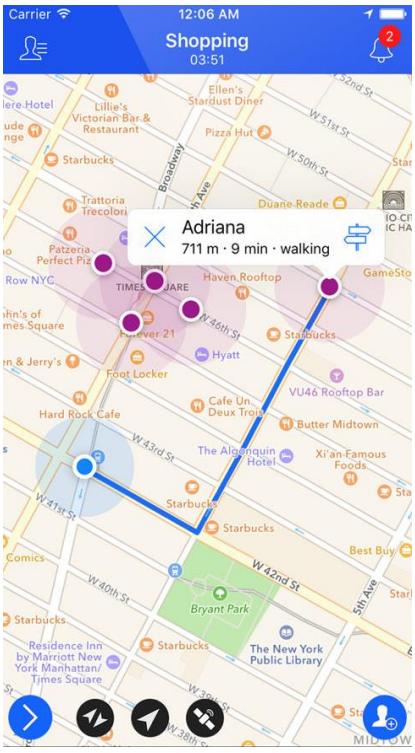

Figura 2.23: Personas cercanas y el mejor camino entre dos usuarios [Pathshare].

Una diferencia de *Pathshare* con respecto a *CorvusGPS y Glympse* es que indica la posición actual de una persona y no el trayecto que realizó.

Pathshare cuenta con modos de ahorro de batería y utiliza conexión segura HTTPS. Uno de sus socios es el pueblo de Hattiesburg del Condado de Forrest, Misisipi, Estados Unidos, que utilizan su servicio en el colectivo eléctrico que recorre el pueblo. *Pathshare* también provee integración mediante un SDK que permite utilizar todo el potencial de este sistema.

- *Nereus [Nereus]*

  En el ámbito local, *Nereus* una solución de bajo costo de instalación y operación, que se promociona a través del CESPI (*CEntro Superior para el Procesamiento de la Información*). *Nereus* posibilita el seguimiento, monitoreo, pero en este caso a diferencia de los sistemas mencionados anteriores permite enviar imágenes y eventos geo-referenciados a través de la red de telefonía móvil a un centro de monitoreo por intermedio de un celular inteligente.

  Nereus es confiable en caso de auditorías de vehículos automotores. Permite operar incluso cuando no hay señal por un período de tiempo determinado. Se puede integrar con otros dispositivos (AVLs - Localización Vehicular Automatizada, botones de pánico, etc.). Tiene un uso amplio, por ejemplo:

  - Combis escolares, colectivos urbanos, taxis, remises.
  - Control de patrullas municipales.
  - Fotos geo-referenciadas para mostrar sitios turísticos.
  - Botón de emergencia para violencia de género o comercios.

  Nereus requiere de un soporte para fijar el celular al vehículo. En la Figura 2.24 se puede apreciar un posible soporte. La aplicación tiene que estar ejecutándose, y poner el celular para que la cámara del mismo pueda capturar imágenes.

Figura 2.24: Soporte para el auto [Nereus].

En la Figura 2.25 se puede apreciar la posición de un usuario sobre un mapa junto a la imagen capturada por su dispositivo móvil.

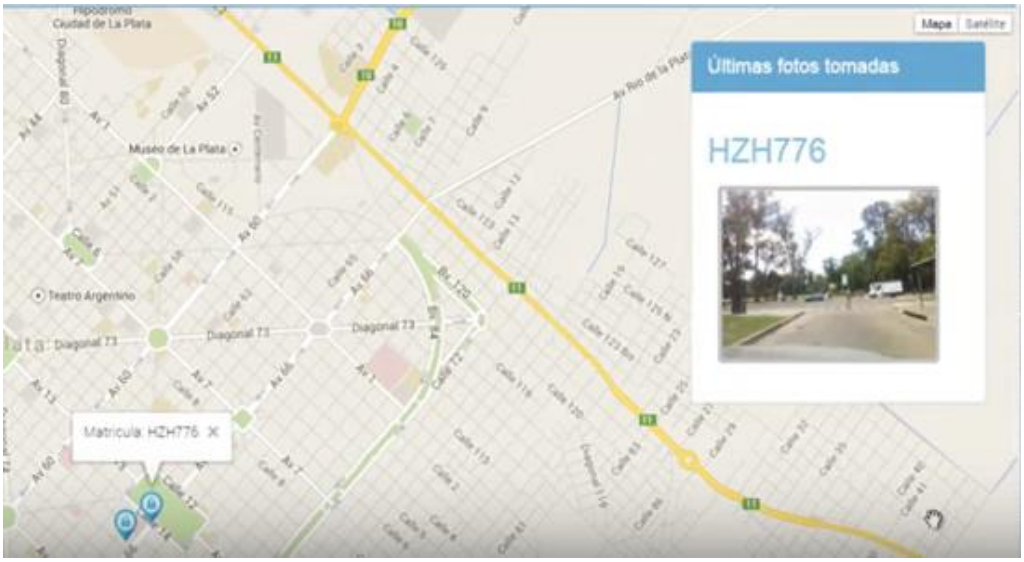

Figura 2.25: Imagen capturada desde un celular [Nereus].

2.2 Tecnología utilizada como base

En esta sección se relazará un resumen de las principales características de las tecnologías usadas como base para esta tesina.

2.2.1 Ruby on Rails [RubyonRails]

Ruby on Rails o simplemente *Rails* es un framework web open source (código abierto) escrito en el lenguaje *Ruby*[9]. Este framework como muchos otros frameworks web organiza la lógica de negocio de una aplicación siguiendo el patrón de diseño Model-View-Controller (MVC) como se puede apreciar en la Figura 2.26.

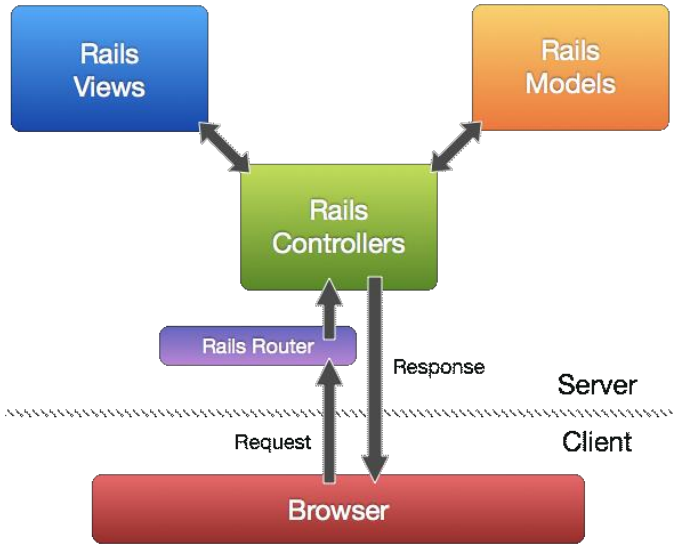

Figura 2.26: Arquitectura utilizada por Ruby on Rails [RubyonRails].

[9] *Ruby* es un lenguaje dinámico de código abierto basado en el paradigma orientado a objetos, enfocado en la simplicidad y productividad.

Rails fue pensado y concebido para conseguir una alta productividad en el desarrollo. Para conseguir esto, muchas de las funcionalidades requeridas para el desarrollo web están implementadas en el mismo framework y listas para utilizarse.

El framework *Rails* propone dos principios fundamentales para ayudar, guiar y agilizar el trabajo del desarrollador. Estos dos principios son la base de lo que plantea la "*filosofía Rails*":

- DRY (del inglés, *"Don't Repeat Yourself"*) - sugiere que escribir el mismo código una y otra vez es una mala práctica. Esto persigue como fin mantener el código limpio, reutilizar código cada vez que se pueda y reducir errores ya que los mismos errores no se replicaran en código repetido.
- "*Convención sobre Configuración*" - esto hace referencia a que el framework hace algunas suposiciones sobre lo que se quiere hacer y cómo se va a hacer, en lugar de requerir que se especifique cada pequeña cosa a través de un sin fin de archivos de configuración. Es así como se agiliza el desarrollo al requerir menos codificación por parte del desarrollador.

Rails, además, cuenta con una gran comunidad y un amplio conjunto de librerías presentes, facilitando y guiando aún más el desarrollo. En su última versión (al momento de redactar la tesina versión 5.0.1), *Rails* incluyó el modo –api el cual permite iniciar aplicaciones configuradas con un subconjunto reducido de componentes, obtenido a partir de la eliminación de aquellos que no son necesarios para el desarrollo de APIs web, y preparadas para servir contenido en formato JSON por defecto, aprovechando las técnicas de *caching* [Wessels, 2001] principales para este tipo de aplicaciones.

2.2.2 Sistema Operativo Android [Android]

El sistema operativo *Android* está basado en el kernel de Linux y ha sido desarrollado bajo el amparo de *Google*. En principio, nació como un sistema operativo para móviles, aunque hoy en día lo encontramos también en tablets, en la TV, en wearables, y en automóviles. *Android* está basado en una versión modificada del kernel de Linux, ofrece un conjunto completo de software de código libre para dispositivos móviles: un sistema operativo, middleware y aplicaciones esenciales para móviles[10]. Las aplicaciones para *Android* se programan en lenguaje Java y son ejecutadas en una máquina virtual diseñada para esta plataforma, que ha sido bautizada con el nombre de *Dalvik*. Dicha máquina virtual, está optimizada para requerir poca memoria y diseñada para permitir ejecutar varias instancias de la máquina virtual simultáneamente, delegando en el sistema operativo subyacente el soporte de aislamiento de procesos, gestión de memoria e hilos.

Android permite un acceso fácil a prácticamente todas las funcionalidades hardware del dispositivo en el que esté instalado. Provee a los desarrolladores de librerías que permiten la creación ágil y rápida de aplicaciones. Se distribuye bajo una licencia *Apache*[11] *versión 2*, es una licencia de software libre creada por la *Apache Software Foundation* (ASF). La Licencia *Apache* permite al usuario del software la libertad de usarlo para cualquier

[10] Android. What is?: https://developer.android.com/about/index.html (Ultimo acceso 28/01/2017).
[11] Condiciones de la Licencia Apache versión 2.0: //www.apache.org/licenses/LICENSE-2.0.html (Ultimo acceso 28/01/207).

propósito, distribuirlo, modificarlo, y distribuir versiones modificadas de ese software. La licencia *Apache* requiere la conservación del aviso de copyright y el disclaimer, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

A continuación, se hará hincapié en algunos detalles relevantes para esta tesina:

- *Arquitectura del Sistema Operativo Android [Android]*

  Como se mencionó previamente, el corazón de *Android*[12] es el Kernel Linux, donde se encuentran los drivers necesarios para el acceso al hardware del dispositivo en el que se encuentre instalado. En la Figura 2.27 se pude apreciar la organizado en capas con cuatro niveles de la arquitectura de *Android*.

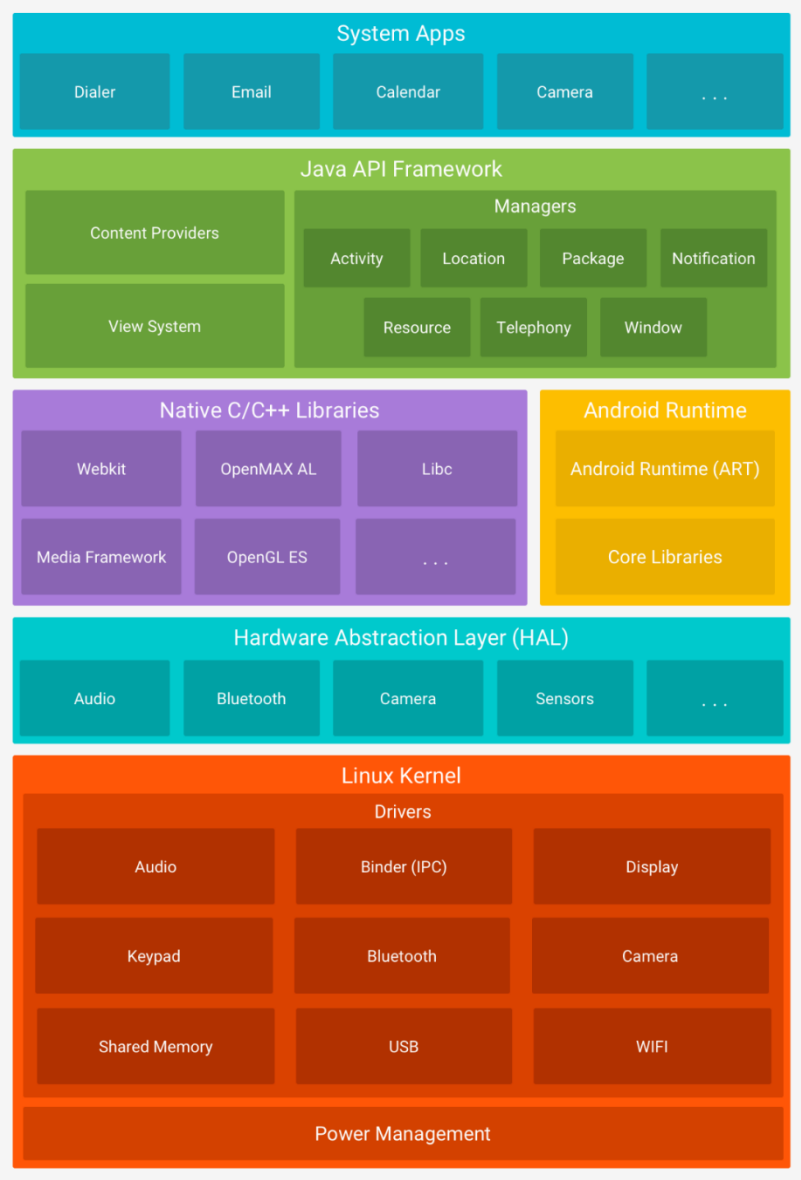

Figura 2.27: Arquitectura de Stack del Sistema Operativo Android [Android]

[12] Android: https://developer.android.com/guide/index.html

A continuación, se detallará una breve descripción de las distintas capas mostradas en la Figura 2.27:

- *Aplicaciones del sistema*

  Android incluye un conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las apps incluidas en la plataforma no tienen un estado especial entre las apps que el usuario elije instalar; por ello, una app externa se puede convertir en el navegador web, el sistema de mensajería SMS o, incluso, el teclado predeterminado del usuario (existen algunas excepciones, como la app Settings del sistema). Las aplicaciones del sistema funcionan como aplicaciones para los usuarios y brindan capacidades claves a las cuales los desarrolladores pueden acceder desde sus propias apps.

- *Java API Framework*

  Todo el conjunto de funciones de *Android* está disponible mediante API escritas en el lenguaje Java. Estas API son los cimientos que se necesita para crear aplicaciones simplificando la reutilización de componentes del sistema y servicios centrales y modulares, como los siguientes:

  - Un sistema de vista enriquecido y extensible que puedes usar para compilar la interfaz del usuario de la aplicación; se incluyen listas, cuadrículas, cuadros de texto, botones e incluso un navegador web integrable.
  - Un administrador de recursos que te brinda acceso a recursos sin código, como strings localizados, gráficos y archivos de diseño.
  - Un administrador de notificaciones que permite que todas las aplicaciones muestren alertas personalizadas en la barra de estado.
  - Un administrador de actividad que administra el ciclo de vida de las aplicaciones y proporciona una pila de retroceso de navegación común.
  - Proveedores de contenido que permiten que las aplicaciones accedan a datos desde otras aplicaciones o compartan sus propios datos.

  Los desarrolladores tienen acceso total a las mismas API del framework que usan las apps del sistema *Android*.

- *Bibliotecas C/C++ nativas*

  Muchos componentes y servicios centrales *Android*, como el ART y la HAL, se basan en código nativo que requiere bibliotecas nativas escritas en C y C++. La plataforma *Android* proporciona la API del framework de Java para exponer la funcionalidad de algunas de estas bibliotecas nativas. Por ejemplo, se puede acceder a *OpenGL ES* a través de la *Java OpenGL API* del framework de *Android* para agregar a tu aplicación compatibilidad con los dibujos y la manipulación de gráficos 2D y 3D.

  Si se desarrolla una aplicación que requiere C o C++, se puede usar el NDK de *Android* para acceder a algunas de estas bibliotecas de plataformas nativas directamente desde el código nativo.

- *Android Runtime (ART)*

  Para los dispositivos con *Android* 5.0 (nivel de API 21) o versiones posteriores, cada aplicación ejecuta sus propios procesos con sus propias instancias del ART. Éste está escrito para ejecutar varias máquinas virtuales en dispositivos de poca memoria ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para *Android* y optimizado para ocupar un espacio de memoria mínimo.

  Antes de *Android* 5.0 (nivel de API 21), *Dalvik* era el ART del sistema operativo. Si una aplicación se ejecuta bien en el ART, también debe funcionar en *Dalvik*, pero en el sentido inverso posiblemente no valga la misma afirmación.

  En *Android* también se incluye un conjunto de bibliotecas de tiempo de ejecución centrales que proporcionan la mayor parte de la funcionalidad del lenguaje de programación *Java*; se incluyen algunas funciones del lenguaje *Java 8*, que el framework de la Java API usa.

- *Capa de abstracción de hardware (HAL)*

  Brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al framework de la Java API de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la cámara o de bluetooth. Cuando el framework de una API realiza una llamada para acceder a hardware del dispositivo, el sistema *Android* carga el módulo de biblioteca para el componente de hardware en cuestión.

- *Kernel Linux*

  La base de la plataforma *Android* es el kernel de Linux. Por ejemplo, el ART se basa en el kernel de Linux para funcionalidades subyacentes, como la generación de subprocesos y la administración de memoria de bajo nivel. El uso del kernel de Linux permite que *Android* aproveche funciones de seguridad claves y, al mismo tiempo, permite a los fabricantes de dispositivos desarrollar controladores de hardware para un kernel conocido.

- *Componentes de una aplicación Android [Android]*

  Android está basado en componentes independientes que interactúan entre sí. En una aplicación con audio, por ejemplo, se tiene al menos dos componentes: uno ejecutándose en segundo plano reproduciendo el audio, y otro en primer plano gestionando la interfaz de usuario. Es importante destacar que estos componentes son independientes, ya que podría haber una aplicación de terceros que también gestione ese componente de audio en segundo plano, o usar un componente que exponga un tercero.

  Dentro de una aplicación de *Android* hay cuatro componentes principales: *Activitity*, *Broadcast Receivers*, *Services* y *Content Provider*. Todas las aplicaciones de *Android* están formadas por algunos de estos elementos o combinaciones de ellos.

A continuación, se describen algunas características de los cuatro componentes principales de *Android*:

- *Actividades (Activity)*

  Una *Actividad* se corresponde con una pantalla de la aplicación, es decir, una aplicación consistirá en un conjunto de actividades independientes que trabajan juntas, de las cuales una se especificará como la principal de la aplicación. Para implementarlas se utiliza una clase por cada actividad que extiende de la clase base `Activity`. Cada clase mostrará una interfaz de usuario, compuesta por Views (o Vistas). Cada *Actividad* es responsable de mantener su estado, de forma que puedan integrarse en el ciclo de vida de la aplicación, que es gestionado por el propio framework de aplicación. En *Android* se puede crear las interfaces de usuario de dos formas, desde la propia actividad usando código Java, o usando un fichero XML para describirla como si fuera una página HTML. Esta última es la manera más sencilla y cómoda.

- *Servicios (Services)*

  Un servicio no tiene una interfaz de grafica (UI), sino que se ejecuta en background por un período indefinido de tiempo, cada servicio extiende de la clase `Service`. Por ejemplo: un reproductor de música que se ejecuta en background permitiendo que se realicen otras actividades como enviar mensaje de texto (SMS) o ejecutar alguna otra aplicación mientras se escucha música.

- *Receptores de eventos (Broadcast Receivers)*

  Los receptores de eventos no realizan ninguna acción por sí mismo, recibe y reacciona ante anuncios de tipo broadcast, por ejemplo: batería baja. No tienen UI, aunque pueden iniciar una actividad para atender al anuncio. Todos los receiver extienden de la clase `BroadcastReceiver`.

- *Proveedores de contenido (Content Providers)*

  Un proveedor de contenido permite que una aplicación ponga contenido a disposición de otra. Se encargan de administrar el acceso a un conjunto estructurado de datos. Encapsulan los datos y proporcionan mecanismos para definir la seguridad de los datos. Los proveedores de contenido son la interfaz estándar que conecta datos en un proceso con código que se ejecuta en otro proceso.

  Cuando se quiere acceder a datos en un proveedor de contenido, se utiliza el objeto `ContentResolver` en el `Context` de la aplicación para comunicarte con el proveedor como cliente. El objeto `ContentResolver` se comunica con el objeto del proveedor, una instancia de una clase que implementa `ContentProvider`. El objeto del proveedor recibe solicitudes de datos de clientes, realiza la acción solicitada y devuelve resultados.

  No se necesita desarrollar un proveedor propio si no se pretende compartir los datos con otras aplicaciones. Un ejemplo de proveedor de contenidos que ofrece *Google* son los contactos que se tengan almacenados en el celular.

Cabe mencionar que los *Content Provider* son activados al recibir una petición de un `ContextResolver`, pero los otros tres componentes (*Actividades*, *Servicios* y *Broadcast Receivers*), son activados por mensajes asincrónicos llamados *Intents* (Intención).

Los *Intents* son los elementos básicos de comunicación entre los distintos componentes Android. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un *Intent* se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje *broadcast*, iniciar otra aplicación, etc. Un *Intent* es un objeto mensaje y que, en general, describe que quiere hacer una aplicación. Las dos partes más importantes de un *Intent* son la acción que se quiere realizar y la información necesaria que se proporciona para poder realizarla, la cual se expresa en formato URI.

Relacionado con los *Intents* hay una clase llamada `IntentFilter` que es una descripción de que *Intents* puede gestionar una *Actividad*. Mediante los *IntentFilters*, el sistema puede resolver *Intents*, buscando cuales posee cada actividad y escogiendo aquel que mejor se ajuste a sus necesidades. El proceso de resolver *Intents* se realiza en tiempo real y nos da el beneficio de que las actividades pueden reutilizar funcionalidades de otros componentes simplemente haciendo peticiones mediante un *Intent*. Por ejemplo, una aplicación puede requerir visualizar una página web, para ello sería necesario crear un *Intent* de tipo VIEW y añadir como datos de la intención la URI de la página web, de esta forma el sistema registraría la creación de un *Intent* y se encargaría de buscar que aplicaciones son capaces de resolverlo.

Los *Intents* e *IntentFilters* son una manera de encapsular tareas, aislar dependencias y asegurar la reutilización de funcionalidades. Tener en cuenta:

- Cada aplicación es responsable únicamente de su funcionamiento, sabiendo que para realizar cualquier tarea que no sea de su ámbito únicamente debe realizar un *Intent*. De esta forma se puede construir aplicaciones centradas en una única actividad y cuyo trabajo pueda ser aprovechado por otras aplicaciones.
- Los *Intents* sirven como interfaz de comunicación entre aplicaciones. Cualquier aplicación puede lanzar un *Intent* sin necesidad de conocer ningún detalle sobre la aplicación que lo va a resolver. No genera dependencia entre aplicaciones, cualquier actividad puede ser reemplazada por otra que tenga sus mismos *IntentFilters* y el reemplazo es transparente para todos los procesos que usan la funcionalidad original.
- Cuando una aplicación registrar su *IntentFilter* y por ser declara capaz de manejar una tarea, está aportando una funcionalidad que el resto de las aplicaciones no necesita implementar. Su código será reutilizado por todas las aplicaciones que lo necesiten.

- *Ciclo de vida de una actividad (Activity) [Android]*

  En *Android*, cada aplicación se ejecuta en su propio proceso. Esto aporta beneficios en cuestiones básicas como seguridad, gestión de memoria, o la ocupación de la CPU del dispositivo. *Android* se ocupa de lanzar y parar todos estos procesos, gestionar su

ejecución y decidir qué hacer en función de los recursos disponibles y de las órdenes dadas por el usuario.

Android lanza tantos procesos como permitan los recursos del dispositivo. Cada proceso, correspondiente a una aplicación, estará formado por una o varias actividades independientes. Cuando el usuario navega de una actividad a otra, o abre una nueva aplicación, el sistema duerme dicho proceso y realiza una copia del estado para poder recuperarlo más tarde.

Cada uno de los componentes básicos de *Android* cuenta con un ciclo de vida bien definido, lo que le permite al desarrollador controlar en cada momento que acciones tomar dependiendo del estado del componente. El ciclo de vida de una *Actividad* es presentado en la Figura 2.28.

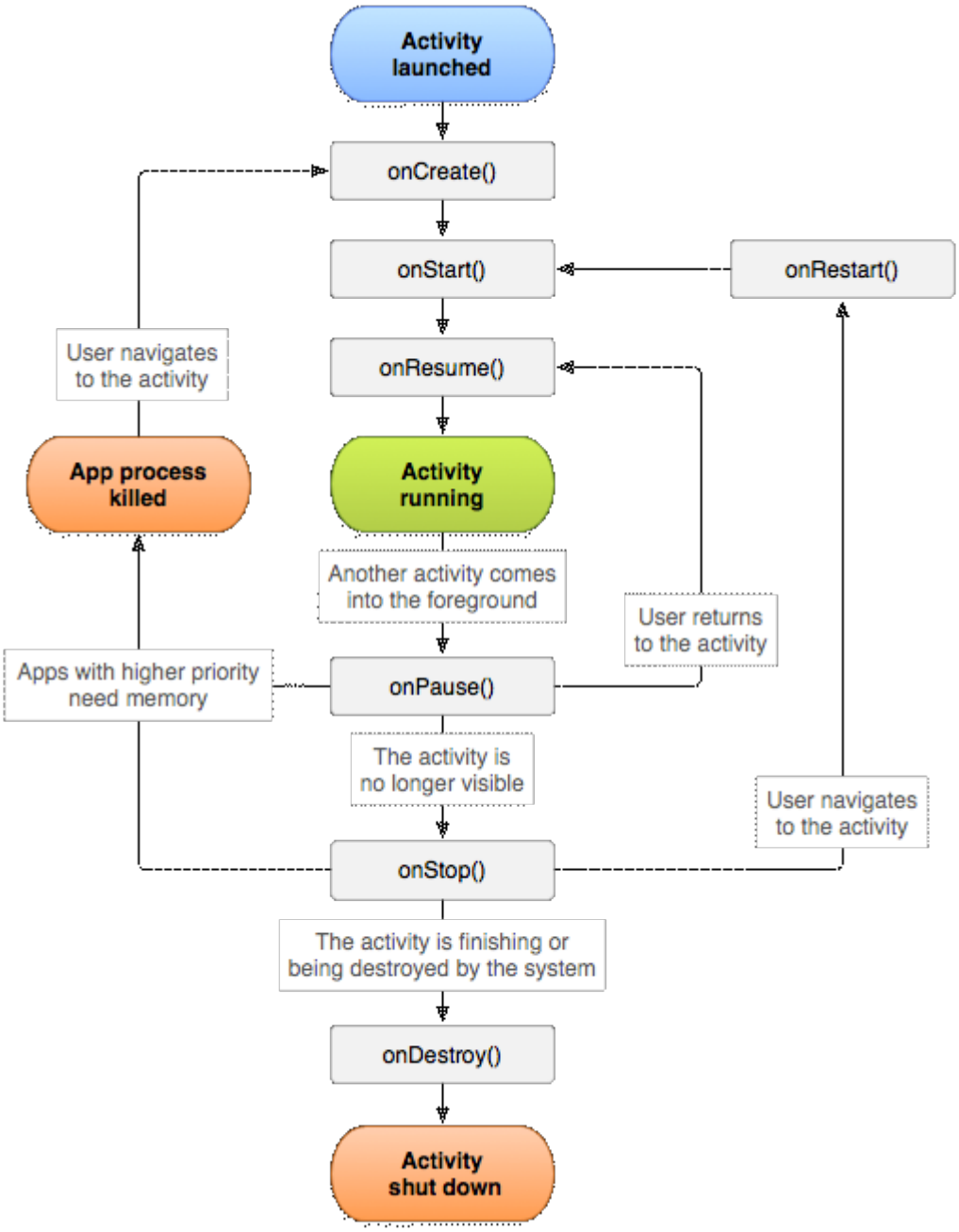

Figura 2.28.: Ciclo de vida del componente *Actividad* [Android].

Cuando una actividad entra y sale de los diferentes estados, esto se notifica a través de diferentes métodos `callback` que el desarrollador puede implementar para realizar operaciones cuando la actividad cambie de estado.

En conjunto, estos métodos definen el ciclo de vida completo de una actividad (como se mostró en la Figura 2.28). Al implementarlos, puedes monitorear tres bucles anidados en el ciclo de vida de la actividad, estos son:

- El *ciclo de vida completo* de una actividad transcurre entre la llamada a `onCreate()` y la llamada a `onDestroy()`. La actividad debe configurar el estado "global" (como la definición del diseño) en `onCreate()`, y liberar todos los recursos restantes en `onDestroy()`. Por ejemplo, si hay un subproceso de descarga de datos de la red en ejecución en segundo plano en la actividad, esta podría crear ese subproceso en el `onCreate()` y luego detenerlo en `onDestroy()`.
- El *ciclo de vida visible* de una actividad transcurre entre la llamada a `onStart()` y la llamada a `onStop()`. Durante ese tiempo, el usuario puede ver la actividad en pantalla e interactuar con ella. Por ejemplo, se llama a `onStop()` cuando se inicia una nueva actividad y esta ya no está visible. Entre estos dos métodos, puedes conservar los recursos necesarios para mostrarle la actividad al usuario. El sistema podría llamar a `onStart()` y `onStop()` muchas veces durante el ciclo de vida completo de la actividad, ya que la actividad pasa de ser visible y a estar oculta para el usuario.
- El *ciclo de vida en primer plano* de una actividad transcurre entre la llamada a `onResume()` y la llamada a `onPause()`. Durante ese tiempo, la actividad se encuentra al frente de todas las demás actividades en la pantalla y tiene el foco en la interacción del usuario. Con frecuencia, una actividad puede entrar y salir de primer plano. Dado que este estado puede cambiar con frecuencia, el código en estos métodos debe ser bastante liviano para evitar las transiciones lentas que hacen que el usuario deba esperar.

Acorde a lo descripto en la Figura 2.28, una *Actividad* puede tener básicamente tres estados:

- *Reanudada*

  La actividad se encuentra en el primer plano de la pantalla y tiene la atención del usuario. (A veces, este estado también se denomina "en ejecución").

- *Pausada*

  Otra actividad se encuentra en el primer plano y tiene la atención del usuario, pero esta todavía está visible. Es decir, otra actividad está visible por encima de esta y esa actividad es parcialmente transparente o no cubre toda la pantalla. Una actividad pausada está completamente "viva" (el objeto *Activity* se conserva en la memoria, mantiene toda la información de estado y miembro y continúa anexado al administrador de ventanas), pero el sistema puede eliminarla en situaciones en que la memoria sea extremadamente baja.

- *Detenida*

  La actividad está completamente opacada por otra actividad (ahora la actividad se encuentra en "segundo plano"). Una actividad detenida también permanece "viva" (el objeto *Activity* se conserva en memoria, mantiene toda la información de estado y miembro, pero *no* está anexado al administrador de ventanas). Sin embargo, ya no está visible para el usuario y el sistema puede eliminarla cuando necesite memoria en alguna otra parte.

Si se pausa o se detiene una *Actividad*, el sistema puede excluirla de la memoria al solicitarle que se cierre, o simplemente eliminando su proceso. Cuando se vuelve a abrir la *Actividad* (después de haber sido eliminada o destruida), es necesario crearla nuevamente.

2.2.3 Google Maps API's [GoogleMaps]

A continuación, se mencionan algunas API's de *Google Maps* para interactuar con información de geolocalización y que sirvieron de base para esta tesina.

- *Servicios web de Google Maps* [GoogleMaps]

  Estos servicios son un conjunto de interfaces para los distintos servicios de *Google Maps*, los cuales proporcionan datos geográficos para integrar en aplicaciones que deseen contar con los mismos a través de solicitudes HTTP. Por ejemplo, los servicios *Google Maps Geocoding API*[13]*, Google Maps Directions API*[14] y *Google Maps Javascript API*[15].

  Estos servicios fueron diseñados para geocodificar y calcular direcciones estáticas (ya conocidas) para la agregación de contenido de la aplicación en un mapa; para responder en tiempo real a las entradas del usuario (por ejemplo, dentro de un elemento de la interfaz de usuario). A continuación, se describen algunos detalles de algunos servicios de mapas que son de interés para esta tesina.

  - *Google Maps Geocoding API* [GoogleMaps]

    Cabe mencionar que la *geocodificación* es el proceso que convierte direcciones (como “1600 Amphitheatre Parkway, Mountain View, CA”) en coordenadas geográficas (como latitud 37,423021 y longitud -122,083739) que se pueden usar para disponer marcadores en un mapa o posicionar el mismo. La *geocodificación inversa* es el proceso de conversión de coordenadas geográficas en direcciones en lenguaje natural. El servicio de *geocodificación inversa* de *Google Maps Geocoding API* también permite buscar la dirección por un identificador de sitio determinado.

---

[13] Página de la API de *Google Maps Geocoding API*: https://developers.google.com/maps/documentation/geocoding/intro?hl=es-419 (Ultimo acceso 1/8/2017)

[14] Página de la API de *Google Maps Directions*: https://developers.google.com/maps/documentation/directions/intro?hl=es-419 (Ultimo acceso 1/8/2017)

[15] Página de la API de *Google Maps Javascript:* https://developers.google.com/maps/documentation/javascript/?hl=es-419 (Ultimo acceso 1/8/2017)

- *Google Maps Directions API* [GoogleMaps]

  Con este servicio se puede buscar indicaciones para diferentes medios o modos de transporte, incluido transporte público, manejo, desplazamiento a pie o en bicicleta. Las indicaciones pueden especificar sitios de origen, sitios de destino y waypoints (los cuales se podrían pensar como puntos intermedios) ya sea en forma de cadenas de texto (p. ej., "Chicago, IL" o "Darwin, NT, Australia") o como coordenadas de latitud/longitud. La *Google Maps Directions API* puede devolver indicaciones en múltiples partes mediante una serie de waypoints.

- *Google Maps JavaScript API* [GoogleMaps]

  Para responder a solicitudes dinámicas relativas a geolocalización, solicitudes del lado del cliente[16], se pueden usar los servicios brindados por las *API's JavaScripts*[17] de *Google Maps*.

- *Google Maps Android API* [GoogleMaps]

  Con la API para *Android*[18] de *Google Maps*, se puede agregar mapas basados en datos de *Google Maps* a la aplicación. La API administra en forma automática el acceso a servidores, descargas de datos, visualización de mapas y respuesta a gestos de mapas de *Google Maps*. También se puede usar llamadas a la API para agregar marcadores, polígonos y superposiciones a un mapa básico, y para cambiar la vista del usuario de modo que se muestre un área del mapa en particular. Estos objetos proporcionan información adicional de ubicaciones en el mapa y permiten la interacción del usuario con este.

  La API permite agregar los siguientes gráficos a un mapa:

  - Iconos anclados en posiciones específicas del mapa (marcadores).
  - Conjuntos de segmentos de líneas (polilíneas).
  - Segmentos cerrados (polígonos).
  - Gráficos de mapa de bits anclados en posiciones específicas del mapa (marcadores).
  - Conjuntos de imágenes que se muestran sobre los mosaicos de mapas básicos (superposiciones de mosaicos).

[16] lado del cliente o *cliente-side* (su denominación en inglés) hace referencia a código que se ejecuta, en este caso, en el navegador web del cliente.
[17] *JavaScript* (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScrip. Se utiliza principalmente en su forma del lado del cliente (client-side).
[18] Página de la API de *Google Maps para Android*:
https://developers.google.com/maps/documentation/android-api/intro (Último acceso 1/8/2017)

3. Modelo Propuesto

En este capítulo se realizará primero una descripción general de la problemática a resolver, acorde a esto se describirán los requerimientos funcionales juntos con los casos de uso identificados. Basándonos en estos casos de uso se realizará una presentación incremental del modelo propuesto. Luego, para brindar una mejor comprensión de este modelo se presentarán diferentes diagramas de secuencia de los casos de uso más relevantes.

3.1 Descripción de la problemática a resolver

Se desea lograr una solución de modelado para el seguimiento continuo del posicionamiento de entregas realizadas mediante repartos a domicilio. En particular, se busca mantener informada a cada una de las partes involucradas en cada entrega de un pedido: el cliente, el repartidor y el local de pedidos (negocio).

El cliente deberá conocer en todo momento (siempre que sea posible) el estado de su pedido, en particular, saber la posición en tiempo real del mismo (por ejemplo, en el negocio o en un lugar específico de la ciudad). Este estado determinará la información visible por parte del cliente y del repartidor.

En cuanto al local de pedidos, debe poder monitorear en tiempo real a sus repartidores, es decir, conocer la posición de cada uno de ellos, y en consecuencia también de los pedidos que el mismo debe entregar. Cada negocio podría establecer distintos tipos de estrategias de reparto. A partir de la estrategia elegida, se generará el recorrido tentativo que debería realizar el repartidor y establecerá así como será la distribución de cada pedido.

Los repartidores deben recibir la lista de pedidos que le fueron asignados para entregar, con información sobre los mismos, y el recorrido que debe realizar para entregar los mismos. Este recorrido se determina con la estrategia de reparto que establezca cada local. Cada uno de los repartidores debe informar en todo momento su posición actual con el fin de conocer respectivamente la posición de las entregas.

Además, se desea contar con un mecanismo con el cual se lleve un registro de los eventos que puedan surgir en cualquier momento del reparto de pedidos, estos deben ser detectados automáticamente por el sistema. Por ejemplo, el repartidor se desvía de la ruta asignada para el reparto.

Si bien el dominio descrito anteriormente puede involucra diferentes aspectos, para esta tesina nos focalizaremos solo en aquellos relacionados a la movilidad y el posicionamiento.

A continuación se identifican aquellos requerimientos funcionales[19] más relevantes para el foco de esta tesina. Es decir, aquellos que involucran el posicionamiento tanto de la entrega como del repartidor.

[19] Entendemos por requerimiento a una característica que se debe brindar o satisfacer. Muchas veces estos son provisto como servicios, estableciendo que entradas requieren y cuál será su comportamiento.

Cabe aclarar, que para los requerimientos funcionales, se denomina:

- *Orden,* a la orden de pedido que hace un cliente.
- *Entrega,* al conjunto de órdenes de pedido que debe entregar un repartidor

Los requerimientos funcionales identificados relacionados al posicionamiento son:

- *Establecer estrategia de entregas* (permite determinar la estrategia que se usará para calcular el recorrido que deberá realizar el repartidor)
- *Obtener recorrido de entrega* (permite obtener todos los domicilios a los que el repartidor deberá hacer entregas de pedidos)
- *Registrar rastro de repartidor* (registra información, por ejemplo, la posición actual del repartidos)
- *Registrar evento*
- *Obtener rastro de entrega* (permite determinar, por ejemplo, la posición actual de cada una de las órdenes de pedido. Puede ser que algunas de estos pedidos hayan sido entregados)
- *Obtener rastro de orden* (permite determinar, por ejemplo, el estado actual de una orden de pedido)
- *Obtener posición de orden*
- *Obtener posición de repartidor*

También se consideran los siguientes requerimientos que no involucran posicionamiento, pero están íntegramente relacionados y sirven para poder obtener información relevante acorde al foco de la tesina.

- *Asociar orden a repartidor*
- *Obtener listado de ordenes*
- *Obtener detalle de orden*

Cabe mencionar que los diagramas de casos de uso describen en forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario (actor), son descripciones de la funcionalidad del sistema independientes de la implementación, se centran en describir cómo es el diálogo entre el usuario y el sistema. De los requerimientos funcionales identificados anteriormente se plasmarán en casos de uso solo los siguientes: *Establecer estrategia de entregas, Asociar orden a repartidor, Obtener recorrido de entrega, Registrar rastro de repartidor, Obtener rastro de entrega, Obtener rastro de orden, Obtener listado de órdenes* y *Obtener detalle de orden*. Estos brindan un comportamiento no trivial, junto con un usuario (actor) que hace uso del mismo en el sistema. Por su trivialidad no se llevarán a casos de uso los requerimientos *Obtener posición de orden, Obtener posición de repartidor* y *Registrar evento* (este en particular, es una tarea automática que carece de relación con un usuario que haga uso del mismo).

En la Figura 3.1 se pueden apreciar aquellos casos de uso que fueron mencionados anteriormente y que son de interés para mostrar el funcionamiento del sistema que se desea lograr.

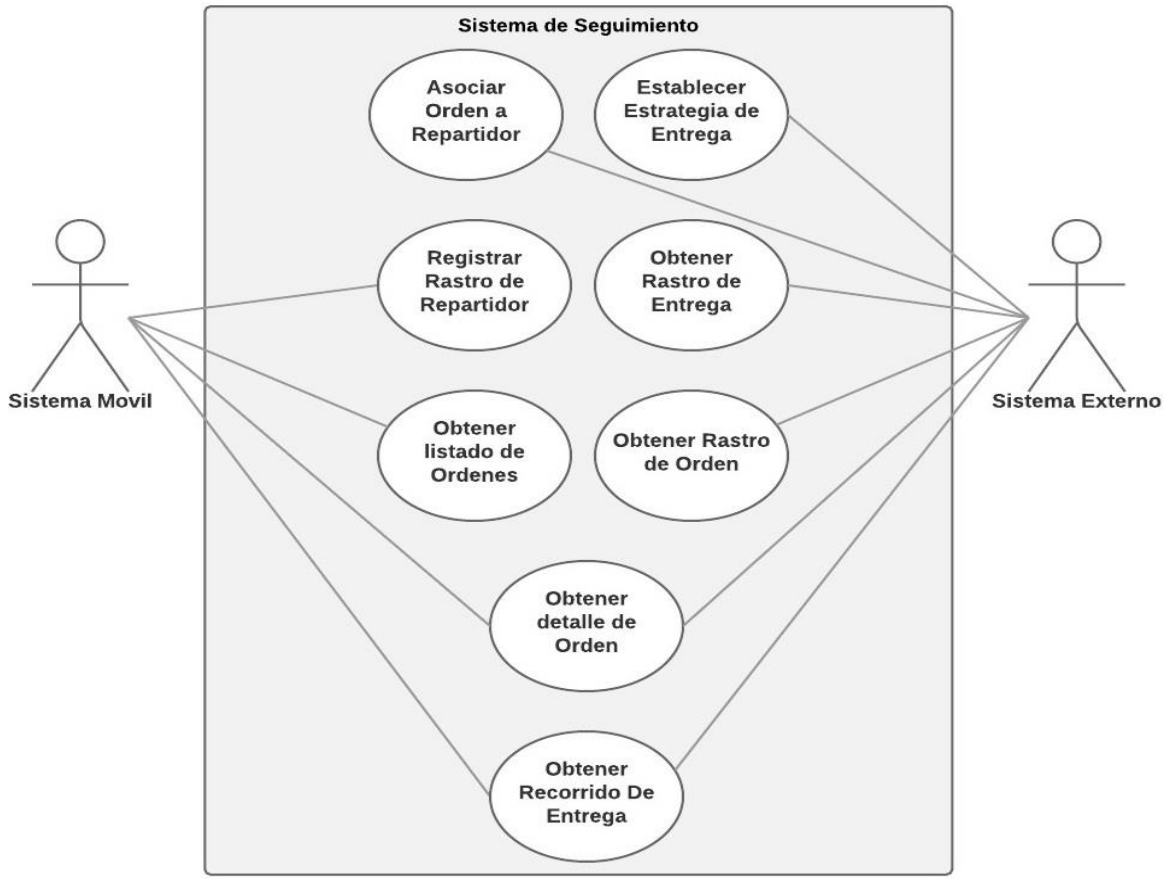

Figura 3.1: Casos de uso relacionado *seguimiento continuo de las entregas de pedidos*.

Se puede apreciar en la Figura 3.1, que los actores en este dominio son dos:

- *Sistema Externo:* se encarga de representar tanto al sistema que permite hacer las peticiones de órdenes de pedido por parte de los clientes y seguir el rastro que deja el repartidor para una orden, como también representa el sistema que permite administrar a los repartidores y seguir el rastro de entregas por parte de los locales de pedidos, incluso puede ser el mismo sistema pero con diferentes vistas que dependan de los permisos que tenga cada tipo de usuario.
- *Sistema Móvil:* representa a un usuario *repartidor,* el cual a medida que se va moviendo, su rastro y acciones son registradas por el *Sistema de Seguimiento*. Brinda una forma de indicar que el repartidor se encuentra activo a partir de un momento dado, es decir en horario laboral, con el fin de realizar las acciones previamente detalladas.

Para una mayor comprensión de los actores involucrados se presentan en la Tabla 3.1 más detalles de los mismos. Se puede apreciar que se detalla cada actor junto con algunas observaciones.

Tabla 3.1. Descripción de actores del diagrama de casos de uso.

Actor	Descripción	Observaciones
Sistema Externo	Este realiza la administración de *Negocios*, *Repartidores* y *Órdenes*, los cuales son necesarios para establecer la comunicación con el *Sistema de Seguimiento*. Tiene la capacidad de asociar un *Repartidor* a un *Negocio* y *Órdenes* a un repartidor. También puede obtener la posición del *Repartidor* y de la *Orden*, así como consultar el estado de la misma.	El *Sistema Externo* podría ser, por ejemplo, una aplicación cliente Web, que utiliza la funcionalidad del *Sistema de Seguimiento*.
Sistema Móvil	Es el encargado de abastecer de posiciones geográficas al *Sistema de Seguimiento*. Puede consultar *Órdenes* pertenecientes al *Pedido* y ver sus detalles, así como también puede comunicarse con el *Sistemas Externo*.	El *Sistema Móvil* podría ser, por ejemplo, una aplicación cliente móvil, específicamente diseñada para funcionar con el *Sistema de Seguimiento*. Por ejemplo, la aplicación móvil manda las posiciones al *Sistema de Seguimiento.*

3.2 Especificación de los casos de uso

En esta sección se especificarán los casos de uso relacionados al *Sistema de seguimiento*, presentados previamente en Figura 3.1. A continuación se detallan cada uno de estos.

- *Establecer estrategia de entregas*

  En las Tablas 3.2 se puede apreciar los datos generales del caso de uso denominado "*Establecer estrategia de entregas*", mientras que en la Tabla 3.3 se presenta el desarrollo del mismo.

Tabla 3.2. Datos generales del caso de uso "Establecer estrategia de entregas".

Datos Generales	
Descripción Contextual	Permite establecer la *Estrategia* con el que se van a organizar el *Recorrido* de las *Entregas* de un *Negocio*. Las estrategias pueden ser el camino más corto, más rápido, etc.
Actores Involucrados	▪ Sistema Externo
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ Ninguna.
Post-condiciones	▪ El Negocio establece satisfactoriamente la *Estrategia* de entregas deseada.
Observaciones	▪ Ninguna.

Tabla 3.3. Desarrollo del caso de uso "Establecer estrategia de entregas".

Desarrollo del Caso de Uso
Curso Normal
1. El *Sistema Externo* solicita establecer la Estrategia con la que se van a crear los *Recorridos* de las *Entregas* de un *Negocio*. 2. El *Sistema* establece la nueva configuración con la que se van a armar los *Recorridos* de las próximas *Entregas* para el *Negocio* que representa el *Sistema Externo*.

3. El *Sistema* valida si para el *Negocio* existen *Entregas* por en desarrollo. a. El *Sistema* almacena el *Recorrido* actual de la *Entrega* en el historial de recorridos de la misma. b. El *Sistema* vuelve a generar los *Recorridos* de las *Entregas* que están en desarrollo. 4. El *Sistema* confirma el cambio.

- *Obtener recorrido de entrega*

En la Tabla 3.4 se especificaron los datos generales del caso de uso denominado "*Obtener recorrido de entrega*". Mientras que en la Tabla 3.5 se detalla el desarrollo del mismo, donde se puede apreciar el flujo normal y el alternativo. Para simplificar la lectura se nombrará como "*actor*" a cualquiera de los posibles actores de este caso de uso que pueden ser tanto el *Sistema Móvil* como el *Sistema Externo*.

Tabla 3.4. Datos generales del caso de uso "Obtener recorrido de entrega".

Datos Generales	
Descripción Contextual	Permite obtener las *Posiciones* de los destinos de las *Órdenes* (es decir de la *Entrega*), estas posiciones definirán según la *Estrategia* de ordenamiento elegido por el *Negocio* el *Recorrido* sugerido que deberá seguir el *Repartidor*.
Actores Involucrados	▪ Sistema Externo, Sistema Móvil
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ La *Entrega* debe existir en el *Sistema*. ▪ Al menos una de las *Órdenes* de la *Entrega*, debe estar en estado enviado o en desarrollo.
Post-condiciones	▪ El actor, obtiene el *Recorrido* de la *Entrega*.
Observaciones	▪ Ninguna

Tabla 3.5. Desarrollo del caso de uso "Obtener recorrido de entrega".

Desarrollo del Caso de Uso
Curso Normal
1. Un actor solicita el *Recorrido* de una *Entrega*. 2. El *Sistema* verifica la existencia de la *Entrega*. 3. El *Sistema* verifica que al menos una *Orden* de la *Entrega* esté en estado enviado o en desarrollo. 4. El *Sistema* brinda el *Recorrido* recomendado para la *Entrega*.
Curso alternativo – No existe la Entrega correspondiente.
2.1 El *Sistema* no puede verificar la existencia de la *Entrega*. 2.2 El *Sistema* informa al actor que no existe la *Entrega*.
Curso alternativo – No hay Órdenes de la Entrega en estado enviado o en desarrollo.
3.1 El *Sistema* verifica que no existe *Orden* de la *Entrega* en estado enviado o en desarrollo. 3.2 El *Sistema* informa al actor que no puede brindar el *Recorrido* de la *Entrega*.

- *Registrar rastro de repartidor*

En la Tabla 3.6 se puede apreciar los datos generales del caso de uso denominado "*Registrar Rastro de Repartidor*". Mientras que en la Tabla 3.7 se detalla el desarrollo del mismo, donde se puede apreciar el flujo normal y el alternativo.

Tabla 3.6. Datos generales del caso de uso "Registrar rastro de repartidor".

Datos Generales	
Descripción Contextual	Permite guardar en el *Sistema* la o las *Posiciones* pendientes junto con la hora en que se registró cada una, es decir el *Rastro* del *Repartidor*. Debe validarse que el *Repartidor* esté activo antes de registrar la nueva Posición.
Actores Involucrados	▪ Sistema Móvil
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ El *Repartidor* debe estar dado de alta en el *Sistema*. ▪ El *Repartidor* debe pertenecer a un *Negocio*. ▪ El *Repartidor* debe estar activo.
Post-condiciones	▪ El *Rastro* del *Repartidor* se encuentra actualizado.
Observaciones	▪ Un *Repartidor* activo es aquel que está trabajando en el momento de realizar el registro.

Tabla 3.7. Desarrollo del caso de uso "Registrar rastro de repartidor".

Desarrollo del Caso de Uso
Curso Normal
1. El *Sistema Móvil* envía una lista con el *Rastro* pendiente de ser registrado. 2. El *Sistema* valida que el *Repartidor* esté activo. 3. El *Sistema* registra en el *Repartidor* su último *Rastro*. 4. El *Sistema* registra en la *Entrega* la lista enviada por el *Sistema Móvil*. 5. El *Sistema* confirma el registro del *Rastro* del *Repartidor*.
Curso alternativo – Repartidor no se encuentra activo.
2.1. El *Sistema* detecta que el *Repartidor* no se encuentra activo. 2.2. El *Sistema* informa que no puede guardar el *Rastro*.

- *Obtener rastro de entrega*

En la Tabla 3.8 y 3.9 se especifican los datos generales y el desarrollo del caso de uso denominado "*Obtener rastro de entrega*".

Tabla 3.8. Datos generales del caso de uso "Obtener rastro de entrega".

Datos Generales	
Descripción Contextual	Permite obtener el *Rastro* registrado por un *Repartidor* para una *Entrega* en estado enviado. Con el fin de que se pueda saber en tiempo real, en donde se encuentra una *Entrega* en particular.
Actores Involucrados	▪ Sistema Externo
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ El *Sistema Externo* debe estar asociado a la *Entrega*. ▪ La *Entrega* debe tener al menos una *Orden* en estado enviado.
Post-condiciones	▪ El *Sistemas Externo* obtiene el *Rastro* de una *Entrega*.
Observaciones	▪ Ninguna

Tabla 3.9. Desarrollo del caso de uso "Obtener rastro de entrega".

Desarrollo del Caso de Uso
Curso Normal
1. Un *Sistema Externo* solicita el *Rastro* de una *Entrega*. 2. El *Sistema* verifica la existencia de la *Entrega*. 3. El *Sistema* verifica que la *Entrega* tenga al menos una *Orden* en estado enviado. 4. El *Sistema* brinda el *Rastro de la Entrega*.
Curso alternativo – No existe la Entrega correspondiente.
2.1 El *Sistema* no puede verificar la existencia de la *Entrega*. 2.2 El *Sistema* informa al *Sistema Externo* que no existe la *Entrega*.
Curso alternativo – La Entrega está en estado en desarrollo, finalizado, cancelado o suspendido.
3.1 El *Sistema* verifica que el estado de la *Entrega* no es enviado. 3.2 El *Sistema* informa al *Sistema Externo* que no puede brindar el *Rastro* de la *Entrega*.

- *Obtener rastro de orden*

En la Tabla 3.10 se puede apreciar los datos generales del caso de uso denominado "*Obtener rastro de orden*", éste puede verse como un subconjunto del rastro de la entrega a la que pertenece la orden. Mientras que en la Tabla 3.11 se detalla el desarrollo del mismo, donde se puede apreciar el flujo normal y el alternativo.

Tabla 3.10. Datos generales del caso de uso "Obtener rastro de orden".

Datos Generales	
Descripción Contextual	Permite obtener el *Rastro* registrado por un *Repartidor* para una *Orden* en particular que se encuentre en estado enviado. Con el fin de que se pueda saber en tiempo real, en donde se encuentra una *Orden* específica a lo largo de su trayecto.
Actores Involucrados	▪ Sistema Externo
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ El *Sistema Externo* debe estar asociado a la *Orden*. ▪ La *Orden* debe existir en el *Sistema*. ▪ El *Rastro* de una *Orden* puede ser visto mientras que el estado de la misma no sea un estado final.
Post-condiciones	▪ El *Sistemas Externo* obtiene el *Rastro* de la *Orden*.
Observaciones	▪ Ninguna

Tabla 3.11. Desarrollo del caso de uso "Obtener rastro de orden".

Desarrollo del Caso de Uso
Curso Normal
1. Un *Sistema Externo* solicita el *Rastro* de una *Orden*. 2. El *Sistema* verifica la existencia de la *Orden*. 3. El *Sistema* verifica que el estado de la *Orden* no es un estado final. 4. El *Sistema* brinda el *Rastro de la Orden*.
Curso alternativo – No existe la Orden correspondiente.
2.1 El *Sistema* no puede verificar la existencia de la *Orden*. 2.2 El *Sistema* informa al *Sistema Externo* que no existe la *Orden*.
Curso alternativo – La Orden está en estado en finalizado o cancelado.
3.1 El *Sistema* verifica que el estado de la *Orden* es un estado final. 3.2 El *Sistema* informa al *Sistema Externo* que no puede brindar el *Rastro* de la *Orden*.

- *Asociar orden a repartidor*

  En la Tabla 3.12 y 3.13 puede verse los datos generales y el desarrollo del caso de uso denominado "*Asociar Orden a Repartidor*" respectivamente.

Tabla 3.12. Datos generales del caso de uso "Asociar orden a repartidor".

Datos Generales	
Descripción Contextual	Permite asociar una *Orden* existente a un *Repartidor* activo.
Actores Involucrados	▪ Sistema Externo
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ El *Repartidor*, la *Orden* y el *Negocio* deben existir.
Post-condiciones	▪ El *Repartidor* tiene una nueva *Orden* en su lista de órdenes y la *Orden* tiene una asociación con el *Repartidor*.
Observaciones	▪ Un *Repartidor* activo es aquel que está trabajando en el momento de realizar la asociación.

Tabla 3.13. Desarrollo del caso de uso "Asociar orden a repartidor".

Desarrollo del Caso de Uso
Curso Normal
1. El *Sistema Externo* selecciona una *Orden* y un *Repartidor* para asociarla. 2. El *Sistema* valida que el *Repartidor* pertenezca al *Negocio* dueño de la *Orden.* 3. El *Sistema* verifica que el Repartidor esté activo. 4. El *Sistema* asocia la *Orden* al *Repartidor*. a. El *Sistema* almacena el *Recorrido* de la *Entrega* actual en el historial de recorridos de la misma. b. El Sistema vuelve a generar el Recorrido de la *Entrega* que está en desarrollo. 5. El *Sistema* confirma la operación.
Curso alternativo – Repartidor no se encuentra activo.
3.1. El *Sistema* verifica que el *Repartidor* seleccionado para asociar la *Orden* no se encuentra activo. 3.2. El *Sistema* informa que no se puede asociar la *Orden* al *Repartidor*.

- *Obtener listado de órdenes*

  En la Tabla 3.14 se puede apreciar los datos generales del caso de uso denominado "*Obtener listado de órdenes*".

Tabla 3.14. Datos generales del caso de uso "Obtener listado de órdenes".

Datos Generales	
Descripción Contextual	Permite obtener un listado de las órdenes que tiene asignadas el *Repartidor*. Las órdenes se muestran ordenadas según la *Estrategia* de entregas.
Actores Involucrados	▪ Sistema Móvil
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ El *Repartidor* debe tener al menos una *Orden* asignada.
Post-condiciones	▪ El *Repartidor* obtiene una lista de las órdenes que tiene asignadas.
Observaciones	▪ Ninguna.

- *Obtener detalle de orden*

  En la Tabla 3.15 se puede apreciar los datos generales del caso de uso denominado "*Obtener detalle de orden*". En la Tabla 3.16 se detalla el desarrollo del mismo, donde se puede apreciar el flujo normal y el alternativo. Para simplificar la lectura en el caso de uso se usará "actor" para hacer referencia a que el que realizar dicha acción en el sistema puede ser el *Sistema Externo* como al *Sistema Móvil*.

Tabla 3.15. Datos generales del caso de uso "Obtener detalle de orden".

Datos Generales	
Descripción Contextual	Permite consultar el detalle de una *Orden*. Dependiendo del estado de la misma (En desarrollo, enviada, cancelada, finalizada o suspendida), y el actor involucrado, serán los campos que se puedan obtener.
Actores Involucrados	▪ Sistema Externo, Sistema Móvil
Relaciones	
Extiende	
Usa	
Proceso	
Pre-condiciones	▪ La *Orden* debe estar en el *Sistema*.
Post-condiciones	▪ El actor obtiene el detalle de una *Orden*.
Observaciones	▪ Ninguna

Tabla 3.16. Desarrollo del caso de uso "Obtener detalle de orden".

Desarrollo del Caso de Uso
Curso Normal
1. El actor solicita el detalle de una *Orden*. 2. El *Sistema* verifica la existencia de la *Orden*. 3. El *Sistema* envía el detalle de la *Orden*. a. Si el estado es en desarrollo, se envía la posición del local, la dirección de entrega, fecha y hora en la que inició el pedido. b. Si el estado es enviado, se envía la dirección de entrega, fecha y hora en la que inició el pedido. c. Si el estado es otro, se informa el porqué de dicho estado, la dirección de entrega, la fecha y hora en la que finalizó la entrega. 4. El actor recibe satisfactoriamente el detalle de la orden solicitada.
Curso alternativo – No existe la orden correspondiente.
2.1 El *Sistema* no puede verificar la existencia de la *Orden*. 2.2 El *Sistema* informa al actor que no existe la *Orden*.

3.3 Descripción del Modelo propuesto

Analizando los casos de usos presentados en la Sección 3.2, se identificaron algunos conceptos relevantes que a simple vista se tienen que modelar. En la Tabla 3.17 se puede apreciar los conceptos a modelar identificados en cada caso de uso.

Tabla 3.17. Conceptos a modelar a partir de los casos uso.

Caso de Uso	Conceptos a modelar
Establecer estrategia de entregas	*Entrega, Estrategia de entrega, Negocio*
Asociar Orden a Repartidor	*Repartidor, Orden, Negocio*
Obtener recorrido de entrega	*Posición, Orden, Entrega, Negocio, Estrategia de entrega, Recorrido, Estado de orden*

Caso de Uso	Conceptos a modelar
Registrar rastro de repartidor	*Rastro, Repartidor, Negocio*
Obtener rastro de entrega	*Rastro, Repartidor, Posición, Entrega, Negocio, Estado de orden*
Obtener rastro de orden	*Rastro, Repartidor, Posición, Entrega, Negocio, Orden, Estado de orden*
Obtener listado de órdenes	*Orden, Repartidor, Entrega, Estrategia de entrega*
Obtener detalle de orden	*Orden, Estado de orden*

Se puede apreciar en la Tabla 3.17 que algunos de los conceptos identificados se repiten. En resumen, se detectaron los siguientes conceptos a modelar:

- *Entrega (conjunto de órdenes)*
- *Estrategia de Entrega*
- *Orden (junto con su estado)*
- *Negocio*
- *Repartidor*
- *Posición*
- *Recorrido*
- *Rastro*

También se identificó el concepto de "*Evento*" aunque no se lo relaciona con un caso de uso. Estos conceptos sirvieron de base para plantear el modelo propuesto. A continuación, se realizará una descripción incremental de nuestro modelo, con el fin de proporcionar un mayor entendimiento del mismo.

Se comienza con la descripción de una clase encargada de intervenir o gestionar la mayor parte de las funcionalidades del modelo, esta clase se denomina *Negocio* (como se puede ver en la Figura 3.2). Es el punto de entrada al sistema y a partir de ella se desencadena comportamiento que involucra a otras clases importantes del modelo. Esta clase actúa dentro del modelo respetando el patrón de diseño "*Facade*" [Gamma et al, 1995], es decir, es el punto de entrada.

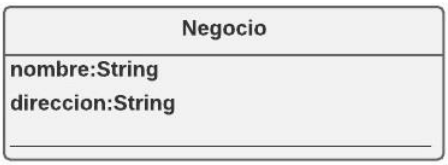

Figura 3.2: Modelado del *negocio*.

Se puede apreciar en la Figura 3.2, que el *negocio* cuenta con un conjunto de atributos básicos que pueden ser extendidos por un sistema que use sus servicios, esto nos facilita prescindir de, por ejemplo, el manejo de usuarios o alguna otra funcionalidad que no está acorde al foco de esta tesina.

Cada *negocio* debe contar con un conjunto de trabajadores. Para realizar esta representación del trabajador o encargado del transporte de pedidos, se incorpora el concepto de *Repartidor* como se puede observar en Figura 3.3. Se puede apreciar que un *repartidor* cuenta con un conjunto de características básicas que pueden ser extendidas. Un *repartidor* puede trabajar para uno o más negocios.

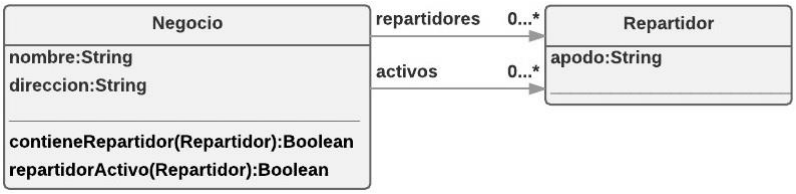

Figura 3.3: Representación del repartidor.

Se debe poder brindar a un *negocio* la posibilidad de administrar los pedidos que deben llegar a destino, para lo cual se introdujo la clase *Orden*. Esto se puede apreciar en la Figura 3.4. Al igual que en los casos anteriores, la *orden* cuenta con un conjunto de características básicas que pueden ser extendidas.
Se puede apreciar en la Figura 3.4 que la *orden* tiene definido el método `obtenerDetalle()`, este método es el encargado de mostrar la información de la *orden* según el *estado* en el que se encuentre; los estados se describirán más adelante.

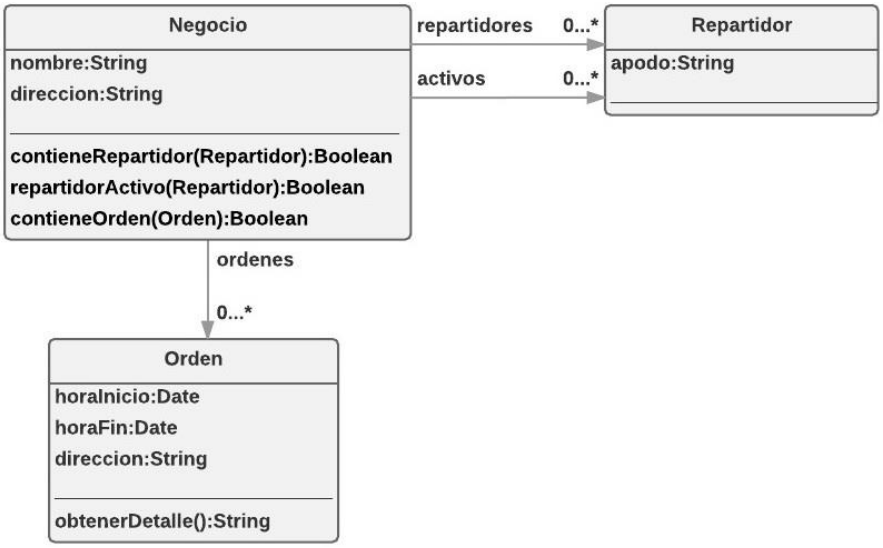

Figura 3.4: Modelado de la orden.

Ante la necesidad de representar un conjunto de órdenes que debe entregar cada repartidor se introduce el concepto de *Entrega* como se puede apreciar en la Figura 3.5. Esta es la clase de quién se va a realizar la trazabilidad. Cada *entrega* debe pertenecer a un *repartidor*, que es el encargado de llevar a destino cada una de las órdenes involucradas en dicha *entrega*.

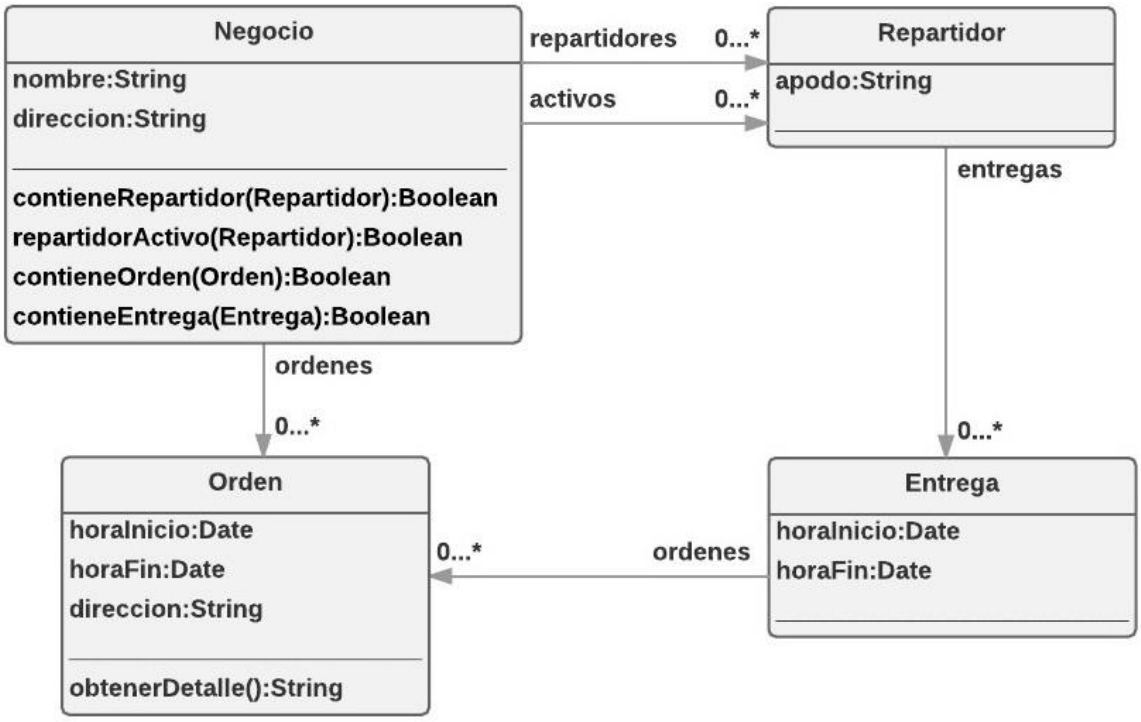

Figura 3.5: Representación de la entrega.

En cada una de las etapas del proceso que va desde la realización del pedido hasta la *entrega* del mismo, una *orden* (de pedido) no permanece invariante, por lo que se necesita representar los diferentes estados asociados a la misma. Con la finalidad de reflejar estos posibles estados, se incorpora una clase que represente el *estado* de la *orden* *EstadoDeOrden*. Este concepto permite describir el *estado* en el que se encuentra una *orden*, junto con el comportamiento que tiene la misma para dicho *estado*. Esta clase cuenta con las siguientes subclases:

- *Desarrollo* (la *orden* está siendo procesada, aun no salió del local)
- *Enviado* (la *orden* ya se encuentra en camino)
- *Cancelado* (la *orden* se canceló y no será entregada)
- *Finalizado* (la *orden* se entregó)
- *Suspendido* (la *orden* se encuentra suspendida momentáneamente)

Los estados de la *orden* respetan el patrón de diseño “*State*” [Gamma et al, 1995]. Esto se puede apreciar en la Figura 3.6.

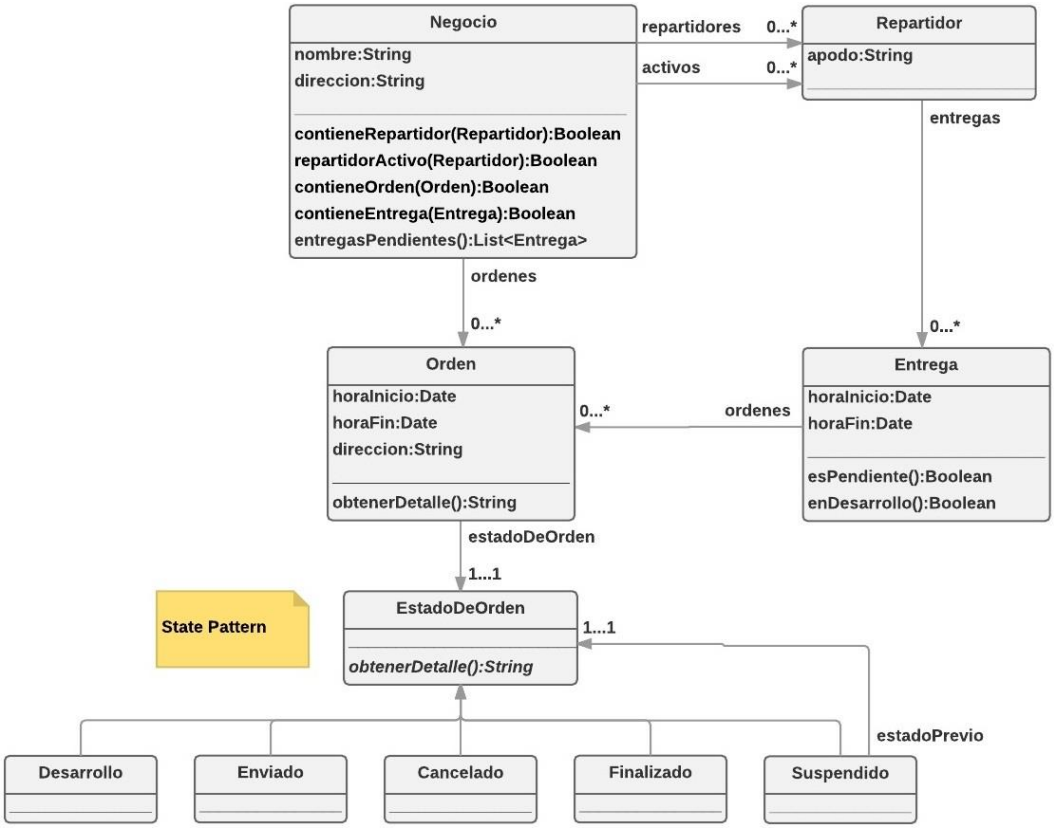

Figura 3.6: Representación de los estados de la orden.

En la Figura 3.7 se muestra un diagrama *DTE* (*Diagrama de Transición de Estados*[20]) referente a los estados que puede tomar una *orden*.

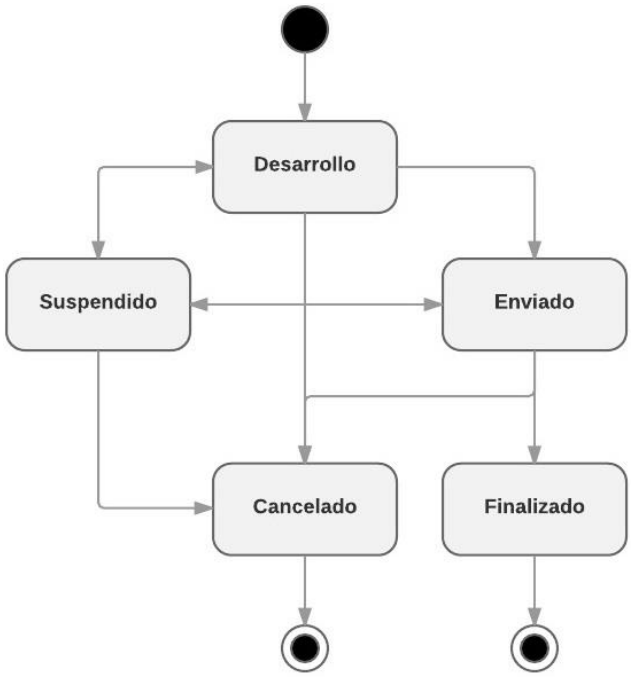

Figura 3.7: Diagrama de transición de estados de la orden

[20] Un diagrama de transición de estados (también conocido como DTE) muestra el comportamiento dependiente del tiempo de un sistema de información. Representa los estados que puede tomar un componente o un sistema y muestra los eventos que implican el cambio de un estado a otro.

Veamos cómo se realiza la secuencia de cambio de *estado* a medida que la *orden* pasa de ser creada a entregada, acorde a las indicaciones de la Figura 3.7. Cuando se crea una *orden* se le asocia una instancia de *estado* en *Desarrollo*, esto indica que la *orden* ingreso al sistema y está siendo procesada. Una vez instanciado el *estado* como en *Desarrollo*, el mismo puede cambiar a *Cancelado*; esto puede suceder si por algún motivo se desea dar de baja la *orden* para que finalice sin éxito. Del *estado Desarrollo*, también puede pasar al *estado Enviado*, haciendo referencia a que la *orden* ya fue procesada y está en manos de un *repartidor* para su envío. Asimismo, tanto del *estado Desarrollo* como del *Enviado* se puede pasar al *estado Suspendido* si por algún motivo se desea suspender momentáneamente la *orden*, pausándola ante algún contratiempo hasta que se resuelva como continuar. Estando en este *estado* (*Suspendido*), se podrá volver su *estado* previo. Por otra parte, una vez la *orden* pasó por el *estado Enviado*, la misma podrá pasar al *estado Finalizado* indicando que dicha *orden* fue concretada con éxito; o podría pasar al estado *Cancelado*.

Al momento de hacer uso de nuestro modelo se debe tener en cuenta que un *repartidor* activo es aquel que se encuentra trabajando en un instante de tiempo dado. Cuando un repartidor tiene una *entrega* activa, esto es, con al menos una *orden* en *estado* de *Enviado* o *Suspendido*, no puede tener otra de las entregas asignadas activa. Además, las entregas se encuentran asignadas a un *repartidor* en secuencia ordenada, por lo que cuando una de las *entrega* finaliza, la siguiente es activada automáticamente. Cuando una *entrega* se activa, todas sus órdenes pasan a *estado* enviado. Una *entrega* finaliza cuando todas sus órdenes pasan a uno de los estados finales (*Cancelado* o *Finalizado*). Las órdenes en *estado Suspendido* que bloqueen la finalización de una *entrega*, deberán cancelarse manualmente aclarando el motivo de la cancelación.

Para representar la posición geográfica de una *entrega* en un determinado instante de tiempo, se incorporó la clase *Rastro* que alude a un *rastro* de migajas o lugares por los que pasó un *repartidor* en su *recorrido*. Junto a ésta se añadió la clase *Posición* que hace referencia una *posición* geográfica (esto se puede apreciar en la Figura 3.8). Cabe aclarar que la clase *posición* es una representación general de una *posición* geográfica y en este sistema es usada por cualquier elemento que quiera representar su *posición*, como por ejemplo la *posición* destino, o la *posición* origen, mientras que el *rastro* surge por la necesidad de registrar no solo posiciones sino que además es relevante saber en qué instante de tiempo se estuvo en esa *posición*. Un *repartidor* sólo tiene la *posición* en que se encuentra en el instante de tiempo actual, es decir, su último *rastro*. En la Figura 3.8 se puede apreciar que el *rastro* también es usado por la *entrega,* de esta manera, queda registrada cada una de las posiciones asociadas a la misma en distintos instantes de tiempo.

Como se puede apreciar en la Figura 3.8, la o*rden* tiene su dirección de *entrega* en formato de texto, pero también cuenta la *posición* geográfica. La primera es utilizada para que el usuario del sistema tenga el detalle del destino, mientras que la segunda es usada internamente en el sistema para calcular los recorridos. En este caso como la *orden* tiene una *posición* de destino que no va a cambiar se puede relacionar dicha clase en forma directa con la *posición*. En el caso del *negocio*, se puede apreciar que también cuenta con

una posición textual y otra posición geográfica (posicónDeDireccion), ésta de igual forma que en la *orden* es usada para calcular los recorridos.

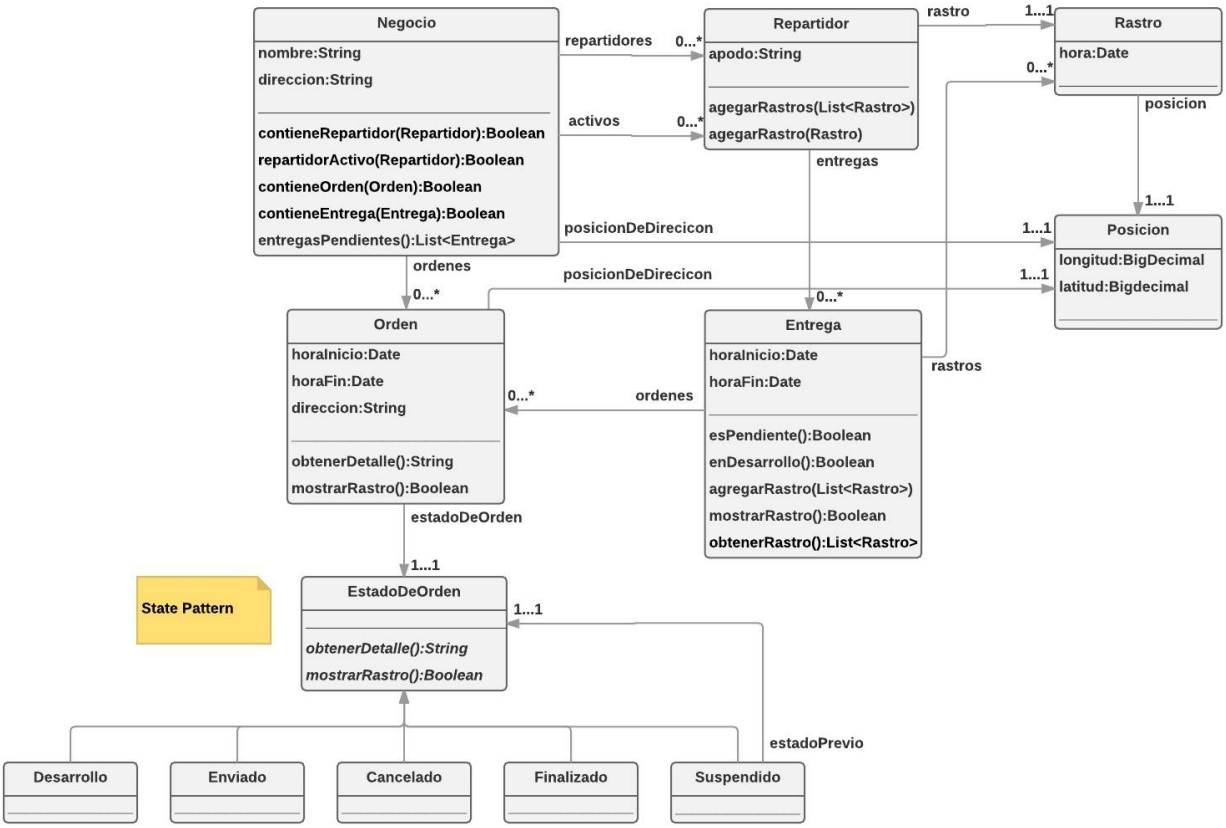

Figura 3.8: Representación de las posiciones.

Como se dijo antes, un *repartidor* debe contar con su último *rastro*, con el fin de tener un conocimiento permanente de donde se encuentra el mismo. De manera similar, una *entrega* debe contar con el *rastro* del *recorrido* que realizó el *repartidor* para dicha *entrega*, ya que este no necesariamente va a coincidir con el *recorrido* estipulado por la estrategia elegida en el sistema. Cada vez que se registre una nueva *posición* de un *repartidor* activo en un instante de tiempo (instancia de la clase *Rastro*), ésta deberá ser añadida a su registro y a la *entrega* que éste posea asociada con al menos una *orden* en *estado* enviado.

Puede ocurrir que un *repartidor* no pueda comunicar al sistema sus posiciones durante un determinado tiempo. Cuando finalmente esto se logra, el *repartidor* solo almacena el último *rastro* registrado, enviando todos aquellos que estaban en espera a que se añadan a la lista de rastros de la *entrega*, por lo que se debe brindar la posibilidad de incorporar a la *entrega* este conjunto de rastros remanentes.

Con el fin de modelar el *recorrido* recomendado por la estrategia elegida por el *negocio* que debería realizar el *repartidor* para llevar a cabo las entregas, se decidió incorporar la clase *Recorrido* como se puede observar en la Figura 3.9. Esta clase representa el camino a seguir por las posiciones que la definen. Como se puede apreciar la *entrega* tiene dos relaciones con la clase *recorrido*, una llamada “*recorrido*” y otra llamada “*historial*”. La primera permite modelar el *recorrido* que se le sugiere realizar el *repartidor*, y está determinado por las direcciones destino de las órdenes contenidas en una *entrega* y se

recalculará cada vez que una *orden* sea cancelada, ya que esa *posición* destino no será válida, y se la deberá sacar del *recorrido*. En este *recorrido* se especifican las posiciones por las que debe pasar el *repartidor* para entregar las distintas órdenes. La segunda relación permite guardar un historial de recorridos con el fin de poder realizar a posterioridad mediciones para poder optimizar el sistema. A modo de ejemplo, nos permite saber cuántas veces se tuvo que recalcular el *recorrido* para una *entrega.*

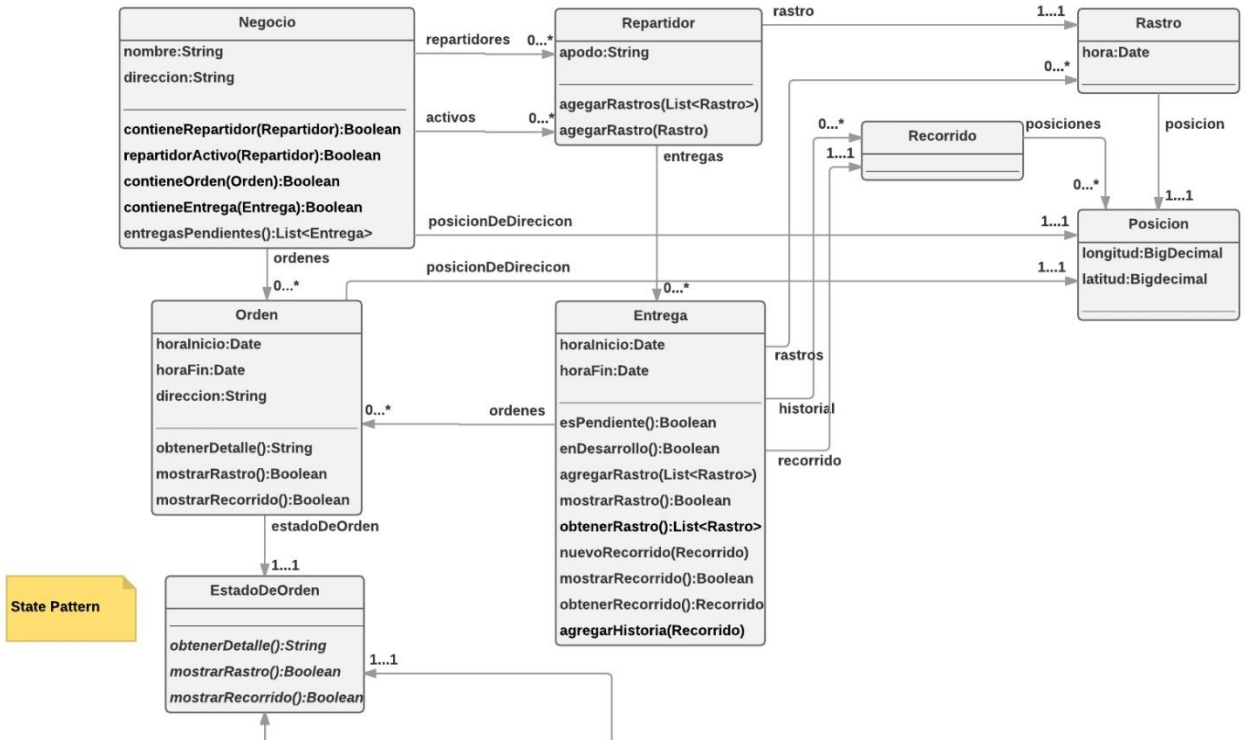

Figura 3.9: Representación de los recorridos.

Cabe mencionar que la *entrega* va a ir registrando las posiciones en cada instante de tiempo en la clase *rastro*, las cuales pueden ser diferentes de las definidas por la estrategia en el "*recorrido"* de la *entrega*. Supongamos que el repartidor se desvía del recorrido recibido, de esta manera se pueden registrar posiciones del recorrido real de la entrega.

Para que cada *negocio* pueda elegir el *orden* o estrategia con el que se van a realizar las entregas, es decir, la forma en la que se van a formar los recorridos, se creó una jerarquía de estrategias llamada *EstrategiaDeRecorrido* las cuales se pueden apreicar en la Figura 3.10. La jerarquía de estrategias respeta el patrón de diseño "*Strategy*" [Gamma et al, 1995]. Se definieron para este modelo tres estrategias: *Algoritmo de Dijkstra, Camino Euleriano y Algoritmo de Kruskal.* La estrategia de "*Dijkstra*" consiste en la obtención de los caminos más cortos entre un nodo origen y todos los demás nodos del grafo, permitiendo de esta manera ordenar los destinos con el fin de obtener el camino más corto, acorde a lo mencionado en [Dijkstra, 1959]. En el caso del "*Camino Euleriano*" [Euler, 1736] lo que se busca es encontrar un camino en el que se pase por cada dirección destino una sola vez hasta volver a la dirección del *negocio* o dirección origen, este caso se lo podría ver como un camino que permite optimizar el uso de combustible. Por otro lado, el "*Algoritmo Kruskal*" [Kruskal, 1956] tiene como propósito la construcción de un árbol de mínimo recubrimiento, por lo tanto, arma un subgrafo de aristas, formando un árbol de coste mínimo que incluya a todos los nodos del grafo, este caso puede ser comparado con el "*Algoritmo de Dijkstra*", la

diferencia se encuentra en la complejidad de ejecución, siendo Kruskal el de mayor eficiencia. También se podrían utilizar estas implementaciones, añadiéndoles información geográfica en tiempo real, de manera en la que el ordenamiento se realice con el fin de obtener el camino considerando información de tránsito, reparaciones de calles, etc. El modelo es extensible para agregar otro tipo de estrategias que utilice otros algoritmos.

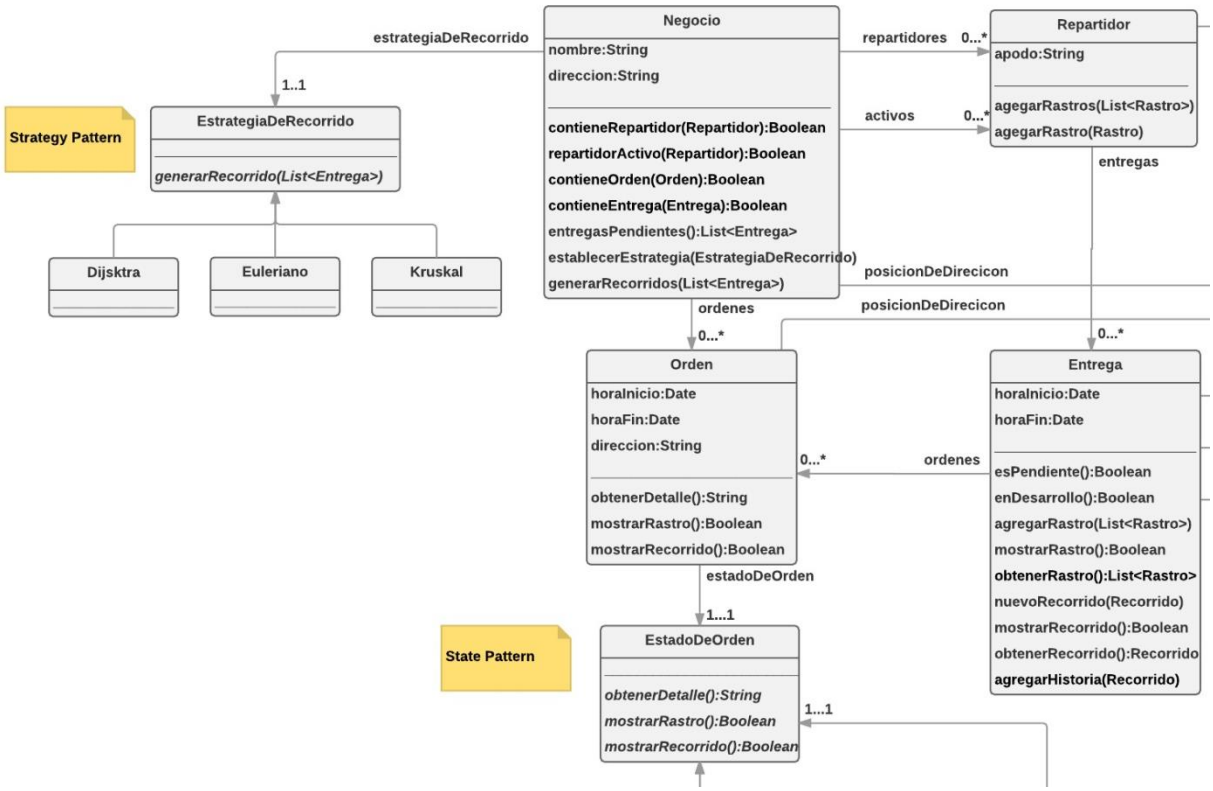

Figura 3.10: Estrategia para calcular recorridos.

Cada vez que el *negocio* decida cambiar la estrategia con la que se van a calcular los recorridos de las entregas, se cancele, suspenda o se vuelva a activar una *orden* suspendida, se deberá generar nuevamente el *recorrido* con las órdenes involucradas de la *entrega*. El método encargado de generar este *recorrido* es `generarRecorrido()` perteneciente al *negocio*, este método se encarga de pedirle a su estrategia que realice el ordenamiento de las posiciones destino. Previendo la necesidad de registrar eventos que ocurran dentro del *recorrido* de una *entrega* en una *posición* particular, se decidió incorporar la clase *Evento* (ver Figura 3.11). Un ejemplo de ello sería el no recibir posiciones que puede darse donde el área de cobertura del servicio de telefonía es deficiente o nulo, o detectar que el *repartidor* a estado enviando posiciones de una misma área por un tiempo determinado. Estos tipos de eventos son detectados por el sistema y opcionalmente etiquetados con posterioridad. Permiten tener una descripción más detallada de lo que ocurre cuando se realiza el *recorrido* de la *entrega*. Ya que en algunos de estos eventos ocurridos se necesitará especificar el lapso de tiempo que duró el mismo (por ejemplo, al detectar que el *repartidor* a estado enviando posiciones de una misma área por un tiempo determinado), se incorpora la clase *EventoTemporal*, la cual cuenta con la hora en la que se comenzó el *evento* y la duración del mismo.
Se puede apreciar en la Figura 3.11 el modelo final resultante para el seguimiento continuo de entrega de pedidos.

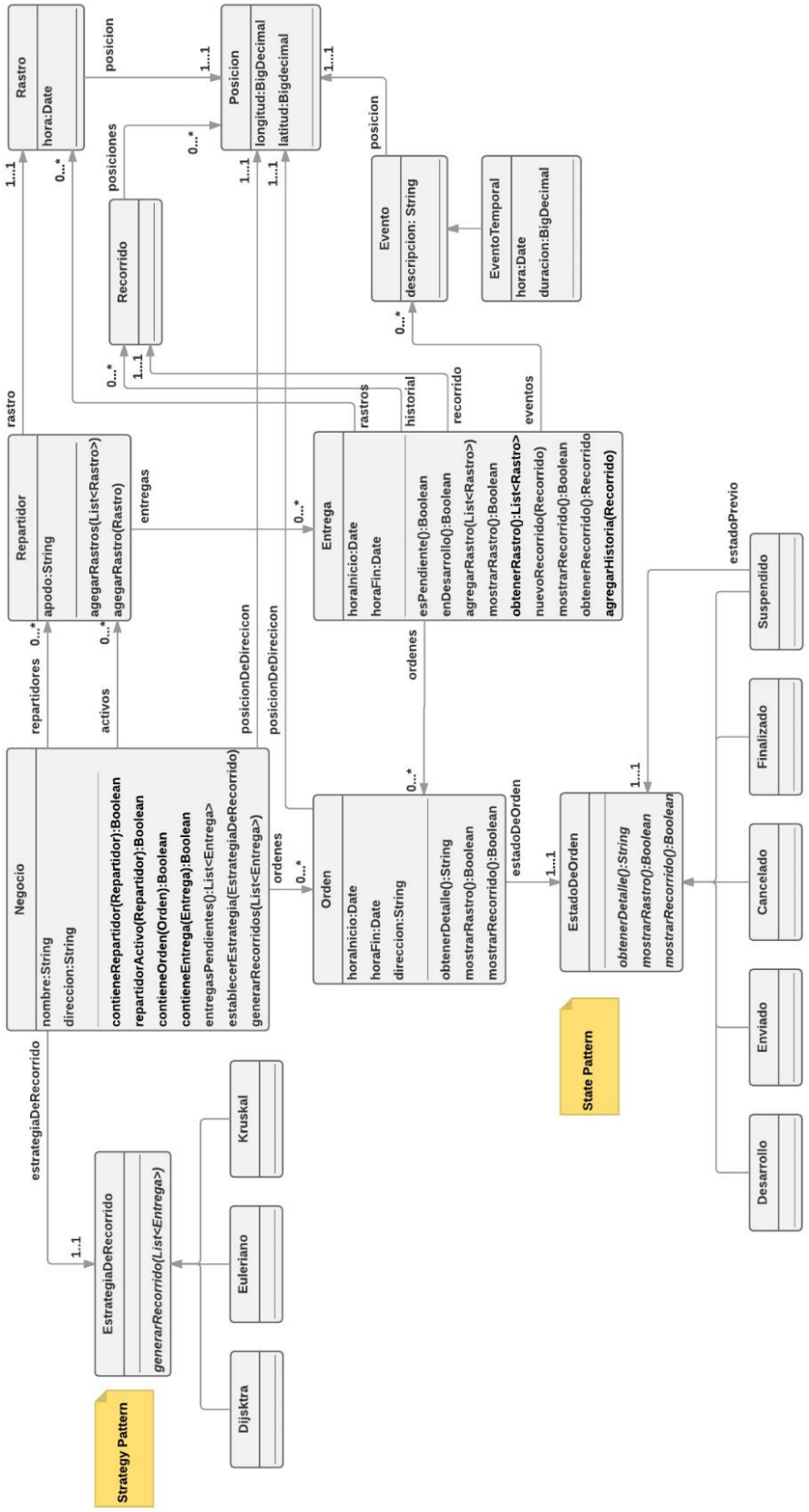

Figura 3.11: Modelo Propuesto.

3.4 Funcionalidad del Modelo Propuesto

En esta sección se presentan los diagramas de secuencia del modelo propuesto, en base a los casos de uso de mayor relevancia. Estos diagramas se van a presentar en el mismo orden en el que fueron presentados los casos de uso en la Sección 3.2.

Antes de comenzar con los diagramas, existen dos aclaraciones que aplican a todos ellos. Si bien en algunos diagramas se utilizó la instancia concreta de estado enviado, cualquier instancia de las subclases de *EstadoDeOrden* responden a los mismos requerimientos, pero comportándose de manera diferente. De la misma forma, vale aclarar que en los diagramas que se utilizó la instancia concreta de estrategia *dijkstra*, igual que en el caso anterior, todas las instancias de las subclases de *EstrategiaDeRecorrido* responden a los mismos mensajes, pero con otro comportamiento.

Dado que los diagramas ocupan toda una hoja, se describen algunos de ellos en forma consecutiva y luego son presentados.

El diagrama de la Figura 3.12 se detalla la secuencia que sigue el sistema para establecer una nueva estrategia con la que se van a organizar los recorridos sugeridos para las entregas de las órdenes que van a ser distribuidas por el *repartidor*, con lo que este diagrama refiere al caso de uso "*Establecer estrategia de entregas*".

Un *repartidor* o el dueño de un *negocio*, puede estar interesado en saber los destinos de las órdenes, es decir, el *recorrido* de la *entrega. Este recorrido*, está formado por las posiciones destino ordenadas mediante la estrategia de reparto que el *negocio* eligió, con el fin de establecer un *recorrido* sugerido para repartir cada *orden*. La Figura 3.13, muestra la forma en que esto se lleva a cabo, se puede apreciar el caso de uso denominado "*Obtener recorrido de entrega*". Se puede apreciar que es desde el *SistemaMovil* se genera el pedido del *recorrido*, vale aclarar que la secuencia sería exactamente igual para el actor *SistemaExterno* y es por esto que solo se presenta un solo diagrama. En este caso, también se puede apreciar que el estado de la *orden* es *enviado*.

En la Figura 3.14 se puede apreciar un diagrama explicando la interacción entre las instancias para el caso de uso "*Registrar rastro de repartidor*". Este diagrama muestra cómo se guarda el *rastro* pendiente de un *repartidor* en el sistema, siempre y cuando el *repartidor* esté activo.

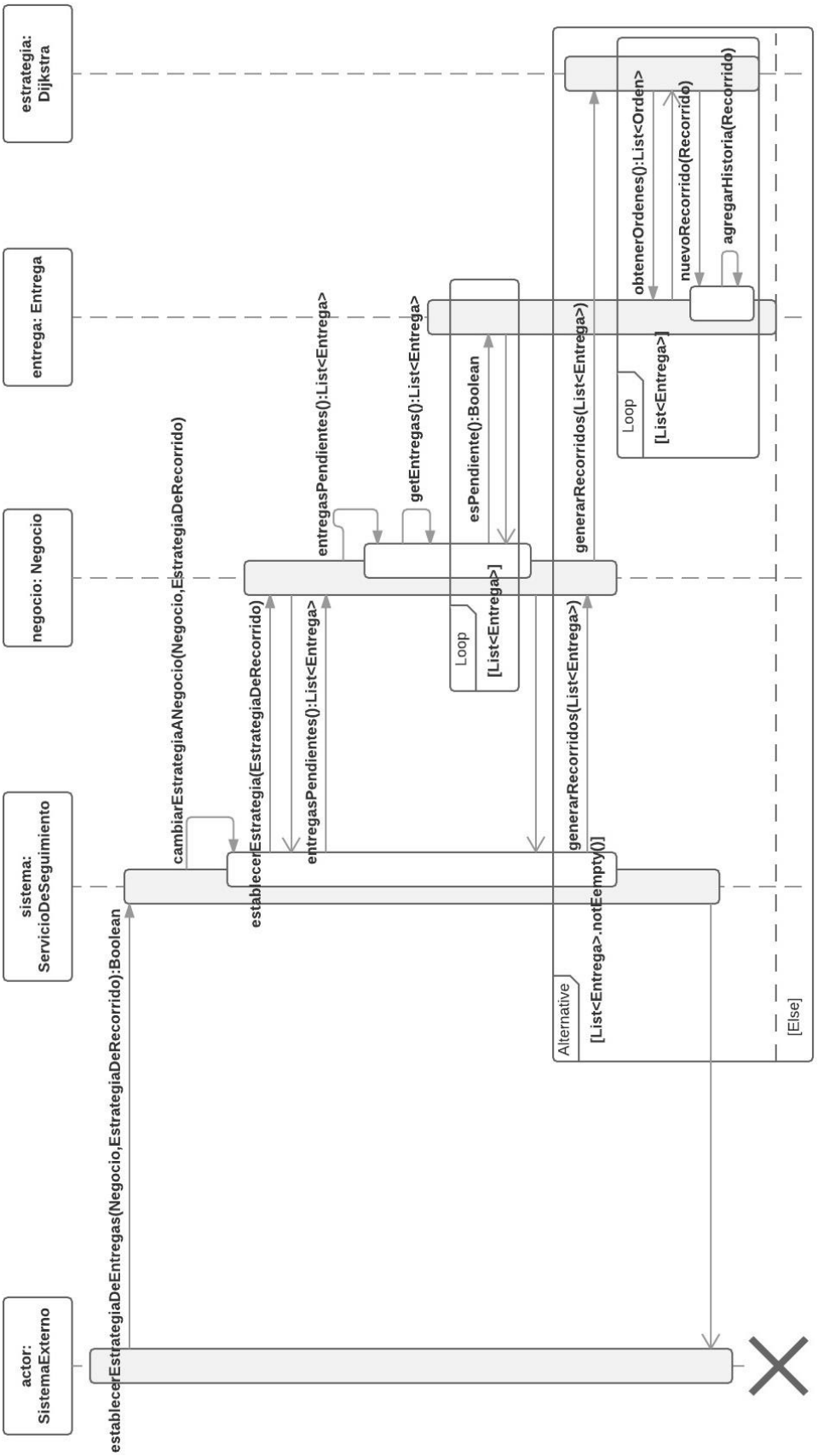

Figura 3.12: Establecer estrategia de entregas.

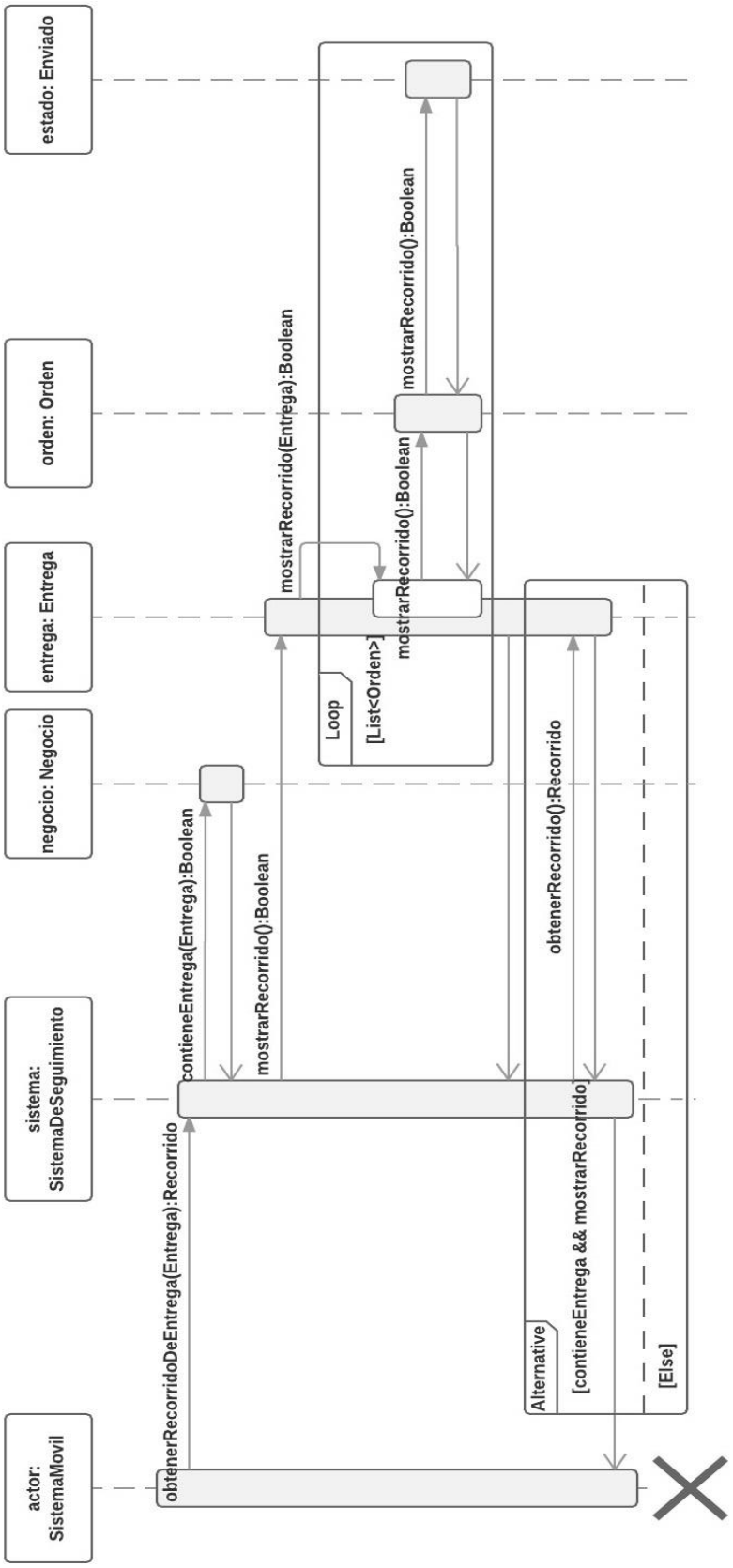

Figura 3.13: Obtener recorrido de entrega.

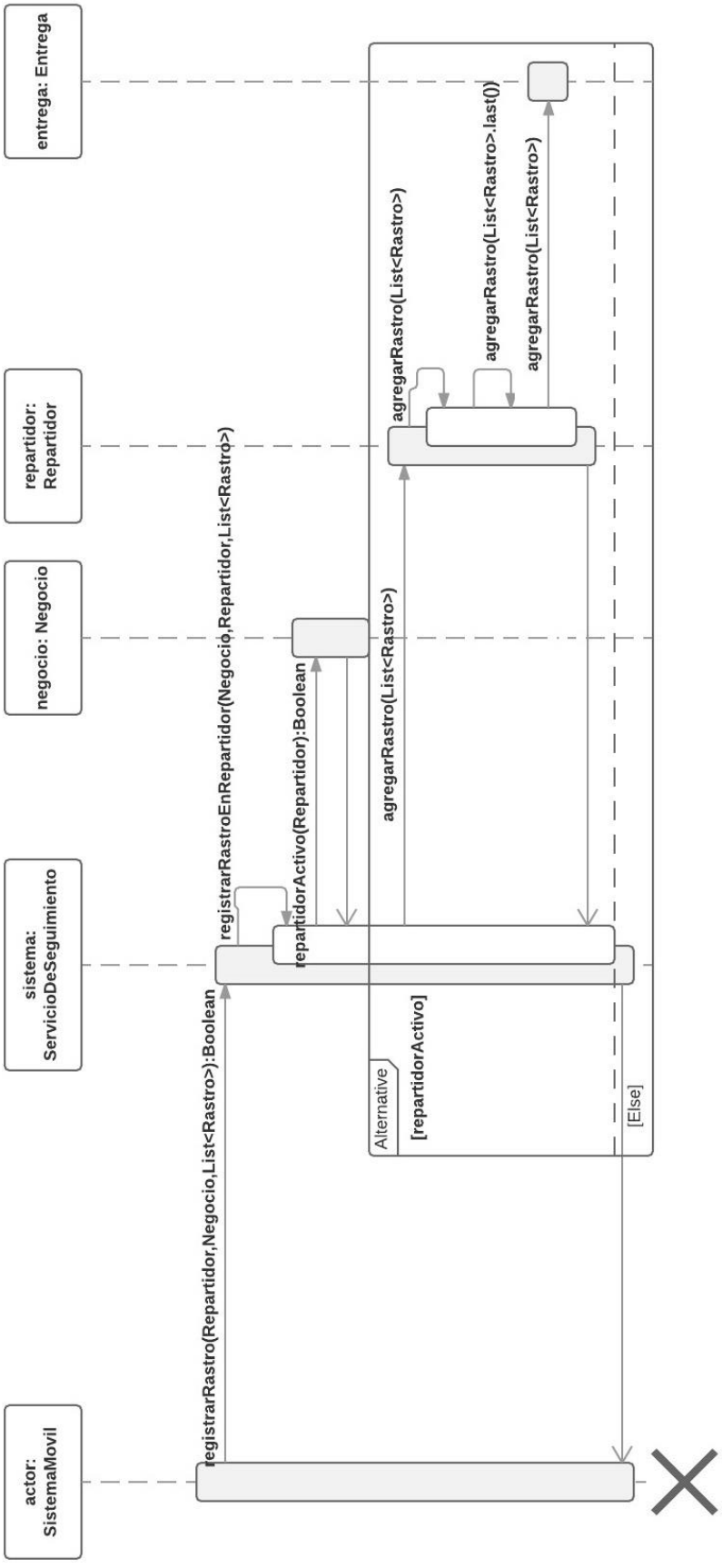

Figura 3.14: Registrar rastro de repartidor.

Un *negocio* puede obtener el *rastro* de un *repartidor* para una *entrega*, teniendo en cuenta esto, se detalla en el diagrama de secuencia del caso de uso "*Obtener rastro de entrega*" en la Figura 3.15. El fin es que se pueda ver en tiempo real el *rastro* que va dejando el *repartidor* para una *entrega*.

Del mismo modo que en el caso anterior, tanto un cliente como un *negocio* puede desear obtener el *rastro* de un *repartidor* para una *orden* en estado enviado, esto se detalla en el diagrama de secuencia del caso de uso "*Obtener rastro de orden*" (ver Figura 3.16). A este caso de uso se lo puede ver como un subconjunto del caso de uso "*Obtener rastro de entrega*", en el que se muestra el mismo *rastro* que la *entrega* hasta que la *orden* deje el estado enviado.

Un *negocio* asocia las órdenes a un *repartidor* activo. Dicha secuencia se detalla en el diagrama de secuencia del caso de uso *"Asociar orden a repartidor*" como se puede visualizar en la Figura 3.17.

Dependiendo del estado de una *orden* y del actor involucrado, el detalle que ésta muestre va a ser distinto, en base a esto, en la Figura 3.18 se muestra el diagrama de secuencia para el caso de uso "*Obtener detalle de orden*". Para este diagrama también cabe mencionar que al igual que en el diagrama "*Obtener recorrido de entrega*", la secuencia sería exactamente igual para el actor *SistemaExterno* y es por esto que solo se muestra uno de los casos (el del *SistemaMovil*).

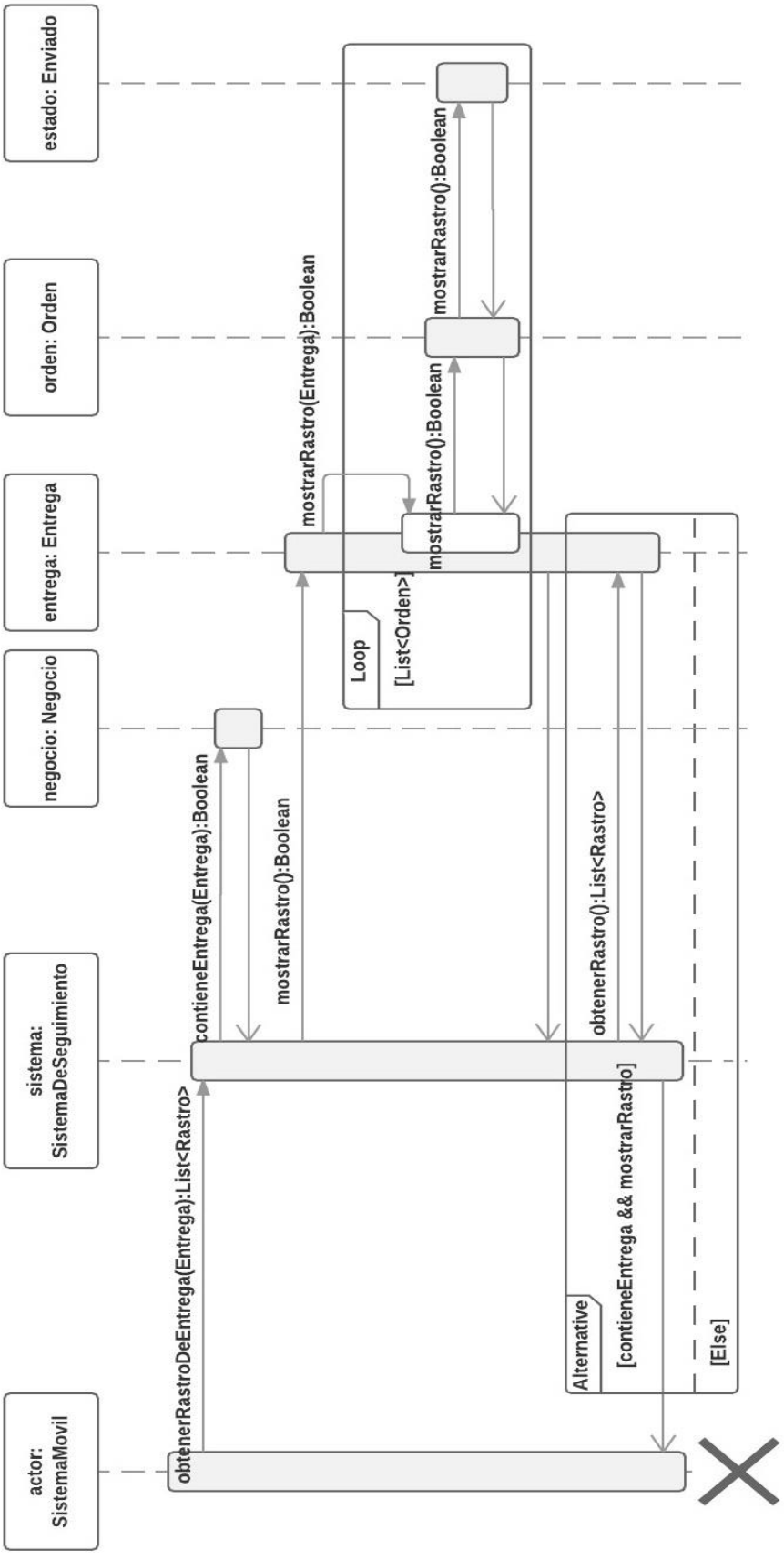

Figura 3.15: Obtener rastro de entrega.

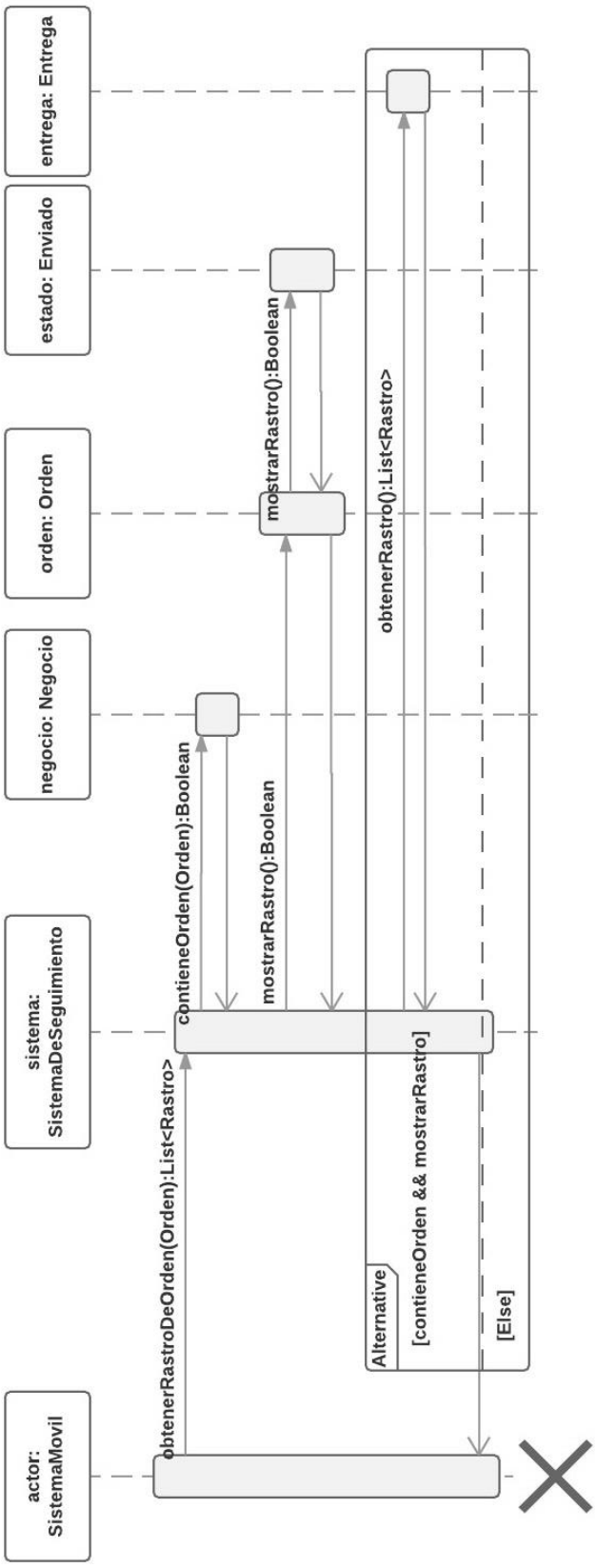

Figura 3.16: Obtener rastro de orden.

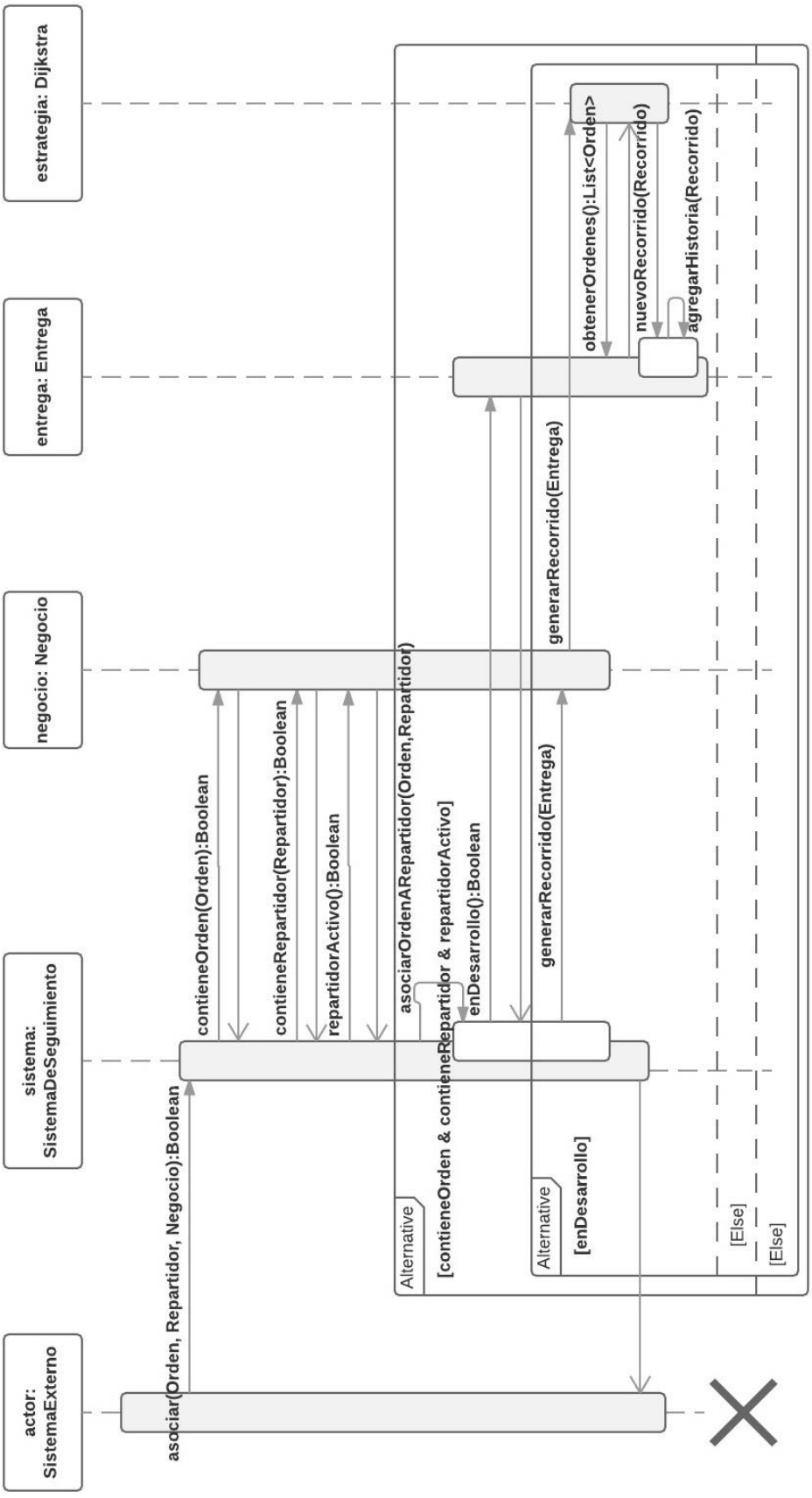

Figura 3.17: Asociar orden a repartidor

.

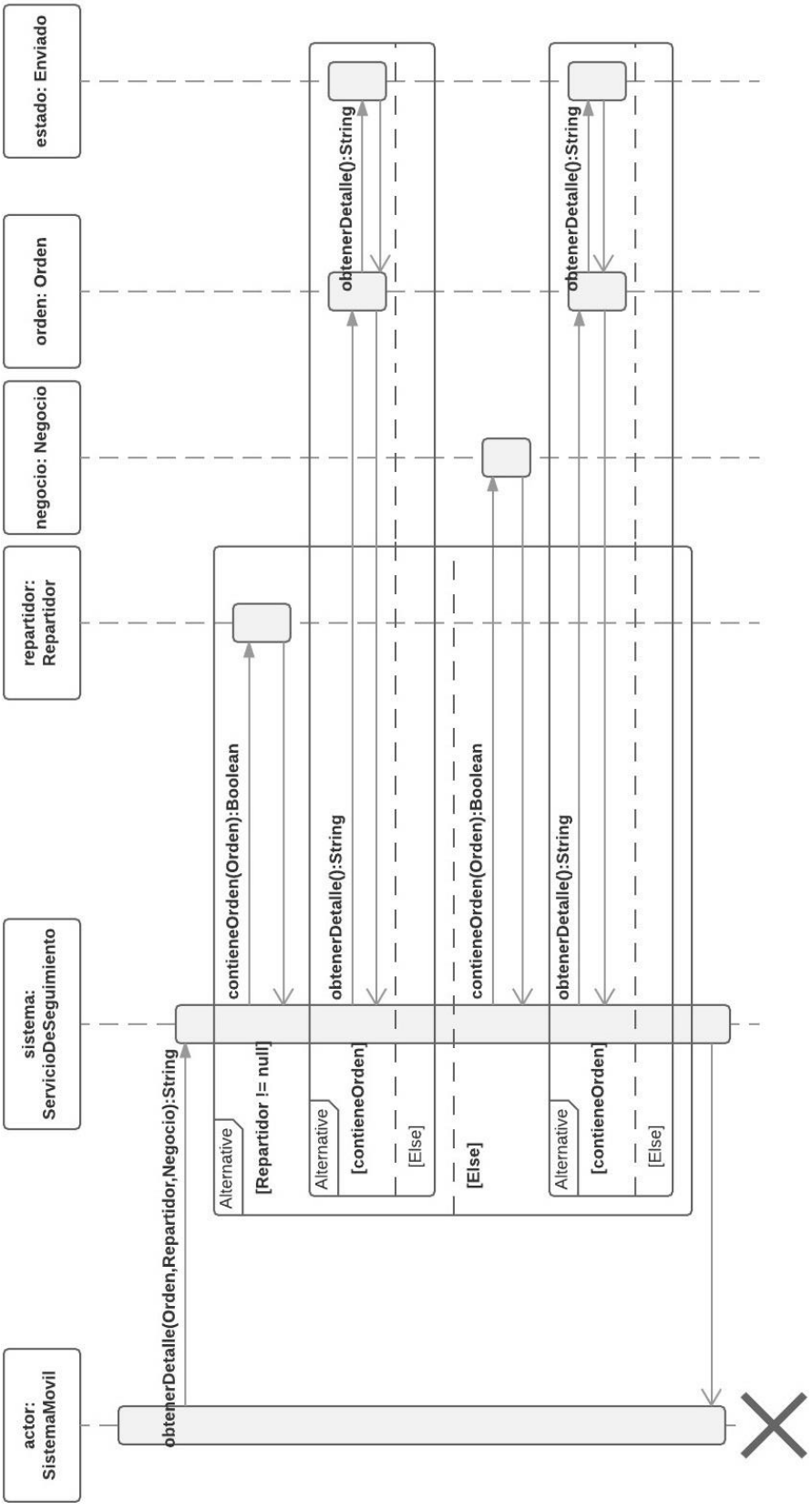

Figura 3.18: Obtener detalle de orden.

4. Prototipo Implementado

En este capítulo se presentará el prototipo desarrollado, el cual usa de base el modelo propuesto en el Capítulo 3. Este modelo es instanciado para un *negocio* de repartos de órdenes o pedidos[21] de comida, con el fin apreciar cómo se comporta el seguimiento de pedidos en tiempo real.

A continuación, se detalla la arquitectura empleada para la integración de cada una de partes que constituyen el sistema. Adicionalmente, se describen los detalles más destacados de la implementación como así también las funcionalidades de cada una estas partes, entre las que se encuentra la plataforma web que permite la gestión del dominio del sistema, y la aplicación móvil que es la encargada de enviar la *posición* del *repartidor*.

4.1. Arquitectura general del Prototipo

El prototipo se plantea con una arquitectura distribuida *Cliente-Servidor* [Sommerville, 2005] como se puede apreciar en la Figura 4.1. El intercambio de mensajes se da entre las aplicaciones clientes, que en nuestro caso, son aplicaciones móviles (de los *repartidores*) y navegadores web (del *negocio* y sus *clientes*) que solicitan servicios, y el proveedor de estos servicios es la plataforma web. Los servicios provistos son accedidos por las aplicaciones a través de llamadas a procedimientos remotos usando un protocolo de petición-respuesta HTTP.

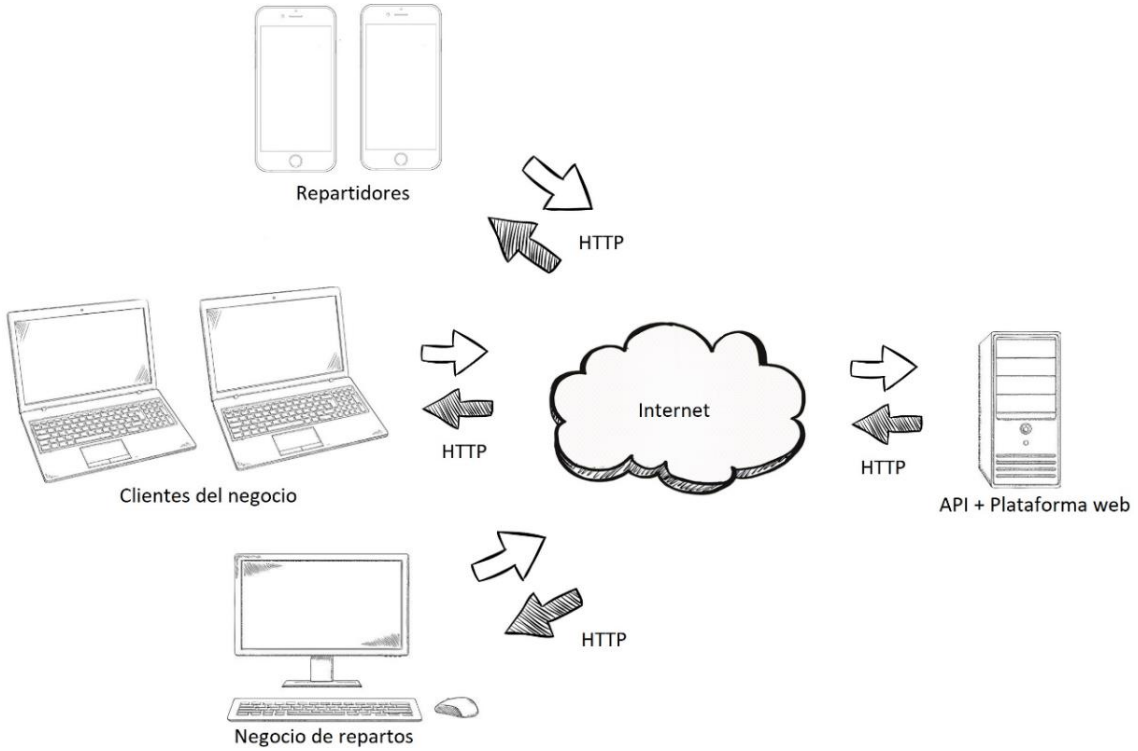

Figura 4.1: Arquitectura Cliente-Servidor del prototipo desarrollado.

[21] La palabra *orden,* pedido u *orden* de pedido, hacen referencia a la unidad de seguimiento de cada usuario, por lo que son sinónimos. El modelo contempla este concepto con el nombre de *orden*, pero en el prototipo también se utiliza la palabra pedido ya que es más representativa para este dominio de instanciación.

La elección de la arquitectura *Cliente-Servidor*, fue realizada en base a las características del dominio, donde se requiere comunicación y compartir información de forma continua entre las aplicaciones móviles y los navegadores Web. Este tipo de arquitectura, permite en un futuro, escalar fácilmente y de forma transparente el número de usuarios que interactúan en el sistema.

Cabe mencionar que las aplicaciones móviles envían al servidor información con las posiciones geográficas del *repartidor*, consulta la *entrega* asociada al *repartidor*, y disponen la lógica necesaria para generar cambios de estados en las órdenes realizadas por los clientes del *negocio* (las cuales residen en el *Servidor*). Por otro lado, los navegadores Web acceden a la aplicación Web que se ejecuta en el *Servidor* para ver la información enviada por los repartidores.

Para que las aplicaciones móviles y la aplicación Web puedan compartir información entre ellas, se desarrolló una API REST[22] [Fielding, 2000]. Es importante destacar que toda API está compuesta por servicios, recurso y clientes; los servicios son los puntos de acceso a la API y serán los encargados de, ante una petición de los clientes que en la implementación de este prototipo son las aplicaciones móviles, responder con los recursos solicitados. Para este prototipo, la API comparte y tiene acceso al mismo modelo de datos y lógica de negocios que la aplicación Web.

El servidor se aloca localmente en una computadora personal, pero se expone a la red con una aplicación de *tunneling*[23] llamada *Ngrok*[24], esto permite agilizar las pruebas de la comunicación entre las aplicaciones móviles y los servicios provistos por el servidor.

A continuación, se brindan más detalles de la aplicación Web, la API y la aplicación móvil desarrolladas para este prototipo.

4.2. Aplicación Web

La aplicación Web contiene la mayor parte de la lógica de *negocio* del prototipo, la cual se ira describiendo en esta sección. Esta lógica de negocios se basa en el modelo propuesto en Capítulo 3.

Dado que el modelo presentado en el Capítulo 3 estaba planteado para pedidos en general no se había hecho hincapié en el detalle o especificación del pedido. Acorde a esto, para el prototipo se decidió extender el modelo incorporando el concepto de *Producto*. Donde cada *negocio* y cada *orden* disponen de una lista de productos. Esto permite que, al realizar una *orden*, se registren los productos asociados a la misma. Además, estos productos se pueden modificar y se registra el valor del mismo en el momento del pedido (compra).

[22] *Representational State Transfer*: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Ultimo acceso 19/07/2017).
[23] *Tunneling*: http://www.cisco.com/c/en/us/products/ios-nx-os-software/tunneling/index.html (Último acceso 20/07/207).
[24] *Ngrok*: https://ngrok.com/ (Último acceso 20/07/2017).

Otra característica que se identificó al implementar el prototipo, es la necesidad de contar con dos estados adicionales relacionados a la *orden*. Estos estados surgen de la necesidad de tener mayor precisión en el prototipo a la hora de informar sobre el *estado* actual de una *orden*, los nuevos estados con los que se extiende el modelo son:

- "*Creado*", indica que la *orden* se creó, pero aún no comenzó a prepararse (es decir, no se empezó a preparar por el personal del *negocio*).
- "*ListoParaEnviar*", permite indicar que el pedido se encuentra listo para ser enviado en un reparto.

La incorporación de estos estados al modelo es posible ya que el mismo está diseñado para extender la jerarquía de estados.

A fines del prototipo, la *entrega* agrega una relación al *negocio.* Esta es utilizada para disponer de una relación de pertenencia con el mismo, facilitando de esta manera algunas características puramente de implementación y agilizando cálculos. Esto no es considerado una extensión al modelo como los casos anteriores, sino una incorporación en el prototipo a fines de eficiencia en realizar determinados cálculos.

La implementación de la aplicación Web se llevó a cabo con el framework Web *Ruby on Rails* [RubyOnRails], el cual se detalló en la Sección 2.2.1. Esta aplicación Web se dividió en dos áreas funcionales, una destinada a la administración que representaría la funcionalidad que requiere el *negocio* de repartos (backend) y otra enfocada a las necesidades de los clientes (frontend).

Desde el área de administración (backend) se pueden gestionar negocios, repartidores, entregas, ordenes, estrategias de gestión de caminos para una *entrega* y productos ofrecidos por cada *negocio*. También es posible visualizar en un mapa el *recorrido* de cada *entrega*, junto a la información de sus órdenes.

En esta misma área (backend) también es posible crear órdenes, provenientes, por ejemplo, de un pedido telefónico realizado por un cliente, permitiendo mantener una gestión de las mismas por parte del *negocio* de repartos. Entre las opciones de gestión, se encuentra la funcionalidad de cambiar los estados de cada *orden*, estos estados son determinantes para todo el proceso de seguimiento, ya que establecen en que momento está disponible una *orden* para formar parte de una *entrega*, el inicio de su seguimiento, la visibilidad de la información por parte del cliente, así como también el fin del seguimiento de la misma.

En la Figura 4.2 se pude apreciar el listado de órdenes de un *negocio* de repartos. Cada pedido (*orden*) se indica el nombre del cliente, la dirección de destino del pedido, el *estado* actual de la *orden*, y la lista de acciones que se pueden realizar. Estas acciones como se pueden observar en la figura pueden variar acorde al *estado* actual de la *orden*.

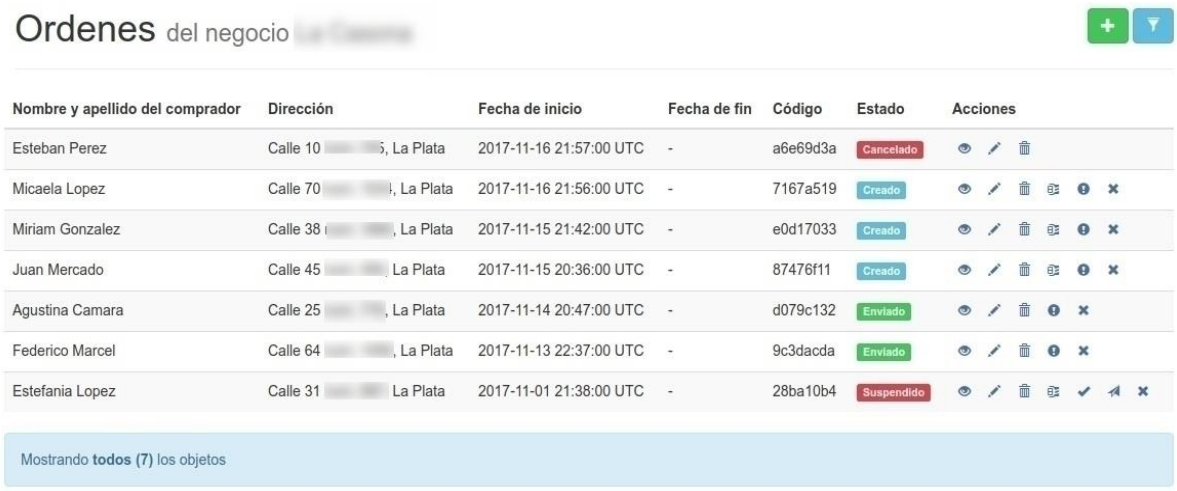

Figura 4.2: Listado de órdenes de un *negocio* de repartos con sus respectivos clientes, direcciones de destino, códigos únicos y estados.

Se pueden apreciar en la Figura 4.3, los iconos que representan las diferentes acciones disponibles por el *negocio* de repartos para aplicar sobre las ordenes (en la Figura 4.2 se mostró como estas acciones son visualizadas por el negocio). Se puede observar los iconos para las acciones "*Ver detalles*", "*Modificar*" o "*Borrar*", estas permiten ver detalles, modificar el pedido y borrar el mismo respectivamente. La opción "*Procesar*" permite indicar que la *orden* se encuentra en estado "*Desarrollo*", con respecto a la acción "*Marcar como enviado*", permite volver a marcar como "*Enviada*" a aquellas ordenes que se hallan suspendido durante su reparto. Por otro lado, "*Lista para enviar*" la da *orden* el *estado* "*ListoParaEnviar*" indicando que la misma se encuentra disponible para ser añadida a una *entrega.* Mientras que "*Cancelar*" y "*Suspender*" llevan la *orden* a los respectivos estados "*Cancelado*" y "*Suspendido*".

Figura 4.3: Opciones de cambio de *estado* para una *orden*.

Al dar de alta o editar conceptos que tienen asociado una dirección, como lo son una *orden* o un *negocio* de repartos, se utiliza la API *Google Maps Geocoding* [GoogleMaps] la cual, como se mencionó en la Sección 2.2.3, provee mediante el proceso de geocodificación[25] la funcionalidad de convertir direcciones en formato de texto en coordenadas geográficas, permitiéndonos guardar el par latitud-longitud asociado a la misma para luego ser usada en la lógica de posicionamiento. Si bien este cálculo se podría hacer cuando se necesite el par

[25] La geocodificación es el proceso de transformar una descripción de una ubicación (por ejemplo, un par de coordenadas, una dirección o un nombre de un lugar) en una ubicación conformadas por el par longitud-latitud.

latitud-longitud y no guardarla, se decide almacenarlo ya que dicho proceso es costoso y haría más lento el cálculo de caminos al intentar realizar este cálculo para todas las órdenes involucradas en la *entrega*.

En el área Web enfocada al cliente, es posible simular la generación de una *orden* de compra, eligiendo el *negocio*, distintos productos ofrecidos por este y proporcionando una dirección donde quiera recibir el pedido y datos referentes a quien realiza la compra. Esta funcionalidad crea una *orden* que se inicializa en *estado* "*Creado*". Una vez registrada en el sistema, se le entregará un código único que sirve para que el cliente busque su *orden* y dependiendo del *estado* de la misma le permita ver distinta información asociada de ella. Como, por ejemplo, cambios de *estado,* así como también seguir su *recorrido* en un mapa cuando la misma este en *estado* de "*Enviado*".

Una pantalla de ejemplo de la creación de una *orden* de pedido se puede apreciar en la Figura 4.4. Se pueden observar los productos incorporados en la orden.

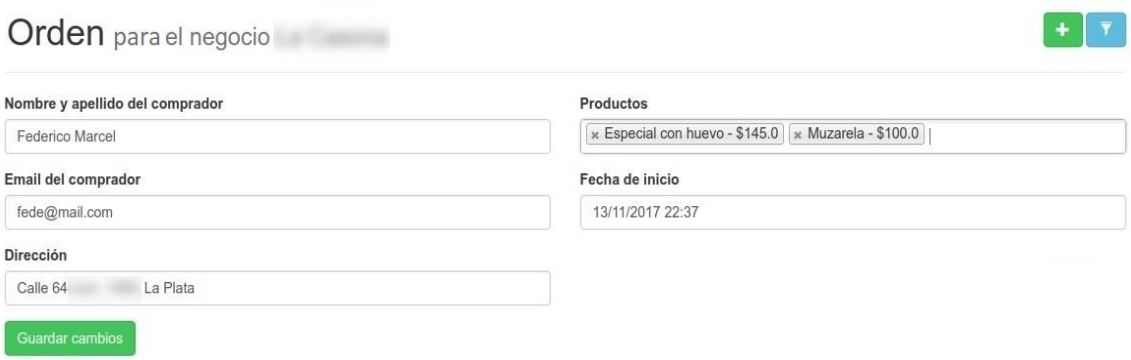

Figura 4.4: Creación de una *orden*.

La aplicación Web recibe la *posición* del *repartidor* cada vez que se detecte un cambio en éste desde la aplicación móvil que tiene instalada en su celular. La *posición* junto a la fecha y hora en que se registró, es utilizada para crear un *rastro* que se asocia a la representación del *repartidor* en la aplicación Web, y en caso de que este tenga asignado un reparto activo[26], también se creará otro rastro para éste (el cual se visualizará en la aplicación móvil). De esta manera, se muestra el *recorrido* continuo que va haciendo el *repartidor* junto con el sugerido por el sistema para la *entrega* que éste está realizando.

Un *negocio* puede agrupar órdenes que estén en *estado* "*ListoParaEnviar*" conformando una *entrega* a la cual se le asocia un *repartidor*. Una vez creada la *entrega* y cuando el *repartidor* se encuentre listo para realizarla, esta se podrá poner en curso desde la administración del *negocio* de repartos, lo que generará que todas las ordenes asociadas a la *entrega* pasen a *estado* "*Enviado*" y se establezca ese instante de tiempo como la fecha de comienzo del reparto.

El cliente puede seguir la evolución de su *orden* de pedido, al ingresar el código único que se entregó al generar el pedido, donde solo puede ver información de su *orden* y en caso de

[26] Como se mencionó en el Capítulo 3, una *entrega* activa, es aquella con al menos una *orden* en *estado* de e*nviado* o *suspendido*, un *repartidor* puede tener solo una entrega activa en un mismo momento.

que el *estado* lo permita el *recorrido* de la misma, pero no puede ver información ni el *recorrido* de las demás ordenes de la *entrega*.

En la Figura 4.5 se puede apreciar la información que puede observar el cliente en relación a su orden de pedido. En este caso, como el *estado* de la *orden* es "*Creado*" el usuario no puede visualizar el *recorrido* del *repartidor* aún.

Figura 4.5: Seguimiento del *estado* de una *orden* desde la perspectiva de un cliente. El mismo no puede ver el *recorrido* de su orden debido a que este está en *estado* "*Creado*".

Una vez que las ordenes asociadas a la *entrega* pasen a *estado* "*Enviado*" y se establezca ese instante de tiempo como la fecha de comienzo del reparto. En este momento es cuando los clientes comenzaran a ver el *recorrido* que realiza su pedido en un mapa de *Google Maps* [GoogleMaps] (más detalles de esta funcionalidad se brindó en la Sección 2.2.3), este *recorrido* será visible hasta que la *orden* sea entregada y pase al *estado* de "*Finalizada*", o hasta que la misma sea cancelada y pase al *estado* "*Cancelado*", momento en el cual ya no se podrán ver información de posicionamiento[27]. A su vez, desde el negocio de repartos se puede seguir el *recorrido* completo de la *entrega* hasta que la misma finalice.

En la Figura 4.6 se puede apreciar como el cliente puede visualizar el mapa asociado a su orden de pedido cuando el mismo tiene *estado* "*Enviado*". De esta manera, el usuario puede visualizar en tiempo real donde está el *repartidor* que hará entrega de su pedido, y el camino que el mismo viene realizando.

Por ahora el prototipo no informa cuantos pedidos faltan entregar antes de que le llegue el turno al usuario, pero esta información podría ser útil en un futuro para tener una estimación más real del tiempo que puede demorar el pedido en ser entregado.

[27] A modo expresar que una *orden* se encuentra en uno de estos estados (*finalizado* o cancelado), se usara el sinónimo de "*estado final*".

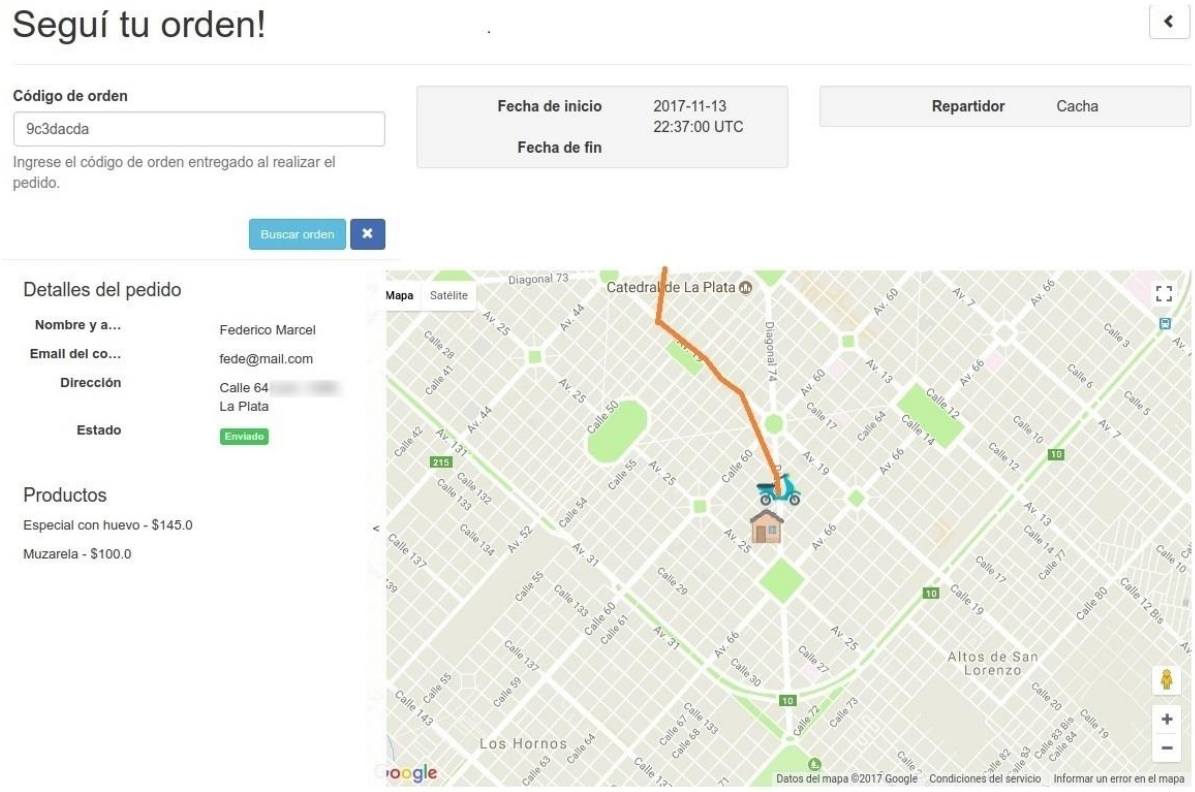

Figura 4.6: Seguimiento del *estado* de una *orden* desde la perspectiva de un cliente. *Recorrido* realizado por un *repartidor* para una *orden* en estado "*Enviado*".

El *negocio* de repartos, para el prototipo implementado, tiene la capacidad de elegir entre dos estrategias de búsquedas de caminos, la estrategia elegida va a ser utilizada cuando se genera el *recorrido* recomendado que vera el *repartidor* en su aplicación móvil al momento de distribuir cada *entrega*, este *recorrido* esta armado a partir de los destinos de cada una de las ordenes que la componen. Este cálculo se realiza del lado del *Servidor*, con el fin de aliviar de carga de procesamiento y de cálculo en el dispositivo móvil del *repartidor*.

Para ambas estrategias, el algoritmo utilizado tiene en cuenta la dirección del *negocio* de repartos como punto de partida y destino (para que también se calcule la vuelta al negocio de repartos) y como puntos intermedios, las direcciones destino de las ordenes que se encuentren en *estado* "*Enviado*".

Las estrategias implementadas en el prototipo son las siguientes:

- Algoritmo de grafos *TSP* (*Travelers Salesman Problem*) el cual se desarrolla a partir de una variante de Dijkstra propuesta en [Kuo-pao Yang et al., 2015]. Este algoritmo es conocido como el problema del viajante [Applegate et al., 2011] y responde a la siguiente pregunta general: dada una lista de ciudades y las distancias entre cada par de ellas, *¿Cuál es la ruta más corta posible que visite cada ciudad exactamente una vez y finaliza al regresa a la ciudad origen*?. Esto fue implementado, en particular, considerando el camino más corto para entregar todos los pedidos de la entrega.

  En particular, esta estrategia utiliza las distancias entre las direcciones mencionadas anteriormente (como origen y destino la dirección del *negocio*, y las direcciones de las *ordenes* como puntos intermedios). Estas distancias son obtenidas a través de

sus coordenadas longitud-latitud utilizando la API *Google Maps Distance Matrix*[28], la cual permite obtener la distancia entre un punto orígenes y destino a partir de su longitud y latitud. Estas distancias, sirven de ponderación que ayudan al algoritmo para determinar el camino recomendado.

- Algoritmo *Random*, este surge con el fin de poder proveer otra alternativa de *estrategia* y así poder probar el funcionamiento del cambio de estrategias para calcular caminos. En este caso, el *orden* de los destinos se establece de forma aleatoria. Si bien es una estrategia ineficiente para este domino, en este prototipo por simplicidad se decidió implementarla a fines prácticos de las pruebas de funcionalidad.

Como resultado de cualquiera de las dos estrategias se obtiene una lista ordenada de las órdenes que serán enviadas a la aplicación móvil del *repartidor* cuando este consulte por la *entrega* activa. Cada *reparto* guarda una referencia a la *estrategia* con que fue calculado el *recorrido* de la misma.

Una vez que se cuenta con la lista de órdenes procesada por el algoritmo de la *estrategia* elegida, tanto la aplicación móvil (del *repartidor*) como el *negocio* de repartos, las utilizaran para mostrar el *recorrido* recomendado al *repartidor*. Adicionalmente, el *negocio* de repartos, puede ver en el mismo mapa el recorrido real realizado por el *repartidor* para dicha *entrega*. En la Figura 4.7 se observa la perspectiva que puede apreciar un *negocio* al inspeccionar el detalle de una *entrega*. En particular, se muestra en el mapa el *recorrido* sugerido a partir de una *estrategia* elegida, y además el camino real realizado por el *repartidor*.

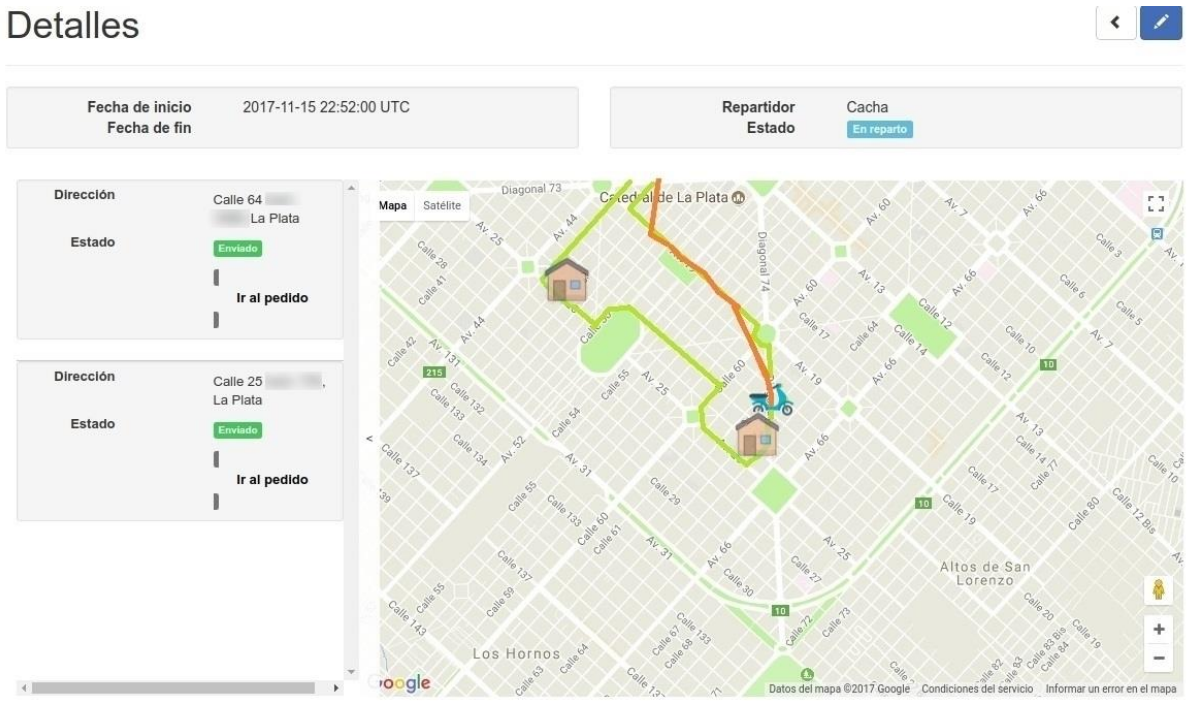

Figura 4.7: *Recorrido* sugerido y real realizado por un *repartidor* para una *entrega* de un *negocio,* vista desde el backend.

[28] Página de *Google Maps Distance Matrix API*: https://developers.google.com/maps/documentation/distance-matrix/intro?hl=es-419 (Ultimo acceso 21/11/2017)

Existe una limitante en cuanto a la cantidad de órdenes que se le puede asociar a un *reparto*, debido a que la versión gratuita de la *API de Google Maps* usada en este prototipo establece un límite de 23 posiciones como máximo por petición para armar los recorridos. Como los recorridos se armarán a partir de las posiciones de cada *orden* del reparto más la *posición* del *negocio* como inicio y fin (serían dos posiciones más), con lo cual como máximo se puede asociar a un reparto 21 órdenes de pedido.

4.3. API

Como se describió en la Sección 4.1, la API se implementó con el fin de dar soporte para la interconexión de la aplicación Web con las aplicaciones móviles, por lo que se implementaron solo los servicios requeridos para esta comunicación. Tanto la API como la aplicación Web tienen acceso al mismo modelo de datos.

En esta implementación particular, la API provee las funcionalidades de recibir peticiones por parte de la aplicación móvil para brindar información sobre la *entrega* activa asignada al *repartidor* (ordenada según la *estrategia* de recorridos elegida), el cambio de *estado* de las ordenes que conforman la *entrega*, o la actualización de información de posicionamiento del *repartidor*.

Se aprovecharon funcionalidades disponibles en la versión del framework *Ruby on Rails* [RubyOnRails] (utilizada para la creación de la aplicación Web) para el desarrollo de API.

Para estructurar las respuestas de los servicios brindados por la API se utiliza el estándar JSON API[29]. Este estándar busca principalmente la eficiencia en el uso de los recursos. Para esto, define como se deben estructurar las peticiones los clientes, así como también, como los servicios deben responder a estas. Se eligió este estándar ya que nos parece muy acertado como plantea la estructuración de las respuestas, dada la creciente adopción del mismo por gran parte de empresas y desarrollos, y por la facilidad de integración con los lenguajes y frameworks utilizados.

Los puntos de accesos (*endpoints*) a la API son los siguientes:

- GET */api/repartidores/:id_repartidor/entrega_activa*
  Devuelve la lista de ordenes asociada al *repartidor* (representado por ":id_repartidor") ordenadas según la estrategia elegida por el negocio.

- POST *api/ordenes/:id_orden/marcar_como_cancelado*
  Permite pasar la *orden* (representada por ":id_orden") a *estado* "*Suspendido*".

- POST *api/ordenes/:id_orden/marcar_como_finalizado*
  Permite pasar la *orden* (representada por ":id_orden") a *estado* "*Finalizado*".

- POST *api/repartidores/:id_repartidor/rastros*
  Permite enviar la última *posición* del *repartidor* (representados por ":id_repartidor").

[29] JSON API: http://jsonapi.org/ (Ultimo acceso 15/11/2017).

Por simplicidad, no va se realizó la autenticación de las aplicaciones que consumen los servicios de la API, pero sabemos que es vital implementar algún tipo de autenticación para las aplicaciones que deseen consumir los servicios en un ambiente productivo.

4.4. Aplicación Móvil

La aplicación móvil nativa está desarrollada con la tecnología *Android* [Android] (la cual se describió en detalle en la Sección 2.2.2). Esta aplicación desarrollada para funcionar con la API[30] 15 en adelante y se debe instalar en el dispositivo móvil del *repartidor*.

Esta aplicación móvil envía al *Servidor* las posiciones geográficas cuando el *repartidor* se desplaza y permite realizar el cambio de *estado* de cada *orden* de la *entrega* que este repartiendo. Una funcionalidad que se ejecuta en segundo plano se encargar de enviar las posiciones del *repartidor*, mientras que el cambio de *estado* de las ordenes se realiza manualmente por el *repartidor*.

Si bien *Android* provee como parte de su framework la API Location para acceso a las posiciones geográficas, se decidió que las mismas sean obtenidas usando el servicio de *Google Play Services Location*[31] para contar con una mayor precisión. Este servicio facilita agregar información contextual a la aplicación, como puede ser el conocimiento de posicionamiento automatizado para el seguimiento (*Location awareness*), la cercanía geográfica (*Geofencing*) y el reconocimiento de actividad de usuario (*Activity recognition*), algo que se debería hacer manualmente si se optará usar la API Location del framework de *Android*. Además, este servicio provee una herramienta más poderosa y de más alto nivel que maneja automáticamente los proveedores de posicionamiento (GPS, red móvil, red WIFI, posiciones GPS de otras aplicaciones); incluso maneja las actualizaciones de posiciones en base a parámetros de consumo de batería.

En el momento en que un *repartidor* inicia el reparto que tiene asignado desde la aplicación móvil, provoca que su *posición* comience a ser enviada al *Servidor*, usando el punto de acceso "*rastros*" definido en la API del prototipo.

En la Figura 4.8 se puede apreciar la vista de la aplicación móvil de un repartido, donde se ve la posición actual del mismo, y además se indica que no tiene entregas activas hasta el momento.

[30] El nivel de API es un valor entero que identifica de manera única la revisión del Framework API que ofrece una versión de la plataforma de *Android*. Este Framework API es el que las aplicaciones usan para interactuar con el sistema de Android subyacente.
[31] Página de *Google Play Services Loation*: https://developer.android.com/training/location/index.html (Último acceso 08/10/2017).

Figura 4.8: Aplicación móvil sin *entrega* iniciada.

Un *repartidor* solo puede tener una *entrega* activa en un determinado momento, por lo que al iniciar el reparto la aplicación móvil lista las órdenes y el *recorrido* sugerido de dicha *entrega*. Para esto, a través del punto de acceso "*entrega_activa*" se obtiene una lista con los destinos ordenados en base a la *estrategia* elegida por el *negocio*. A partir de esta lista, se utiliza la *API Directions de Google* [GoogleMaps] para obtener el *recorrido* sugerido, dicha API tiene en cuenta además del orden de los destinos que se le envían, el sentido de las calles para calcular el *recorrido*. Esta API fue descripta en la Sección 2.2.3.

El recorrido resultante (JSON) que se obtiene de la *API Directions de Google* se muestra en un mapa de *Google Maps* [GoogleMaps] como se describió en la Sección 2.2.3. Dicho *recorrido* tiene el fin de ayudar al *repartidor* a realizar el reparto de las ordenes de la *entrega* que tiene asignada.

Adicionalmente, el *repartidor* puede ver el *recorrido* real que está realizando, el destino de las ordenes en el mapa, y la lista ordenada de los pedidos previamente consultados al *Servidor*. Un ejemplo de esto se puede apreciar en la Figura 4.9.

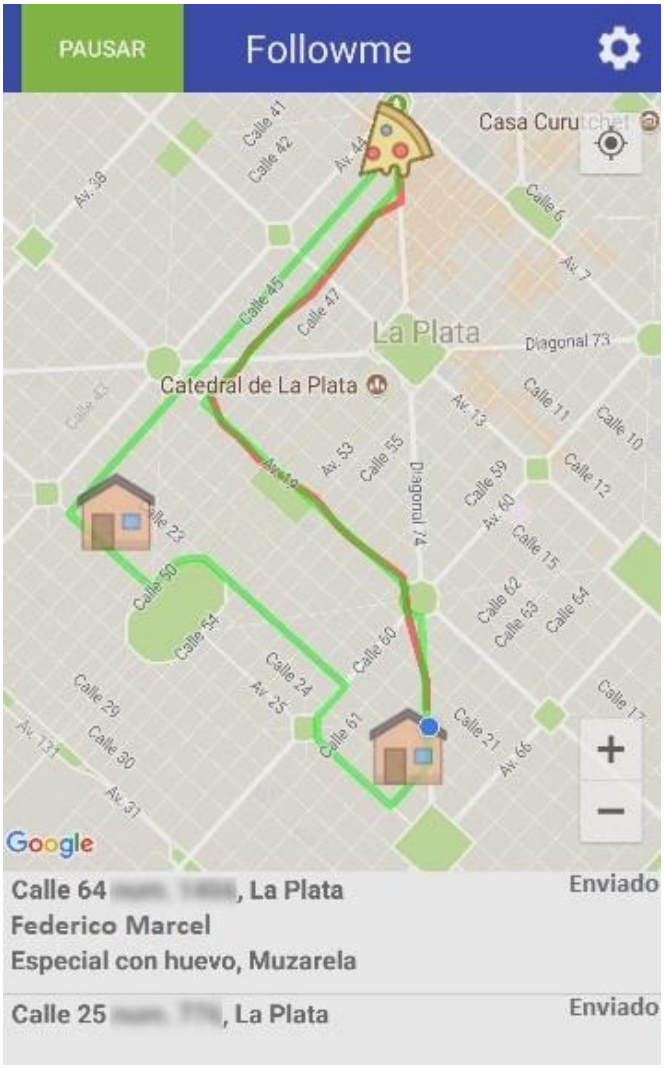

Figura 4.9: Aplicación móvil con *entrega* iniciada en el que se puede ver la lista de órdenes, los destinos de las ordenes, el *recorrido* sugerido y el *recorrido* continuo realizado por el *repartidor*.

Cuando un *repartidor* llega al destino de una de las ordenes, puede indicar a la aplicación a través de un dialogo el *estado* final de esta, es decir, el éxito o el fracaso de la *entrega* de esa orden de pedido en particular. Como se puede observar en la Figura 4.10.

Una vez establecido el *estado* final de una de las órdenes que formaron parte del *recorrido* de la *entrega*, el *repartidor* seguirá informando su *posición* al *negocio* de repartos hasta que éste pase a un *estado* final el resto de las ordenes que conforman la *entrega* en cuestión y que no fueron entregadas aún.

Se puede apreciar en la Figura 4.10 que el *repartidor* tiene la opción de cancelar la entrega como, por ejemplo, en el caso en que no había nadie en el domicilio indicado como dirección de entrega. La operación de cancelación o finalización de una *orden* se realiza a través de los respectivos puntos de acceso definidos en la API del prototipo "*marcar_como_cancelado*" y "*marcar_como_finalizado*" respectivamente.

Figura 4.10: Dialogo que permite indicar el *estado* final de una *orden* desde la aplicación móvil.

No se realizan *backup* locales de la información (salvo de la configuración personal de cada *repartidor*), ya que lo que se busca es tener siempre sincronizado el *Servidor* con las aplicaciones, de esta manera, si se cierra la aplicación y se solicita nuevamente la *entrega* activa, esta se encuentra en el mismo *estado* previo al cierre de la aplicación, de esta manera se evita tener que manejar las inconsistencias, contribuyendo así a una mayor simplicidad en la aplicación. Para mantener esta sincronización y teniendo en mente que el *estado* de las ordenes que aún faltan entregar puede ser cambiado desde el *negocio* de repartos (a través de la aplicación Web), cuando se inicia el *reparto* y luego de ello cada un determinado tiempo (1 minuto), se consulta al *Servidor* la *entrega* activa del *repartidor* para saber si alguna de las ordenes sufrió modificaciones, como podría ocurrir en el caso de que un cliente cancele una orden de pedido. Realizar esta última operación mencionada, llevaría la *orden* a *estado* "*Cancelado*", en tal caso, se recalculará el *recorrido* recomendado, se actualizará la lista de órdenes y los destinos en el mapa, y por último se notificará el *repartidor* de lo sucedido para que no vea cambios sorpresivos en la aplicación móvil.

La aplicación móvil también cuenta con la opción de realizar configuración de algunos parámetros como lo pueden ser la distancia o el lapso de tiempo mínimo para actualizar la

posición, o la *URL Ngrok* (mencionada en Sección 4.1) que la aplicación va a utilizar para interactuar con el Servidor. Estos datos se almacenan localmente en un archivo de configuración de la aplicación móvil. La pantalla de configuración se puede apreciar en la Figura 4.11.

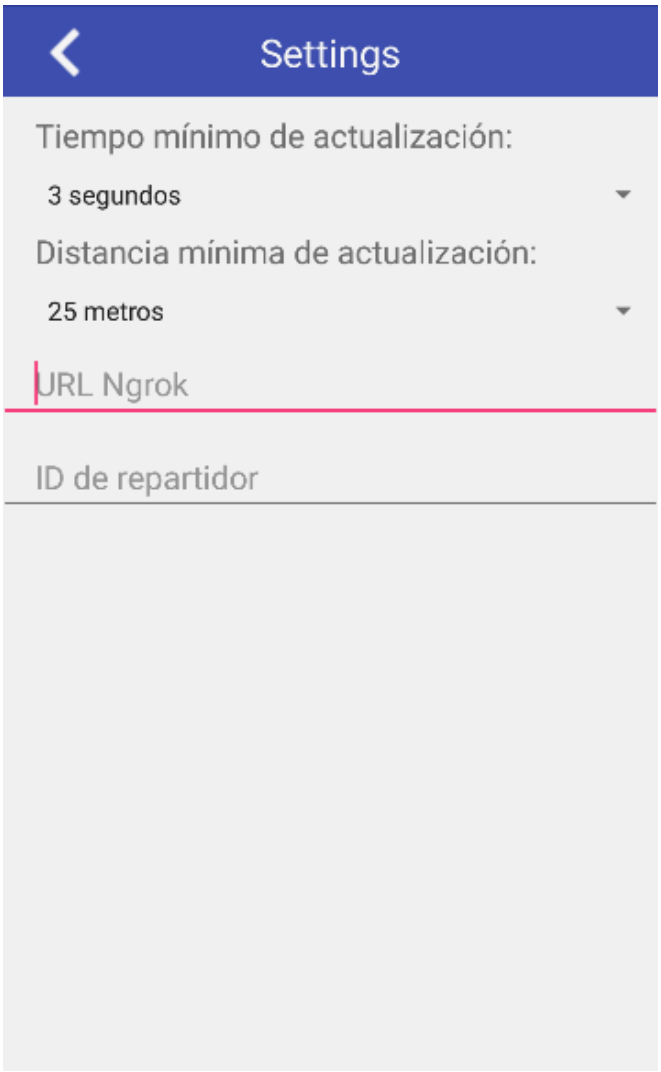

Figura 4.11: Parámetros de configuración de la aplicación móvil.

Cabe aclarar que como no hay autenticación de usuarios que estén dados de alta en la aplicación Web como repartidores, la aplicación móvil va a utilizar los servicios de "*entrega_activa*" y "*rastros"* con un "*id_repartidor*" que también puede ser establecido en la pantalla de configuración (como se puede apreciar en la Figura 4.11), es decir que utiliza uno de los identificadores de los repartidores que se sabe que existe en el *Servidor*.

5. Ejemplo de uso del Prototipo

En este capítulo se explica la utilización del prototipo implementado. Para comprender mejor el funcionamiento del mismo, primero se presentará el estado inicial del prototipo, es decir, aquella información que ya tiene cargada el mismo para poder funcionar, por ejemplo, negocios, ordenes de pedido, etc. Luego, se mostrará el proceso involucrado en la creación de una *entrega* de pedidos, para posteriormente mostrar como un *repartido*r realiza el reparto de sus órdenes de pedidos.

- *Estado inicial del Prototipo*

  Para este ejemplo de uso de prototipo, se partirá de un estado inicial donde la aplicación Web tiene precargados negocios con sus respectivos productos. En la Figura 5.1 se puede apreciar la lista de negocios[32] ya cargados. En particular, para el ejemplo mostrado en este capítulo, solo se hace foco en un solo *negocio*.

Figura 5.1: Listado de negocios vistos desde el *backend*.

Los negocios son visualizados por los clientes como se muestra en la Figura 5.2. Una vez que un cliente selecciona un *negocio* puede realizar pedidos para dicho *negocio*.

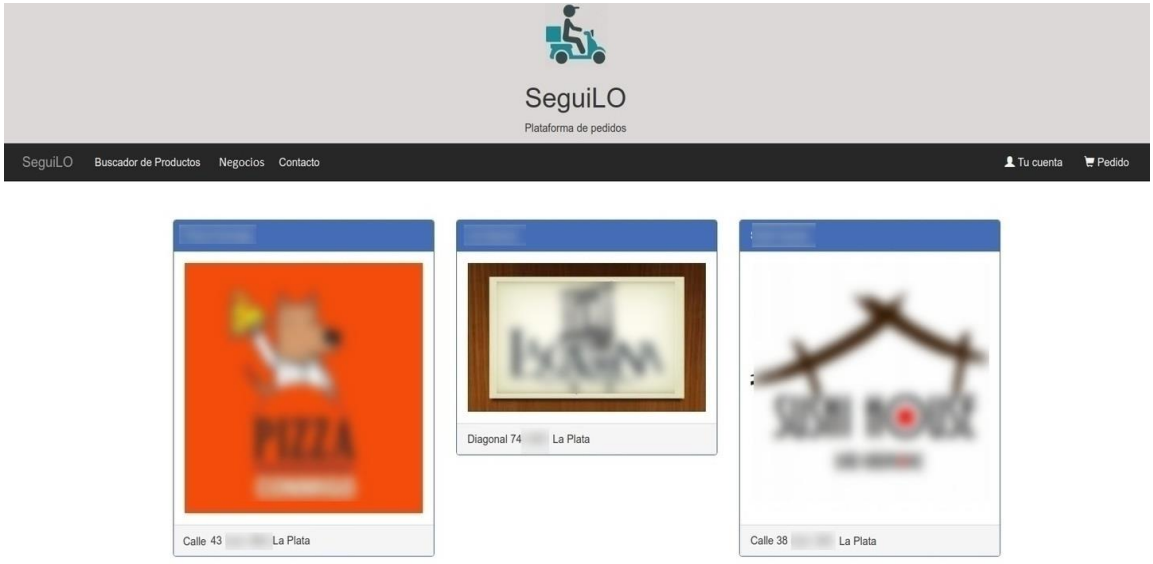

Figura 5.2: Listado de negocios vistos desde el *frontend*.

[32] Si bien se usaron de guía negocios existentes para darle más realidad al prototipo, para las imágenes se han difuminado tanto los nombres de los mismos como las direcciones.

Los negocios además tienen precargados productos para poder facilitar así la creación de los pedidos. En la Figura 5.3, se puede apreciar la lista de productos cargados para un *negocio*. Cabe mencionar, que este *negocio*, es el que se usará para mostrar los distintos ejemplos de funcionalidades del prototipo este capítulo.

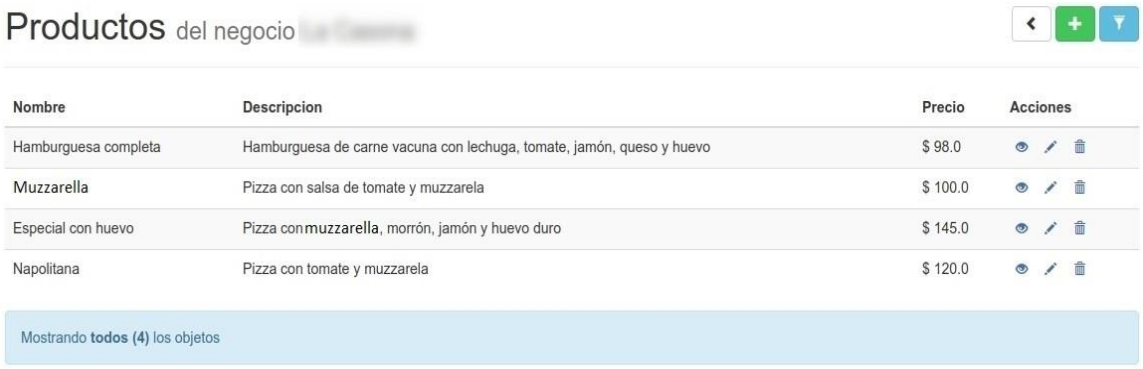

Figura 5.3: Listado de productos precargados del *negocio* utilizado para el ejemplo de uso del prototipo.

Inicialmente el *negocio,* que se va a utilizar para el ejemplo de uso del prototipo, tendrá seleccionada la *estrategia* de cálculos de recorridos "*Armado de Recorrido más Corto*", descripta en Sección 4.2 como *TSP* (*Travelers Salesman Problem*). En la Figura 5.4 se muestra la pantalla que visualiza el *negocio* para poder cambiar la *estrategia* para calcular recorridos. Se puede apreciar que el nombre es descriptivo para que el personal del *negocio* pueda comprender fácilmente la funcionalidad de esta *estrategia*.

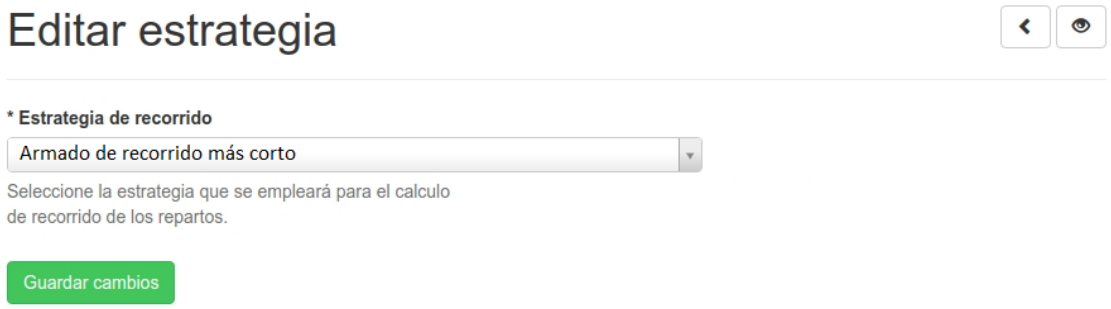

Figura 5.4: Selección de la *estrategia* de cálculos de *recorrido* a utilizar para la generación de los recorridos de los repartos.

El *negocio*, también contara con órdenes de pedido en distintos estadios y repartidores precargados. Recordemos que tanto el cliente como el *negocio* de repartos pueden crear órdenes[33], la cuales comienzan con un *estado* "*Creado*". Cuando dicha *orden* empieza a ser elaborado, algún empleado del *negocio* cambia el *estado* de la misma a "*Desarrollo*". Esto se puede apreciar en la Figura 5.5 donde hay tres órdenes de pedidos en el estado en "*Desarrollo*". En esta figura también se pude contemplar que los pedidos tienen distintos estados, y cada uno de ellos tiene asociadas diferentes acciones que se pueden realizar, como se describió en la Sección 4.2.

[33] La creación de órdenes de pedidos por parte de un cliente fue mostrada en la Figura 4.4 previamente en la Sección 4.2.

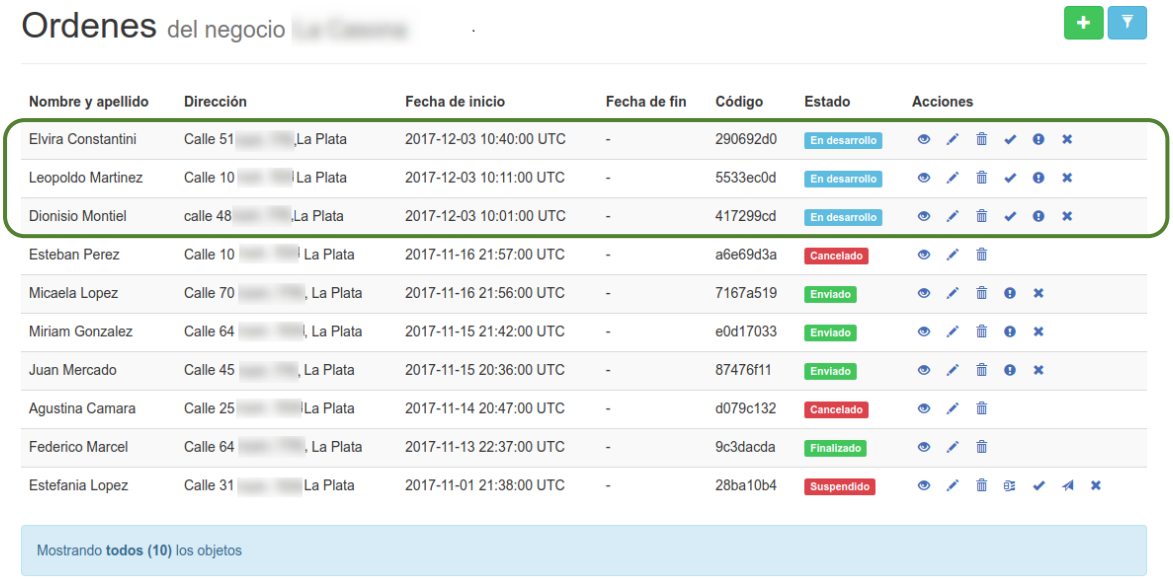

Figura 5.5: Listado de órdenes del *negocio* en un momento dado.

A medida que finalice la elaboración de cada uno de las ordenes, algún empleado del *negocio* cambiará el *estado* de la misma a "*ListoParaEnviar*". Esto permite que puedan ser seleccionadas para formar parte de una nueva *entrega*. En la Figura 5.6 se pueden ver que las tres órdenes de pedidos que se estaban elaborando pasaron a *estado* "*ListoParaEnviar*".

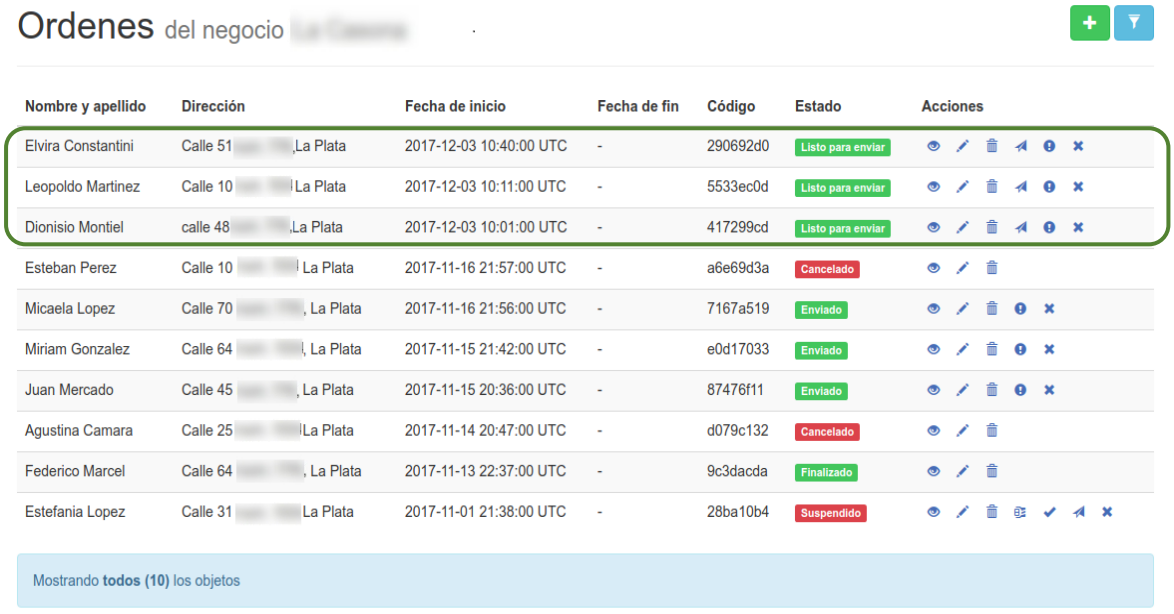

Figura 5.6: Listado de órdenes del *negocio*, con el cambio de *estado* a "*ListoParaEnviar*" de las ordenes que previamente se encontraban en *estado* "*Desarrollo*".

Cabe mencionar, como se destallo en la Sección 4.2, que los clientes, mientras sus órdenes de pedidos están en *estado* "*Creado*", "*Desarrollo*" y "*ListoParaEnviar*", solo visualizan esta información, pero recién cuando el *repartidor* inicia el reparto de la *entrega* es cuando empiezan a ver el *recorrido* de su *orden* de pedido en el mapa.

- *Creación de una entrega de órdenes de pedidos*

  La *entrega* se puede crear solamente con aquellas ordenes de pedido que se encuentran en *estado* "*ListoParaEnviar*". A modo de ejemplo, para la creación de la *entrega* se usarán las tres órdenes de pedidos de la Figura 5.6 que se encuentran en el *estado* "*ListoParaEnviar*". En la Figura 5.7 se puede apreciar cómo se realiza la creación de la *entrega*.

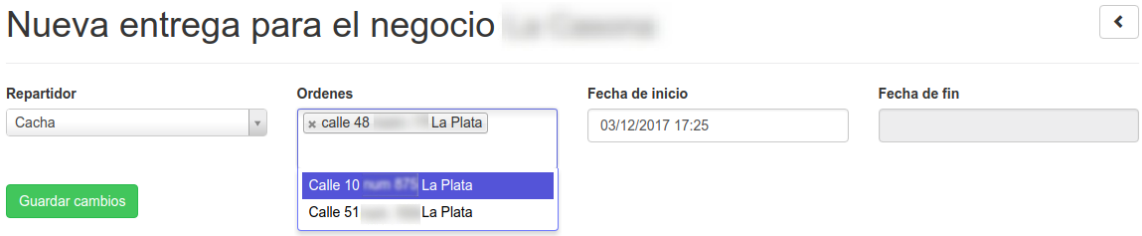

Figura 5.7: Creación de una entrega con las tres órdenes de pedido que estaban en *estado* "*ListoParaEnviar*".

Se puede observar en la Figura 5.7 que la *entrega* creada es asignada a un *repartidor* particular. Una vez creada, algún empleado del *negocio* la marca como enviada indicando que la misma está disponible para ser trasportada por el repartidor y provocando que todas sus órdenes pasen a *estado* "*Enviado*". En la Figura 5.8 se puede apreciar que una de las entregas ha sido enviada y se encuentra en proceso de reparto.

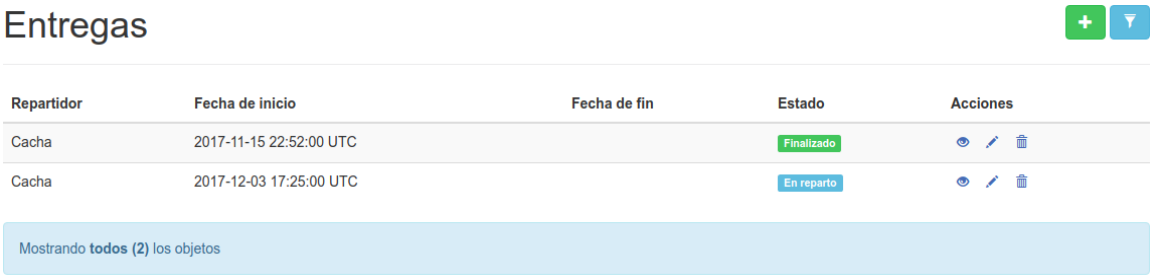

Figura 5.8: Listado de entregas, una ellas en proceso de reparto.

A partir de este momento, las ordenes que conforman la entrega, se comenzaran a visualizar para los respectivos clientes en los mapas de seguimiento, así como también se hace visible para el negocio, el recorrido sugerido de la entrega activa. Por otro lado, el repartidor podrá ver también el recorrido que debe realizar cuando este inicie el reparto de esta entrega en particular.

Supongamos, como se mencionó anteriormente, que esta seleccionada la *estrategia* de cálculo de *recorrido* "*Armado de Recorrido más Corto".* En la Figura 5.9 se puede apreciar la pantalla que recibe el *negocio* de repartos.

Si bien en la Figura 5.9 el *negocio* puede apreciar el *recorrido* recomendado para la *entrega,* dado que el *repartidor* todavía no dio inicio al reparto, este todavía no visualiza dicho *recorrido* en su dispositivo móvil; en el caso de los clientes, hasta este momento, solo ven en el mapa marcada la dirección de su casa.

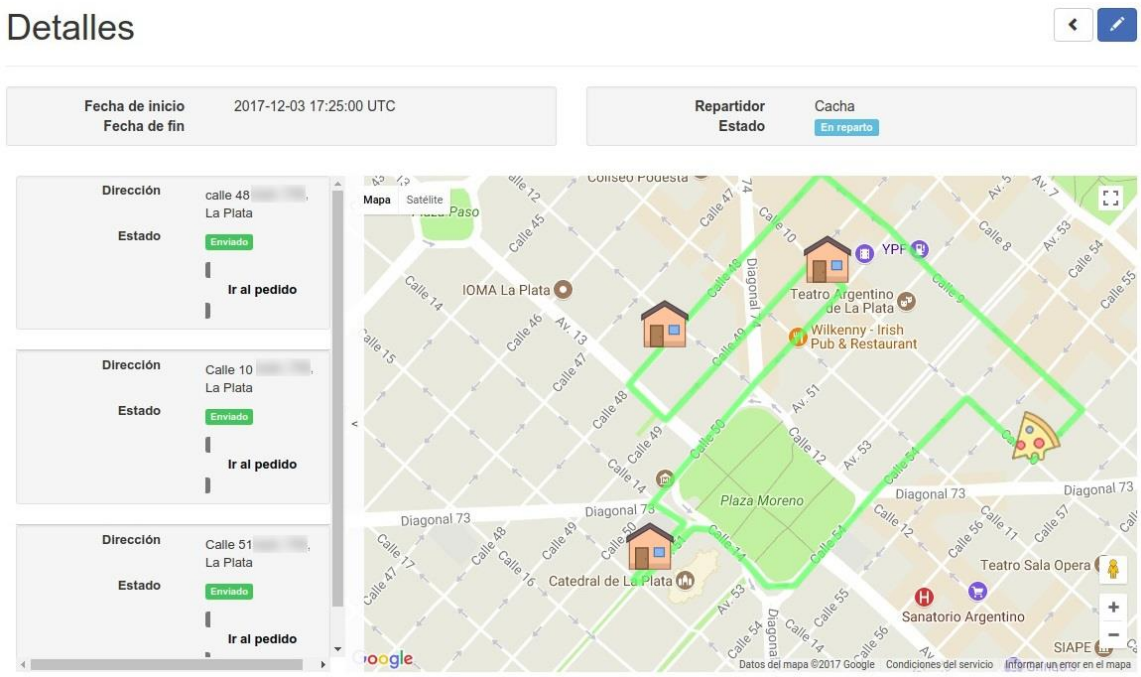

Figura 5.9: *Recorrido* recomendado establecido por la *estrategia* de cálculos de caminos "*Armado de Recorrido más Corto*".

Por simplicidad, y que sea más fácil seguir el funcionamiento que se muestra a continuación en este capítulo, los clientes involucrados en la *entrega* los denominaremos de manera genérica, *Cliente 1*, *Cliente 2* y *Cliente 3*. A continuación se describe quien es cada uno de estos clientes junto a la dirección de su casa para poder entender los recorridos.

- *Cliente 1*: Elvira Constantini - Calle 51 e/ 14 y 15
- *Cliente 2*: Leopoldo Martinez - Calle 10 e/ 49 y 50
- *Cliente 3*: Dionisio Montiel - Calle 48 e/ 12 y 13

En la Figura 5.10 se puede apreciar la pantalla que visualiza cada cliente cuando todavía el *repartidor* no dio inicio al reparto. Se puede observar remarcada la casa de cada uno de ellos. Además, todos los clientes tienen referencia de donde se encuentra posicionado el *negocio* donde se realizó el pedido.

Para poder apreciar en simultaneo lo que visualiza cada cliente en el mismo instante de tiempo, se acoto la pantalla a mostrar haciendo foco solo en el mapa que cada uno puede observar. Esto se aplica a todas las siguientes imágenes que muestran información de los clientes involucrados en la *entrega*.

Figura 5.10: Visualización que tienen los clientes cuando todavía el *repartidor* no comenzó a realizar el reparto de la *entrega*.

En el caso de que la *estrategia* de cálculo de recorridos seleccionada hubiese sido "*Armado de Recorrido Aleatorio*", descripta en Sección 4.2 como *Random*, el *negocio* recibirá una pantalla similar a la visualizada en la Figura 5.11. Como se mencionó para la estrategia anterior, los clientes y el *repartidor* no ven afectadas sus visualizaciones, por un lado, el *repartidor* continua observando una pantalla similar a la Figura 4.8 (Sección 4.4) ya que aún no dio inicio al reparto y por el otro los clientes siguen recibiendo las pantallas mostradas en la Figura 5.10 ya que estos aún no ven el *recorrido* sugerido para el *repartidor*.

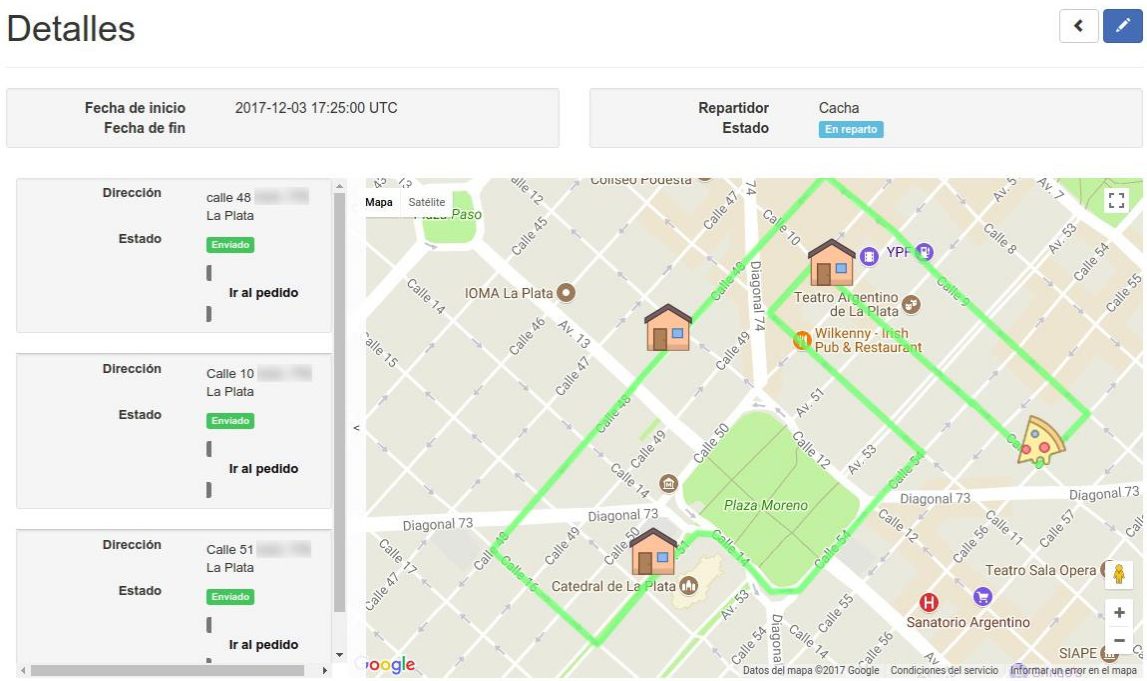

Figura 5.11: *Recorrido* recomendado establecido por la *estrategia* de cálculos de caminos "*Armado de Recorrido Aleatorio*".

Al observar las Figuras 9 y 10 se puede apreciar como cada *estrategia* brinda caminos distintos para las mismas *ordenes* de pedidos.

- *Reparto de la entrega de un pedido*

  Tener en cuenta que el camino sugerido para el reparto que se mostrará a continuación se calculó usando la *estrategia* de "*Armado de Recorrido más Corto"* (el cual se mostró en la Figura 5.9).

  Una vez que el *repartidor* da comienzo al *reparto*[34], esto genera que visualice en la aplicación móvil el *recorrido* recomendado para este y que se empiece a enviar la *posición* geográfica[35] del *repartidor* a la aplicación Web. En la Figura 5.12 se puede apreciar la pantalla tanto del *repartidor* como del *negocio*.

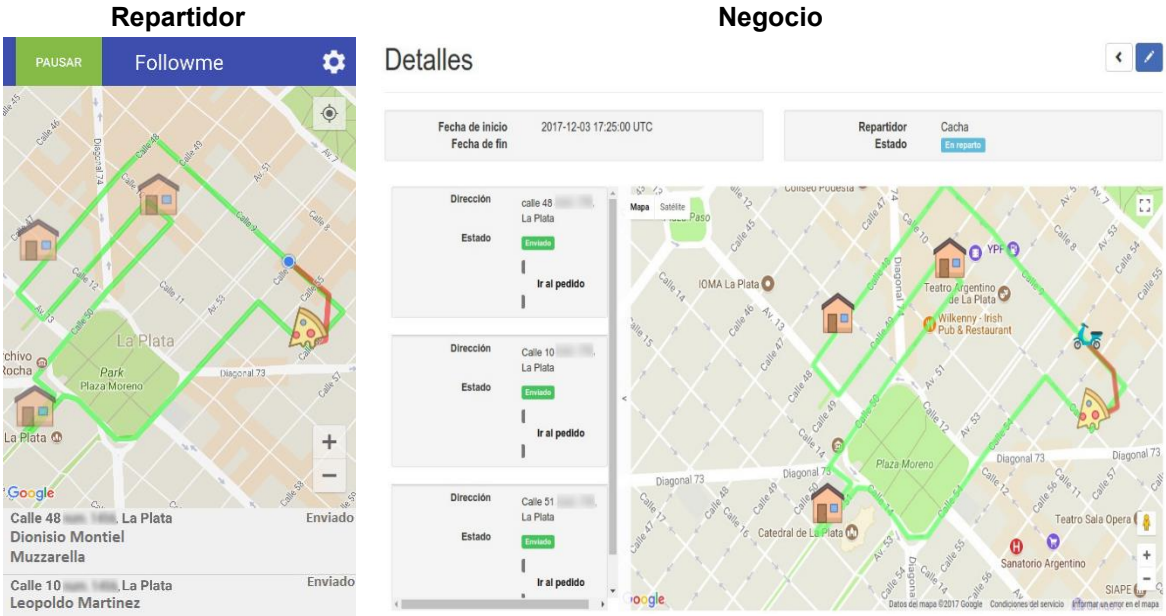

Figura 5.12: El *repartidor* avanza en el *recorrido* del reparto de la *entrega*.

A partir de que el *repartidor* da comienzo al reparto, los clientes pueden seguir además del *estado* de la *orden* y puede visualizar el *recorrido* que realiza el *repartidor* hasta entregar su *orden*. En la Figura 5.13 se puede apreciar como los tres clientes van recibiendo la posición actual del *repartidor*.

Como se puede apreciar en la Figura 5.13 cada cliente puede ajustar el zoom del mapa y acorde a esto, poder ver más nivel de detalle en el recorrido que está haciendo el *repartidor*. Esto se puede apreciar al observar el mapa del *Cliente 2*.

[34] Cabe mencionar como fue explicado en la Sección 4.4 que el *repartidor* tiene que configurar su aplicación para poder recibir así la información de las ordenes que conforman las entregas que se le van asignando.

[35] Los cambios de posición de *repartidor* fueron simulados con posiciones geográficas que permitieron hacer las diferentes pruebas.

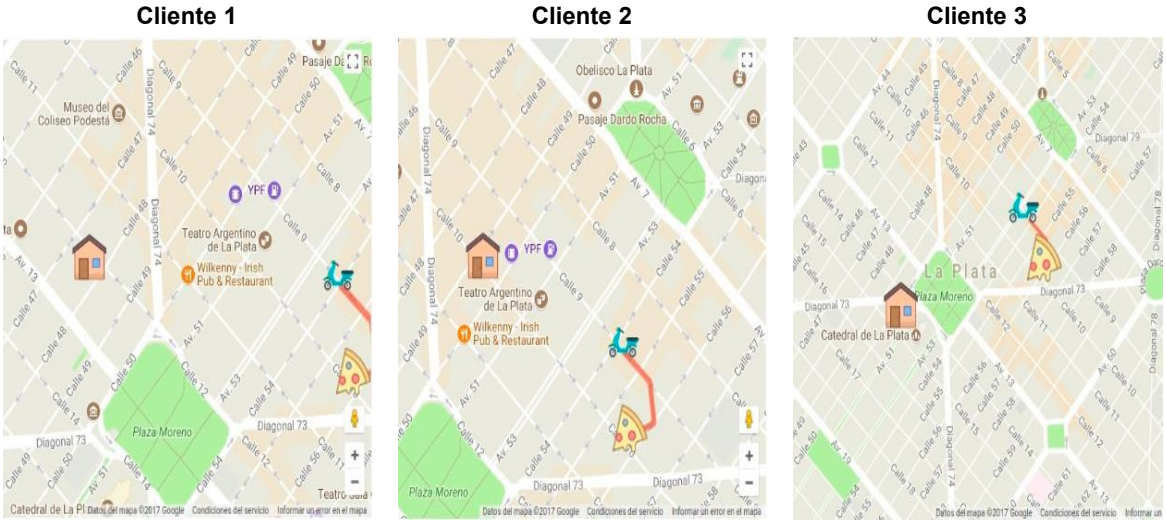

Figura 5.13: Clientes recibiendo la *posición* actual del *repartidor*.

Supongamos que el *repartidor* entrega el *pedido* al *Cliente 1* (realizando la acción correspondiente como se mostró en la Figura 4.10 de la Sección 4.4). En la Figura 5.14 se puede apreciar la pantalla tanto del *repartidor* como del *negocio* actualizadas, se puede observar que el pedido entregado ahora figura como "*Finalizado*".

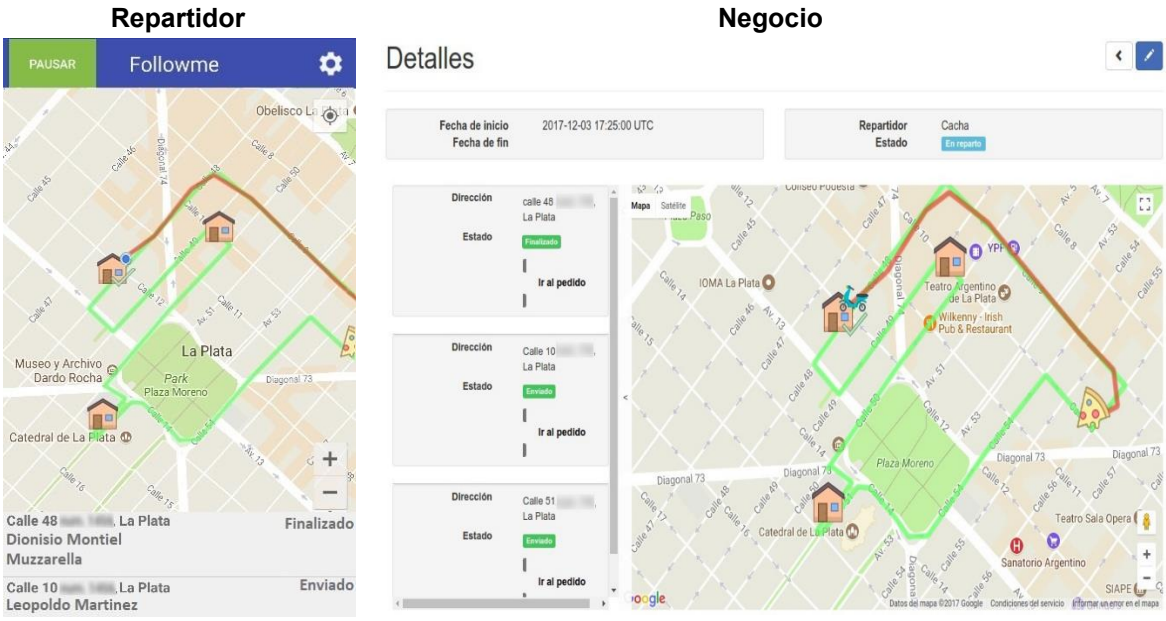

Figura 5.14: El *repartidor* entrega el *pedido* al *Cliente 1* de su *recorrido*.

Simultáneamente a las pantallas mostradas en la Figura 5.14, los clientes pueden seguir el *recorrido* que realiza el *repartidor* hasta entregar su pedido. En la Figura 5.15 se puede apreciar como dos de los clientes van recibiendo la posición actual del *repartidor*, mientras que el *Cliente 1* ya recibió su pedido.

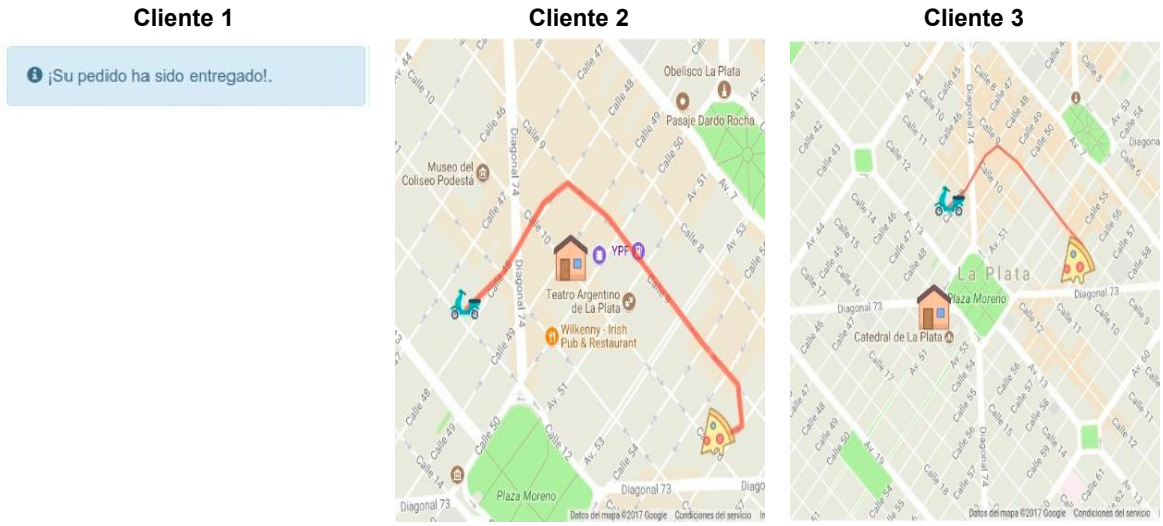

Figura 5.15: El *pedido del Cliente 1* fue entregado y los dos clientes restantes esperan sus pedidos.

Supongamos que el *Cliente 2* cancela su pedido, en la Figura 5.16 se puede apreciar la pantalla que recibe tanto el *repartidor* como del *negocio*. En particular, esto afecto el *recorrido* que tenía que realizar el *repetidor,* ya que el mismo debió ser recalculado.

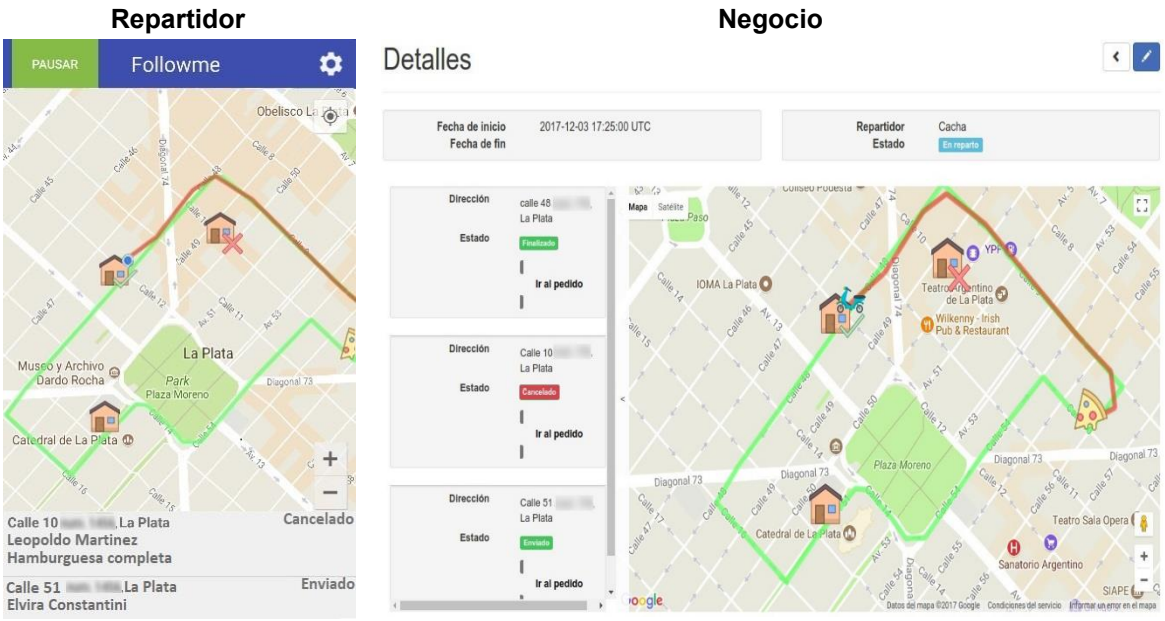

Figura 5.16: Se canceló una *orden* de la *entrega*.

Se puede observar en la Figura 5.16 como uno de los pedidos aparece como "*Cancelado*", y además se puede comparar como los caminos variaron respecto de los mostrados en la Figura 5.14.

En el momento que el *Cliente 2* cancela su *orden de pedido*, recibe una pantalla como se observar en la Figura 5.17. Además, se puede apreciar en esta figura que el *Cliente 3*, al cual todavía le resta entregar su pedido. Si bien esta cancelación afectó el recorrido sugerido al repartidor, esto es algo transparente para el *Cliente 3*.

Cliente 2 **Cliente 3**

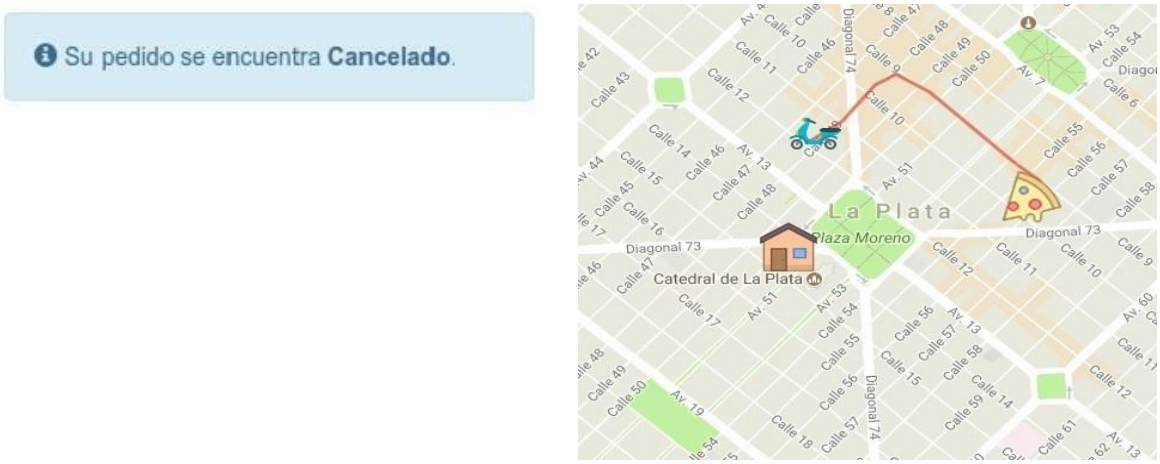

Figura 5.17: El pedido del *Cliente 2* fue cancelado. Información que reciben los *Clientes 2* y *3* ante esta cancelación.

Finalmente, la última *orden* de pedido de la *entrega* es entregada. En la Figura 5.18 se puede visualizar la pantalla tanto del *repartidor* como del *negocio* ante esta situación.

Repartidor **Negocio**

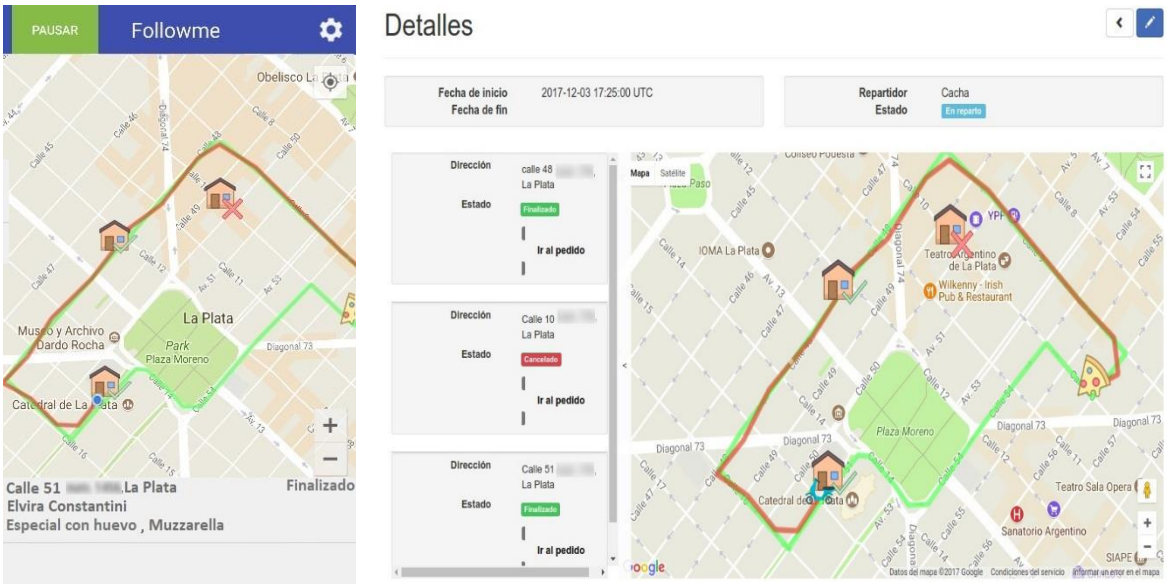

Figura 5.18: Se repartió la última *orden* de la *entrega*.

En simultáneo con la Figura 5.18, el *Cliente 3* puede observar una pantalla como se muestra en la Figura 5.19, donde se indica que su pedido ha sido entregado.

Con este ejemplo, se puede apreciar que a veces contar solo con la posición del *repartidor* no da una idea real de cuanto falta para que el mismo sea entregado, en el caso de haber tenido que entregar el pedido del *Cliente* 2, el *Cliente* 3 hubiera tardado más en recibir el pedido.

Figura 5.19: El pedido del *Cliente* 3 fue entregado, información que recibe este cliente.

De esta manera, se puedo apreciar como es la interacción de los distintos actores involucrados en el prototipo. Y como cada uno de ellos va recibiendo la información en tiempo real.

6. Conclusiones y Trabajos Futuros

En la tesina presentada se pudo apreciar el diseño de solución de modelado para el seguimiento continuo del posicionamiento de las entregas realizadas mediante repartos a domicilio. En el Capítulo 3, se presentó un modelo flexible y adaptable a nuevos requerimientos [Weyns et al., 2015]. También se fueron mencionando en distintos patrones de diseño [Gamma et al, 1995] que fueron empleados en el momento del diseño de la solución

En el Capítulo 3 también se puedo apreciar cómo fue modelado, en particular, el comportamiento para brindar soporte al seguimiento continuo del posicionamiento de las ordenes, es decir, que cada orden tenga un posicionamiento continuo en tiempo real. Para esto, se decidió tomar la posición que tiene el *repartidor* a lo largo del reparto de la entrega, y así poder extrapolar está a la posición actual de cada orden. Esto permitió abordar uno de los objetivos propuestos.

En base al modelo propuesto, en el Capítulo 4, se presentó el prototipo desarrollado el cual era otro de los objetivos de la tesina. El mismo se focaliza en brindar servicios de seguimiento continuo de entregas de pedidos provistos por un negocio de comidas. Como se pudo apreciar en este capítulo, el prototipo está compuesto de tres partes, una plataforma web que se encargara de la lógica de negocio y sirve para mostrar el recorrido de los repartidores en un mapa, una aplicación móvil que cada *repartidor* tiene instalada en su dispositivo móvil, la cual permitirá recolectar su posición actual a medida que este se va moviendo, por último, una API que brinda acceso a parte de la lógica de negocio y sirve de interfaz de comunicación entre la plataforma web y la aplicación móvil.

Un ejemplo de uso se presentó en el Capítulo 5, donde se mostró en particular, como un repartidor realizaba un reparto, y como cada cliente podía observar la información relacionada a su pedido. El prototipo permito apreciar que es posible recolectar la información del *repartidor* a medida que este se desplaza y también mantener informados al cliente como al *negocio* del lugar en donde se encuentra la *entrega*.

Gracias a la implementación del prototipo, pudimos notar que el posicionamiento continuo en tiempo real de una *entrega* va a estar limitado principalmente por la infraestructura de la red móvil, ya que en gran medida no existen problemas con la obtención de la posición del *repartidor* por medio de GPS a través del dispositivo móvil.

También notamos que, así como la infraestructura móvil, el consumo de la batería por parte del GPS es otra de las principales limitantes y será un factor importante a tener en cuenta para la realización de trabajos futuros.

Por otro lado, no podemos dejar fuera de consideración que la utilización de un sistema de estas características podría generar un alto consumo de datos en los dispositivos móviles de los *repartidores*. Esto debería ser una característica a considerar por parte de cada *negocio*, el cual deberá proveer a los *repartidores* paquetes de datos para poder asegurar el funcionamiento de la aplicación.

La mayor curva de aprendizaje involucrada en esta tesis fue el desarrollo de la aplicación móvil, como así también el manejo de las API de mapas y cálculos de caminos.

A medida que se fue planteando la tesina, pudimos ir identificando distintas características que podrían ser analizadas a futuro tanto en general, a nivel de modelado o a nivel de prototipo. A continuación, se describen algunas de estas características que podrían ser trabajos fututos de esta tesina.

Los siguientes son algunos trabajos futuros relacionados con temas generales:

- Debido a que se busca mostrar la información de posicionamiento lo más cercano a tiempo real, y pensando la misma como un sistema SAAS (*Software as a Service*) [Buxmann et al., 2008], donde el servidor recibiría una alta carga de peticiones, un cambio importante si se desea escalar el prototipo a una aplicación real que pueda ser consultada por una cantidad de usuarios considerable, sería analizar la tecnología utilizada en la plataforma Web y API por alguna orientada a respuestas en tiempo real. En base a esto, se debería analizar cuál sería la mejor para un sistema de este tipo, algo que se detectó una vez avanzada la tesina, y por una cuestión de tiempo no fue abordado, y se implementó con las tecnologías de base que conocíamos.
- Se podría analizar cómo la arquitectura presentada puede ser más granularizada para poder así escalar mejor la funcionalidad provista. En este caso, todos los negocios tienen las mismas funcionalidades, sin embargo, cada uno podría tener requerimientos diferentes, y aun así servicios comunes entre ellos. Esto impactaría en la arquitectura propuesta en la tesina.
- Otra característica que se podría considerar es realizar la recolección de información del uso que los usuarios realizan sobre el sistema, buscando indicios de mejoras que podrían influir sobre todo la experiencia de los usuarios. Esto no solo podría impactar en el modelo propuesto sino también en la implementación.
- Analizar que niveles de señal de red móvil existen en las ciudades en las que se utiliza el sistema, con el fin de demarcar zonas en donde el sistema implementado tiene cobertura, es decir donde es posible tener un seguimiento del reparto fluido o fiable del reparto. Y considerar aquellas zonas donde podría ocurrir que no se reciba información de los repartidores.
- Otra característica que se podría analizar es si existen otras tecnológicas que faciliten el manejo de datos georreferenciados, con el fin de mejorar la performance total del sistema.
- Considerar algún tipo de comunicación por medio de mensajes entre el *negocio* de repartos y los *repartidores*, lo que permitiría y dejaría registro de aquellos casos en los que sea necesario realizar alguna indicación que afecte el reparto. Esto no solo podría impactar en el modelo propuesto sino también en la implementación.

Algunos trabajos futuros en relación al modelo propuesto son:

- Explorar distintos dominios de uso, para poder generar así extensiones concretas de los conceptos necesarios, como fue el caso de Producto (como se mencionó en el

Capítulo 4). Esto también podría impactar en los estados necesarios que cada negocio requiere definir.

- Definir otras estrategias para el cálculo de recorrido, por ejemplo, que las entregas tengan una prioridad, no es lo mismo una entrega de una comida caliente que una fría, donde no impacta tanto el retraso de unos minutos. Y acorde a la prioridad y la menor distancia, por ejemplo, calcular el camino.

Por otro lado, con respecto al prototipo implementado, las siguientes son algunos trabajos futuros que se podrían realizar para la API y aplicación web:

- Como se mencionó en el Capítulo 4, por simplicidad no se realizó autenticación de aplicaciones que consuman los servicios de la API, pero sabemos que es vital implementar algún tipo de autenticación para las aplicaciones que deseen consumir los servicios en un ambiente productivo.
- Además de lo mencionado previamente, sería deseable contar con el manejo de usuarios y perfiles, que permita definir la información y funcionalidades a la que tiene acceso cada usuario.
- Otra característica importante a la hora de pasar de un prototipo a un producto funcional, sería gestionar clientes de cada local de pedidos. Esto se llevaría a cabo permitiendo darse de alta y loguearse en la aplicación a la hora de realizar pedidos. Esto permitiría brindar el servicio para que un usuario pueda ver el historial de sus pedidos, no haga falta cargar datos personales cada vez que genere un pedido, le lleguen recomendaciones de negocios, etc.
- En aquellos campos donde sea necesario ingresar direcciones, como por ejemplo al momento de realizar un pedido, sería importante además de disponer de la posibilidad de ingresar un texto para definirlas, tener un mapa que permita elegir o corroborar la dirección textual para que quien la ingrese pueda validar que tanto la dirección en formato de texto como el posicionamiento de la misma en el mapa son correctos y referencian a la dirección deseada. Al tratarse de un sistema basado en posicionamiento, es muy importante tener las direcciones correctamente cargadas, ya que las mismas intervienen de manera crucial en el funcionamiento del sistema.
- Por ahora el prototipo no informa al cliente que realizó un pedido cuantos pedidos faltan entregar antes de que le llegue el turno al suyo, pensamos que mostrar esta información junto al tiempo estimado de *entrega* podría ser una buena mejora para que los clientes puedan tener una estimación más real del tiempo que puede demorar su pedido en ser entregado. Visualmente les podría parecer que está próximo a ser entregado su pedido, sin embargo puede haber otros pedidos por la zona y los usuarios no pueden apreciar esto con el funcionamiento actual del prototipo.

En el caso de la aplicación móvil, se identificaron los siguientes trabajos futuros:

- Como no es posible garantizar una conexión de datos estable debido los problemas conocidos de la infraestructura móvil, se podrían realizar acciones de contingencia para que la información de posicionamiento que no pudo ser enviada durante el periodo sin conexión llegue a destino cuando esta sea recuperada.

- Tal y como se mencionó en el Capítulo 4, para calcular el recorrido sugerido se tiene en cuenta el sentido de las rutas. Adicionalmente, se podría contemplar también otros factores como puede ser el tránsito, rutas alternativas, o evitar medios de transporte masivo.
- Con el fin de mitigar un excesivo consumo en la batería del dispositivo móvil y de no sobrecargar al servidor con información que no es representativa de un desplazamiento, el envío de posiciones geográficas se podría detener automáticamente cuando no se detecta el desplazamiento del *repartidor* y activarse cuando este lo retoma.
- Por otro lado, se podría contar con detección de proximidad con el fin de que la aplicación agilice el proceso de *reparto*, por ejemplo, la aplicación automáticamente podría intervenir en la transición de estados de las ordenes mostrando el dialogo mencionado en el Capítulo 4 cuando el *repartidor* llegue a uno de los destinos, sin necesidad de que este seleccione sobre una *orden* en particular.

Bibliografía

[Android] Página de Android: https://www.android.com/ (Último acceso 20/06/2017).

[Applegate et al., 2011] Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2011). *The traveling salesman problem: a computational study*. Princeton university press.

[Bhatia and Hilal, 2013] Bhatia, S., & Hilal, S. (2013). A new approach for Location based Tracking. IJCSI International Journal of Computer Science Issues, 10(3).

[Buxmann et al., 2008] Buxmann, P., Hess, T., & Lehmann, S. (2008). Software as a Service. *Wirtschaftsinformatik*, *50*(6), 500-503.

[Chadil et al., 2008] Chadil, N., Russameesawang, A., & Keeratiwintakorn, P. (2008, May). Real-time tracking management system using GPS, GPRS and Google earth. In Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-Con 2008. 5th International Conference on (Vol. 1, pp. 393-396). IEEE.

[CorvusGPS] Página de *CorvusGPS*: https://corvusgps.com (Último acceso 10/03/2017)

[Deitel et al., 2014] Deitel, P., Deitel, H., & Deitel, A. (2014). *Android how to Program*. Prentice Hall Press.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische mathematik, 1(1), 269-271.

[Emmanouilidis et al., 2013] Emmanouilidis, C., Koutsiamanis, R. A., & Tasidou, A. (2013). Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. Journal of Network and Computer Applications, 36(1), 103-125.

[Euler, 1736] Dado que no hay publicaciones de 1736, se toma una de las tantas publicaciones existentes disponibles donde se plantea la teoría de Euler. Coto, E. (2003). Algoritmos Básicos de Grafos. Caracas: Universidad Central de Venezuela, 18-20.

[Fielding, 2000] Fielding, R. T., & Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*. Doctoral dissertation: University of California, Irvine.

[Gamma et al., 1995] Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. Reading: Addison-Wesley, 49(120), 11.

[Glympse] Página de *Glympse*: https://www.glympse.com/about (Último acceso 10/03/2017)

[GoogleMaps] Página de *Google Maps Api*: https://developers.google.com/maps/?hl=es-419 (Último acceso 20/06/2017)

[Hartl, 2016] Hartl, M. (2016). *Ruby on rails tutorial: learn Web development with rails*. Addison-Wesley Professional.

[Kuo-pao Yang et al., 2015] MapBuddies: Web Application for the Travelling Salesman Problem, Kuo-pao Yang, Ranjan Poudel, Nishant Jha, Grace Chenevert, Southeastern Louisiana University, 2015.

[Kruskal, 1956] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society, 7(1), 48-50.

[Landau and Werner, 2012] Landau, R., & Werner, S. (2012). Ethical aspects of using GPS for tracking people with dementia: recommendations for practice. International Psychogeriatrics, 24(3), 358-366.

[Nereus] Página de *Nereus*: http://www.cespi.unlp.edu.ar/nereus (Último acceso 10/03/2017)

[Pathshare] Página de *Pathshare*: https://pathsha.re (Último acceso 10/03/2017)

[Perochon, 2014] Pérochn, S.H.S. (2014). Android: Guía de desarrollo de aplicaciones para Smartphones y Tabletas (2a edición). Ediciones ENI.

[RubyonRails] Página de *Ruby on Rails*: http://rubyonrails.org/ (Último acceso 20/06/2017)

[Sommerville, 2005] Sommerville, I. (2005). *Ingeniería del software*. Pearson Educación.

[Wessels, 2001] Wessels, D. (2001). Web caching. O'Reilly Media, Inc.

[Weyns et al., 2015] Weyns, D., Caporuscio, M., Vogel, B., Kurti, A.: Design for Sustainability = Runtime Adaptation ∪ Evolution. In: the 2015 European Conference on Software Architecture Workshops, pp. 62-69. ACM, New York (2015)

[Yang et al., 2015] Yang, K. P., Poudel, R., Jha, N., & Chenevert, G. (2015) MapBuddies: Web Application for the Travelling Salesman Problem. In International Journal of Research in Advent Technology, Vol. 3, No.12, December2015