BENEMÉRITA UNIVERSIDAD AUTONÓMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"*Desarrolló de aplicaciones en dispositivos móviles utilizando agentes móviles*"

Tesis Profesional

Para obtener el título de:

Ingeniero en Ciencias de la Computación.

Presenta

Ricardo Isai González Herrera

Asesor

Dr. Abraham Sánchez López

Puebla, Pue. Otoño 2016

Resumen.

La tecnología de agentes móviles obtuvo recientemente más importancia no sólo por su capacidad de desarrollar y construir sistemas distribuidos, heterogéneos, e interoperables, sino también por su desarrollo robusto de la red móvil y la comunicación [2, 3].

Sin embargo, son pocos los trabajos que se ocupan de los métodos y herramientas de análisis y diseño de los sistemas de agentes móviles [1, 4, 6].

Además, los sistemas móviles han introducido nuevos conceptos como: la migración, la clonación y los lugares.

Proponemos en este trabajo de tesis, una extensión de lo diagramas más importantes de UML 2.x para modelar los sistemas de agentes móviles con el objetivo de hacer frente a estos tres conceptos.

La intención del trabajo no es propiamente contribuir con el desarrollo de una nueva metodología de desarrollo, sino mediante un ejemplo de una aplicación móvil, mostrar las ventajas que ofrecen este tipo de metodologías.

Al tener ya una propuesta de análisis y diseño para desarrollar aplicaciones móviles nos centraremos en crear una sencilla aplicación móvil siguiendo la metodología, buscando las herramientas, plataformas, arquitecturas, etc., para dar la mejor solución al desarrollo de la aplicación siguiendo la metodología para así corroborar el éxito de la misma y ver en lo que se puede mejorar o modificar.

Con la aplicación realizada mediante la metodología propuesta, iniciaremos con ver los resultados de que esta misma nos provee, como se menciono antes, veremos lo obtenido con lo que se necesitaba y como esto da solución a la necesidad para la que fue hecha la aplicación y lo que se puede editar de la metodología así mismo como de la aplicación final.

Y por último se dan las conclusiones sobre el trabajo, lo que se obtuvo a lo largo del desarrollo y lo que se propone como trabajo futuro.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO I INTRODUCCIÓN

Una parte importante de la ingeniería de software es el desarrollo de metodologías y modelos, más para el desarrollo de aplicaciones móviles es lo que nos lleva a desarrollar este tema. Una aplicación móvil no deja de ser un software.

En primera instancia se vera como se han ido desarrollando las metodologías para el desarrollo de aplicaciones empezando por sus antecesoras, lo que las origino y como se ha dado la evolución. Las aplicaciones móviles, como llegaron a surgir, que es lo que las trajo hasta el día de hoy, viendo desde los inicios de estas hasta la actualidad y el porqué es necesario tenerlas, las ventajas al no contar con metodologías eficientes y prácticas, ya que nos basamos en la ingeniería de software para el desarrollo de aplicaciones se ve cómo es que nace esta rama de la computación.

En segunda instancia describiremos el cambio de paradigmas entre cliente servidor y una arquitectura con agentes, las características primordiales de los agentes móviles, su significa, el impacto en el área de la computación, una breve historia del origen de esta disciplina. El punto de vista de la inteligencia artificial a esta arquitectura y como es que puede ser implementada con el auge de los dispositivos móviles hoy, así mismo un punto de vista desde el paradigma cliente servidor.

Al tener en cuenta la arquitectura de agentes móviles se debe tener un enfoque de estandarización y de herramientas propias para el diseño y desarrollo de este tipo de técnicas.

1.1 La importancia de tener una metodología

La Ingeniería de Software es aquella disciplina que se ocupa del desarrollo, la operación y el mantenimiento del software o programas informáticos, cuya meta es el desarrollo costeable de sistemas de software. Este software es abstracto e intangible. Estos productos de software se desarrollan para algún cliente en particular o para un mercado en general.

El término ingeniería del software empezó a usarse a finales de la década de los sesenta, para expresar el área de conocimiento que se estaba desarrollando en torno a las problemáticas que ofrecía el software. En esa época, el crecimiento espectacular de la demanda de sistemas de computación cada vez más y más

complejos, asociado a la inmadurez del propio sector informático y a la falta de métodos y recursos, provocó lo que se llamó la crisis del software.

Durante esa época muchos proyectos importantes superaban con creces los presupuestos y fechas estimados. La crisis del software finalizó pues se comenzó a progresar en los procesos de diseño y metodologías.

Desde 1985 hasta el presente, han ido apareciendo herramientas, metodologías y tecnologías que se presentaban como la solución definitiva al problema de la planificación, previsión de costos y aseguramiento de la calidad en el desarrollo de software. La dificultad propia de los nuevos sistemas, y su impacto en las organizaciones, ponen de manifiesto las ventajas, y en muchos casos la necesidad, de aplicar una metodología formal para llevar a cabo los proyectos de este tipo.

La experiencia previa en la construcción de estos sistemas mostró que un enfoque informal para el desarrollo del software no era muy bueno, los grandes proyectos a menudo tenían años de retraso. Costaban mucho más de lo presupuestado, eran irrealizables, difíciles de mantener y con un desempeño pobre. El desarrollo del software estaba en crisis. Los costos del hardware se tambaleaban mientras que los del software se incrementaban con rapidez, se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los grandes sistemas.

El objetivo principal que busca la ingeniería de software es convertir el desarrollo de software en un proceso formal, con resultados predecibles, que permitan obtener un producto final de alta calidad y satisfaga las necesidades y expectativas del cliente. La Ingeniería de Software es un proceso intensivo de conocimiento, que abarca la captura de requerimientos, diseño, desarrollo, prueba, implantación y mantenimiento.

Generalmente a partir de un complejo esquema de comunicación en el que interactúan usuarios y desarrolladores, el usuario brinda una concepción de la funcionalidad esperada y el desarrollador especifica esta funcionalidad a partir de esta primera concepción mediante aproximaciones sucesivas. Este ambiente de interacción motiva la búsqueda de estrategias robustas para garantizar que los requisitos del usuario serán descubiertos con precisión y que además serán expresados en una forma correcta y sin ambigüedad, que sea verificable, trazable y modificable.

Cabe destacar que es preciso estudiar tanto los principios como las metodologías para llevar a cabo estas acciones mencionadas, en tanto, la disposición de ese conocimiento es lo que permitirá el diseño y la construcción de programas

informáticos con los cuales se pueda operar de modo satisfactorio en las diversas computadoras personales. Entonces, la ingeniería de software implica un trabajo integral, es decir, se produce un análisis del contexto, se diseña el proyecto, se desarrolla el correspondiente software, se efectúan las pruebas para asegurar su correcto funcionamiento y finalmente se implementa el sistema.

La ingeniería de software es una tecnología multicapa en la que se pueden identificar: los métodos, el proceso (que es el fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos). Como disciplina, establece el proceso de definición de requerimientos en una sucesión de actividades mediante las cuales lo que debe hacerse, se modela y analiza.

Para el diseño y desarrollo de proyectos de software se aplican metodologías, modelos y técnicas que permiten resolver los problemas. En los años 50 no existían metodologías de desarrollo, el desarrollo estaba a cargo de los propios programadores. De ahí la importancia de contar con analistas y diseñadores que permitieran un análisis adecuado de las necesidades que se deberían de implementar.

En la actualidad ha habido muchos esfuerzos que se han encaminado al estudio de los métodos y técnicas para lograr una aplicación más eficiente de las metodologías y lograr sistemas más eficientes y de mayor calidad con la documentación necesaria en perfecto orden y en el tiempo requerido.

Una metodología impone un proceso de forma disciplinada sobre el desarrollo de software con el objetivo de hacerlo más predecible y eficiente. Define una representación que permite facilitar la manipulación de modelos, y la comunicación e intercambio de información entre todas las partes involucradas en la construcción de un sistema. Las metodologías ágiles han ganado popularidad desde hace algunos años, ya que constituyen una buena solución para proyectos a corto plazo, en especial, aquellos proyectos en dónde los requisitos están cambiando constantemente.

Un ejemplo de esto son las aplicaciones para dispositivos móviles, debido a que estás tienen que satisfacer una serie de características y condicionantes especiales, tales como: canal, movilidad, portabilidad, capacidades específicas de las terminales, entre otras, y aun cuando existen miles de aplicaciones para dispositivos móviles que corren en diferentes sistemas operativos iOS, Android, BlackBerry y Windows Mobile; éstas llenan las expectativas de los usuarios hasta cierto punto por su escasa calidad en el desarrollo.

Ya que el uso de metodologías de desarrollo de software no se considera importante en este ámbito, por tanto, los desarrollos para dispositivos móviles, hasta el momento, se han venido realizando, principalmente, de manera desordenada y en la mayoría de los casos por desarrolladores individuales que no aplican métodos de ingeniería de software que garanticen su mantenibilidad y por lo tanto su calidad.

1.2 ¿Qué es una metodología de desarrollo?

> *"Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo." [8].*

En esta definición Avison y Fitzgerald, presentan una descripción de las metodologías de desarrollo y destacan sus principales componentes, fases, herramientas y técnicas. Sin embargo una metodología es algo más que una colección, puesto que se basa en una filosofía, distinguiéndose de los métodos o de las simples recetas, que marcan unos pasos a seguir y ya está.

Así, las metodologías difieren ya sea por la cantidad de fases, las técnicas de cada fase, el contenido de la fase o en su base filosófica, todo esto se aplica, dependiendo del contexto de desarrollo, tamaño del proyecto o del equipo de trabajo, cultura organizacional, entre otros aspectos, por lo que en el caso de los desarrollos móviles, es de vital importancia su selección, para garantizar un producto de calidad.

1.3 Metodologías orientadas a dispositivos móviles

Existen muy pocas investigaciones hasta el momento sobre el uso de los métodos ágiles en el desarrollo de aplicaciones para dispositivos móviles, estos sistemas computacionales son sistemas que se pueden mover fácil fisicamente y cuya capacidad de cálculo se puede utilizar mientras se desplazan, ejemplo de ello son las computadoras portátiles, asistentes personales (PDAs) y teléfonos móviles o smartphones.

En la actualidad estos sistemas computacionales son llamados aplicaciones también llamadas apps en los smartphones y están presentes en los teléfonos desde hace mucho tiempo, hoy en día se pueden identificar las diferencias en las

tareas que llevan a cabo, la forma en que son diseñadas y la manera de hacerlas rentables. Con anterioridad los móviles contaban con pantallas reducidas y muchas veces no táctiles, y son los que ahora llamamos *feature phones*, en contraposición a los *smartphones*, más actuales. Actualmente encontramos aplicaciones de todo tipo, forma y color, pero en los primeros teléfonos, estaban enfocadas en mejorar la productividad personal: se trataba de alarmas, calendarios, calculadoras y clientes de correo.

Hubo un cambio grande con el ingreso de iPhone al mercado, ya que con él se generaron nuevos modelos de negocio que hicieron de las aplicaciones algo rentable, tanto para desarrolladores como para los mercados de aplicaciones, como App Store, Google Play y Windows Phone Store.

Entre los aspectos distintivos de los sistemas de computación móvil son su frecuente conectividad a la red inalámbrica, su pequeño tamaño, la naturaleza móvil de su uso, sus fuentes de energía y sus funcionalidades que se adaptan particularmente a los usuarios móviles. Debido a estas características, las aplicaciones son inherentemente diferentes a las aplicaciones escritas para su uso en sistemas de computación estacionaria.

Desde finales del 2007 el incremento en aplicaciones móviles y de la ingeniería de software móvil para el desarrollo de las mismas a ido incrementando, sin embargo, son disciplinas aun jóvenes. Tanto su diseño como su ejecución dan signos de maduración con el desarrollo de métricas, arquitectura y otras metodologías. Tomando en cuenta que existen una gran variedad de metodologías, técnicas, marcos y herramientas que se utilizan para el desarrollo de software para sistemas estacionarios, aun son muy pocas las investigaciones en los sistemas móviles.

Aunque las aplicaciones móviles existen en la actualidad; gran parte de herramientas, metodologías y arquitectura de ingeniería de software tienen como principal frente de necesidades los sistemas de escritorio. Como objetivo de este trabajo es mostrar una metodología ágil para el análisis, diseño y desarrollo de aplicaciones móviles utilizando agentes. Tomando ya la propuesta para el desarrollo de aplicaciones móviles se enfocara en el desarrollar una aplicación móvil simple siguiendo la metodología propuesta.

El propósito de esta tesis es mostrar las ventajas que ofrece el trabajar con metodologías de desarrollo propias para cada área de trabajo, la aplicación ejemplo desarrollada con esta metodología tiene como principal objetivo atacar a la alta contaminación ambiental, ayudando al usuario final de una manera denominada inteligencia ambiental. Se considera que hay un largo campo de trabajo e investigación teniendo un amplio camino de trabajos futuros a tener en consideración.

CAPÍTULO II METODOLOGIA PROPUESTA

2.1 El diseño innovador, Sistemas Distribuidos

Hasta hace algunos años, el término sistema distribuido se utiliza principalmente para describir una red de varios sistemas informáticos con memoria separados que están conectados entre sí por una red dedicada. Los equipos que se utilizan en tal sistema distribuido son casi homogéneos, lo que significa que tienen el mismo tipo de procesador y el mismo tipo de sistema de funcionamiento.

La red es más o menos estática: las computadoras rara vez se apagan, las conexiones de red entre los hosts son siempre fiables y proporcionan un ancho de banda constante, cada computadora tiene una dirección IP fija, y el paquete de enrutamiento de red se realiza a través de switches locales. Este tipo de red se utiliza comúnmente para la mayoría de aplicaciones.

Actualmente, sabemos como se ha desarrollado vertiginosamente las tecnologías de redes y computadoras. El Internet como una red de redes con computadoras heterogéneas es un medio ampliamente aceptado como un dispositivo de intercambio de información. El número de personas y empresas que prestan servicios en Internet aumenta continuamente y es incluso superado por el número de usuarios de Internet.

Hay muchos tipos de servicios que se ofrecen a través de Internet, principalmente el correo electrónico y el intercambio de archivos electrónicos. Sin lugar a dudas, el servicio de Internet de mayor éxito es la World Wide Web. Mientras que en el comienzo de la Web era sólo un medio para publicar sus datos en su sitio web, ahora vemos nuevas aplicaciones en la Web que implican una cantidad cada vez mayor de cálculo, la dinámica y las interdependencias.

La mayoría de estas aplicaciones son parte del dominio de comercio electrónico, por ejemplo, las tiendas en línea ó mercados electrónicos. Sin embargo, todavía se construyen utilizando una técnica de diseño tradicional llamada cliente-servidor, en los que un sistema de computo de alto desempeño (servidor) contiene datos que se comparten a través de los sistemas informáticos con menor desempeño (clientes) que acceden al servidor mediante una red.

Dos grandes tendencias se visualizan como focos de interés. La computación ubicua significa que todo podría convertirse en un sistema distribuido. Como las computadoras se vuelven más y más pequeñas, pueden encontrarse no sólo en

los escritorios, sino también en los coches para regular el control de velocidad, en las muñecas para mostrar el tiempo y el control de impulsos, y en los refrigeradores para controlar la temperatura.

La segunda tendencia que queremos mencionar aquí es informática nómada, lo que significa que los usuarios se desplazan de un lugar a otro durante el trabajo, acceder al sistema desde diferentes sistemas informáticos (por ejemplo, por primera vez de un sistema en la oficina a través de LAN en toda la empresa, más tarde desde su casa a través de una conexión).

Sin embargo, los usuarios quieren ver casi el mismo ambiente de trabajo, las mismas aplicaciones, y, sobre todo, los mismos datos. Además, los usuarios nómadas exigen una perfecta integración de los diferentes dispositivos, por lo que es posible cambiar el entorno de trabajo de una computadora de sobremesa a un smartphone en unos pocos segundos.

Todas estas nuevas tendencias requieren nuevas técnicas de programación centradas en redes que se basan en un principio verdadero peer-to-peer. El patrón de diseño cliente-servidor, utilizado con éxito para sistemas distribuidos en redes de área local, no es capaz de hacer frente a todos los retos de los futuros sistemas distribuidos descritos anteriormente y es, en su concepto básico, sigue siendo un paradigma centralizado.

Hoy en día usamos métodos orientados a objetos y lenguajes de programación, porque están mucho más cercanos de los conceptos y necesidades de nuestros clientes y de nosotros. De una forma similar se podría mirar el código móvil como un paradigma que acepta inherentemente la distribución y el trabajo en red como concepto básico y, por lo tanto, se adapta a nuestros nuevos proyectos centrados en la red, así, que nos da la oportunidad de llegar a un nuevo nivel de abstracción y calidad. Este argumento, por encima de todos los detalles técnicos y discusiones, es lo que impulsa a nuestra creencia en el enfoque de agentes móviles.

Los agentes móviles son un tipo especial de código móvil. El código móvil es una técnica en la cual el código se transfiere desde sistemas computacionales que almacenan los archivos fuente a los sistemas computacionales que ejecutan el código. Un ejemplo bien conocido de código móvil son los applets de Java, los cuales son pequeños programas disponibles en un formato de código de bytes portable e interpretable. Los applets se transfieren desde un servidor web a un navegador web con el fin de ser ejecutado como parte de una página HTML [18].

Un agente móvil es un programa que puede migrar desde un host de partida para muchos otros nodos de una red de sistemas informáticos heterogéneos y desempeñar una tarea especificada por su propietario. Funciona de manera autónoma y se comunica con otros agentes y sistemas host. Durante la migración por iniciativa propia, el agente lleva toda su código y datos, y en algunos sistemas que también lleva a una especie de estado de ejecución.

Una diferencia entre los applets de Java y los agentes móviles es el hecho de que los agentes móviles inician el proceso de migración, mientras que la migración de los applets de Java se inicia a partir de otros componentes de software (por ejemplo, el navegador web).

Otra diferencia es que los applets de Java solo migran desde un servidor a un cliente y no salen del cliente para migrar a otro cliente o de vuelta al servidor. la vida de un applet está ligada a la vida útil de la página web y es parte de ella y muere cuando el navegador, se cierre o se solicita otra página Web. En contraste con esto, los agentes móviles generalmente migran más de una vez. Piense en un agente móvil que viaja a varios anfitriones para cobrar precios de un producto deseado.

Como consecuencia de ello, los componentes de software que diseñamos tendrán que ser tan dinámicos y móviles como los usuarios finales y las redes. Tendrán que ser proactivos y actuar de una manera muy autónoma, o de forma mas “inteligente”. El concepto de agentes de software surge de estas demandas.

2.2 De Cliente-Servidor a Agentes Móviles

Comenzamos nuestra exploración de los agentes móviles en el origen de la noción de agente, y comparamos el término ya generalizado de agentes inteligentes para nuestra comprensión de los agentes móviles. Al mirar la historia de los agentes móviles aprenderemos que este tema de investigación tiene sus raíces en la computación distribuida mas que en la inteligencia artificial.

En esta sección tratamos de converger en la idea de agentes móviles a partir de dos lados.

En primer lugar, se discuten los agentes de software como un concepto desarrollado en el área de la inteligencia artificial a mediados de la década de 1970. La gente de la disciplina de inteligencia artificial define como agentes de software a los que tiene algunas características obligatorias, que no incluyen la movilidad. A veces, se menciona que podría surgir algunos beneficios del uso de

agentes móviles en el futuro; de lo contrario, se afirma que la movilidad del agente es una característica bastante inútil.

Este punto de vista puede ser mejor descrito por la afirmación de que los agentes móviles son una solución en busca de un problema.

En segundo lugar, tratamos de definir los agentes móviles desde el punto de vista de la ingeniería de software y sistemas distribuidos. Veremos que los agentes móviles en nuestra comprensión no tienen mucho que ver con la inteligencia artificial, sino que deben considerar como otro paradigma de diseño para un tipo especial de sistemas distribuidos. Desde este punto de vista, el énfasis radica en la investigación de las consecuencias de la movilidad de código.

Naturalmente, ambas comunidades de investigación pueden beneficiarse mutuamente. Algunos trabajos ya se han hecho para añadir agentes inteligentes con la movilidad como una característica común, y, recientemente, las personas han comenzado a trabajar para que los agentes móviles sean mas inteligentes, por ejemplo, para ayudarlos a planificar sus itinerarios. Con suerte, las dos ramas de la investigación se unirán entre sí con el tiempo.

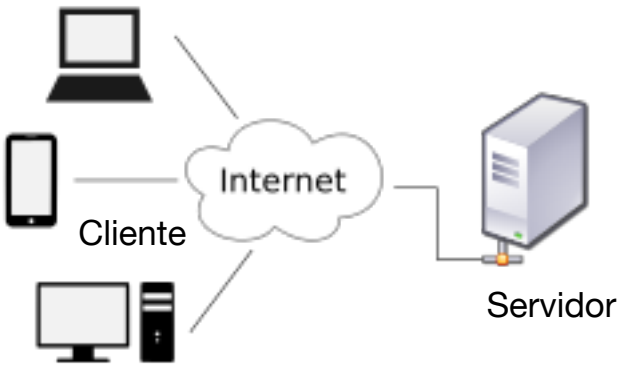


Figura 2.1. Modelo Cliente Servidor

2.3 Punto de vista de la Inteligencia Artificial

Vamos a empezar con la noción de agentes de software. La palabra agente deriva de la palabra latina para el actor, es decir, una persona que actúa en nombre de otro. En diferentes idiomas la noción de agente se utiliza con diferentes significados. En los países de habla Inglés, por ejemplo, la palabra agente se utiliza a menudo en un contexto más general, mientras que en los países de habla alemana en su mayoría un agente trabaja para el servicio secreto.

Por lo general, se emplea un agente de bienes raíces para ayudar en el alquiler o la compra de una casa, y un agente de viajes se consulta para ayudar en la planificación de unas vacaciones. En la ciencia física, un agente puede ser una sustancia activa que causa una reacción. Otras ciencias también utilizan el término agente. Por ejemplo, en ciencias jurídicas un agente provocador es una persona contratada para incitar a personas sospechosas de cometer alguna acción ilegal que les hará pasible de sanción.

En informática, el término agente se ha utilizado desde mediados de la década de 1970. Fue introducido en el área de la inteligencia artificial. Muchos autores se refieren a un trabajo propuso por Hewitt [1977] como origen del término agente. De acuerdo con Foner [1997], la primera referencia se remonta a Vannevar Bush y Douglas Engelbart a finales de 1950 y principios de 1960.

Hoy en día, el término agente es (por desgracia) convertido en una palabra de moda para señalar las características innovadoras del sistema. Por ejemplo, algunos clientes de correo electrónico se llaman agentes de correo, aunque no hacen nada aparte de la tarea habitual de entrega y recogida de mensajes de correo electrónico de su buzón.

Un agente de software es una entidad de software que realiza de forma continua las tareas dadas por un usuario dentro de un entorno restringido en particular. La entidad de software involucrada puede ser un programa de ordenador, un componente de software, o, en el sentido de los lenguajes de programación orientados a objetos, sólo un simple objeto.

Sin embargo, los verdaderos agentes de software deben verse como una extensión del concepto más general de los objetos o componentes de software. Mientras que los objetos de software son pasivos, los agentes están activos.

La definición de lo que constituye exactamente un agente de software se ha debatido intensamente en la comunidad de investigación durante varios años. Aunque este debate continúa, hay un entendimiento común de que una entidad de software debe exhibir ciertas características mínimas para calificar como un agente:

Autonomía: Los agentes operan y se comportan de acuerdo a un plan hecho a sí mismo que se genera de acuerdo con la tarea dada por el usuario. Los agentes no necesitan cada paso de este plan estipulado por su propietario con antelación, y no preguntan a su dueño para la confirmación de cada paso.

Comportamiento social: Los agentes son capaces de comunicarse con otros agentes o seres humanos por medio de un lenguaje de comunicación entre agentes. La comunicación puede ser restringido a puro intercambio de información o puede incluir sofisticados protocolos para la negociación. Una rama separada de la investigación se ocupa del problema de la multiplicidad de agentes que trabajan juntos en una sola tarea en los llamados sistemas multi-agente. En este caso, el comportamiento benévolo es necesario para una empresa exitosa.

Reactividad: Los agentes perciben su entorno por algún tipo de sensores y son capaces de reaccionar a los eventos identificados.

Proactividad: No sólo los agentes reaccionan a los estímulos de su entorno, sino que también son capaces de tomar la iniciativa y planificar de forma activa. B. Le Du explica esto con la siguiente metáfora: "La diferencia entre una Autómata y un agente es algo así como la diferencia entre un perro y un mayordomo. Si envía su perro para comprar una copia del New York Times todas las mañanas, va a volver con la boca vacía si el puesto de periódicos resulta haberse quedado sin este diario específico un día. Por el contrario, el mayordomo probablemente tome la iniciativa y compre una copia de The Washington Post, ya que él sabe, que a veces se leen en su lugar. “ [Bradshaw, 1996, p. dieciséis]

Ahora podemos tratar de añadir a nuestras definiciones de agentes de software:

> Los agentes de software móviles son programas informáticos que actúan como representantes en la red mundial de sistemas informáticos. El agente conoce a su dueño, sabe sus preferencias, y aprende mediante la comunicación con su dueño. El usuario puede delegar tareas al agente, el cual es capaz de buscar eficientemente en la red moviendo al proveedor de servicios o información. Los agentes móviles soportan los usuarios nómadas debido a que el agente puede trabajar asíncronamente mientras el usuario está desconectado. Por último, el agente informa de los resultados de su trabajo al usuario a través de diferentes canales de comunicación como correos electrónicos, sitios web, páginas, o teléfonos móviles.

En esta definición, se pueden encontrar muchas de las características de los agentes de software que hemos descrito anteriormente. Un agente móvil actúa en nombre de un usuario; conoce a su usuario y llega a él o ella en el mejor tiempo. Tiene comportamiento social, ya que es capaz de comunicarse con el usuario, servicios, o incluso otros agentes. Trabaja de manera proactiva, ya que puede, por ejemplo, estar en contacto con su propietario por muchos medios de comunicación.

La propiedad adicional de la movilidad puede verse como una extensión muy directa, al menos desde un punto de vista humano, ya que va bien con nuestra comprensión natural de la forma de buscar información en un entorno distribuido.

2.4 El punto de vista de los Sistemas Distribuidos

En contraste con la definición más acercada para el usuario final orientado a agentes móviles que describimos en la sección anterior, en el que la movilidad era simplemente una característica agradable, empezamos aquí con una definición que llama más la atención en los aspectos técnicos de la movilidad del agente .

> Los agentes móviles se refieren a programas autónomos e informáticos de identificación, que se incluye con su código, datos y estado de ejecución, que se pueden mover dentro de una red heterogénea de los sistemas informáticos. Estos pueden suspender su ejecución en un punto arbitrario y transportarse a otro sistema informático. Durante esta migración, el agente se transmite por completo, es decir, como un conjunto de código, datos y estado de ejecución. En el sistema computacional de destino, la ejecución de un agente se reanuda exactamente en el punto en el que se suspendió antes.

Simplemente hablamos de programas computacionales o procesos en el significado de los sistemas operativos que son capaces de congelarse a sí mismos, pasar a otros sistemas computacionales, y reanudar ahí la ejecución. Esta definición más técnica puede verse como un complemento de la definición guiada por el usuario, simplemente dirigida a un nivel más bajo de abstracción.

En este caso, los agentes móviles se ven desde el punto de vista de la ingeniería de software y sistemas distribuidos. Pueden ser considerados como un paradigma de diseño adicional en el área de la programación distribuida y un complemento útil de las técnicas tradicionales, tales como la arquitectura cliente-servidor.

2.5 Características de los Agentes Móviles

Recordando nuestra definición de los agentes móviles y considerándolos como un nuevo paradigma de diseño para sistemas distribuidos que complementan las técnicas más tradicionales, ahora podemos identificar cuatro características de los agentes móviles:

1. Los agentes móviles se utilizan tradicionalmente en grandes zonas y redes heterogéneas en las que no se pueden hacer suposiciones relativas bien a la fiabilidad de los equipos conectados o la seguridad de las conexiones de red.

2. La migración de agentes móviles se inicia por el agente (más precisamente, su programador), en contraste con sistemas de objetos móviles, en el que el objeto a migrar se inicia por el sistema operativo subyacente o middleware.

3. La migración de agentes móviles se hace para acceder a los recursos disponibles sólo a otros servidores en la red y no sólo para el balance de carga, como en los sistemas de objetos móviles.

4. Los agentes móviles son capaces de migrar más de una vez, esta característica se denomina a veces la capacidad de saltos múltiples. Después de que un agente móvil ha visitado el primer servidor, puede migrar a otros servidores para continuar con su tarea, mientras que el código móvil se transfiere sólo una vez en el paradigma de evaluación remota y en el paradigma de código en demanda.

Vamos a echar un vistazo más de cerca a esta definición técnica antes de continuar: por el término código nos referimos a algún tipo de representación ejecutable de programas computacionales. Con lenguajes de script, como Perl o TCL, este podría ser el código fuente; con el lenguaje de programación Java, este es el intermedio portátil en formato JAVA en codigo de bytes; y con el lenguaje de programación C, que podría ser el formato de lenguaje de máquina ejecutable por un procesador único [16].

Por el término datos nos referimos a todas las variables del agente (en lenguajes orientados a objetos, es el conjunto de todos los atributos del objeto correspondiente). Por último, por el término estado nos referimos a la información sobre el estado de ejecución del agente.

Dejamos abierto lo que comprende exactamente el estado de ejecución. Podría ser información bastante completa desde el interior de la máquina subyacente (virtual) sobre la pila de llamadas, valores de registro, y los apuntadores a instrucciones.

Las herramientas de desarrollo de agentes móviles basadas en Java no proporcionan una determinación sofisticada de la pila de ejecución, debido a algunas limitaciones de la máquina virtual de Java.

2.5.1 Agentes Móviles como un nuevo paradigma de diseño

Podemos describir el paradigma de agentes móviles de la siguiente manera. En el Sitio S_A, una maquina virtual M_A^T tiene que saber como en forma de código, en cual es ejecutado. Durante esta ejecución el código se da cuenta de que necesita tener acceso a otros recursos que actualmente se encuentran en el sitio S_B. Por lo tanto, M_A^T interactúa con M_B^U para transmitir el código, junto con alguna información sobre el estado de ejecución actual. En el sitio S_B, una máquina virtual M_B^U ejecuta el código, facilitando el acceso a los recursos ubicados en el S_B.

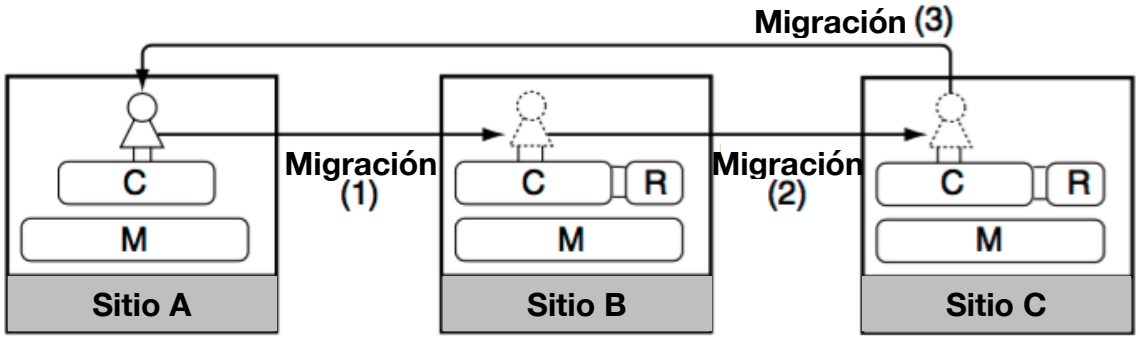


Figura 2.2 El paradigma del Agente Móvil

En la figura 2.2 podemos observar como los agentes están definidos con pequeñas figuras que se muestran arriba. Componentes de código estacionario que son agencias para indicar que estas son dinámicamente limitadas a este código

Más tarde, el código puede decidir que necesita otros recursos en otros sitios, (por ejemplo, el sitio S_C), en cuyo caso el código se migra a otra computadora nuevamente [17].

2.5.2 Los Agentes Móviles necesitan un entorno

Obviamente, los agentes móviles necesitan algún tipo de entorno para estar "vivos". Lo que hemos simplificado como máquina virtual en la figura 2.2 en realidad se compone no sólo del intérprete para el lenguaje de programación sino también del entorno de ejecución para los agentes, que es llamado agente servidor o agencia.

Una agencia es responsable de alojar y ejecutar en paralelo los agentes y les proporciona un entorno para que puedan acceder a los servicios, comunicarse entre sí, y, por supuesto, migrar a otras agencias. Una agencia también controla la ejecución de los agentes y protege el hardware subyacente del acceso no autorizado por parte de agentes maliciosos.

Hoy en día, existen muchos tipos diferentes organismos. Muchas universidades y también algunas empresas han desarrollado sus propios productos. Una simple agencia solo en raras ocasiones tiene sentido, en particular en el caso de agentes móviles. Además, incluso una red de varias agencias todavía no es emocionante a menos que algunos agentes móviles estén vagando por la red, por el uso de servicios para cumplir con alguna tarea.

2.5.3 Una breve historia de los Agentes Móviles

Como hemos señalado, el paradigma de agentes móviles se basa principalmente en la idea de código móvil. Por lo tanto, en cierta medida, debemos tener en cuenta al código móvil como un antepasado de los agentes móviles.

2.5.3.1 Agentes Móviles

Fue en 1994 que James E. White, afiliado con General Magic Inc. en ese momento, publicó un documento que inició la investigación a fondo de lo que llamamos agentes móviles hoy en día. Este artículo fue publicado más tarde en un libro editado por Bradshaw [1996].

En el, White introdujo la tecnología Telescript, que comprende un entorno de ejecución y un lenguaje de programación dedicado para los agentes móviles. Este lenguaje ya ofrecía la mayor parte de los aspectos muy importantes y abstracciones de todos los conjuntos de herramientas de agentes móviles actuales [19].

El mayor desarrollo de Telescript cayo cuando se hizo evidente que esta tecnología no sería capaz de competir con Java como base común para la mayoría de los kits de herramientas de agentes móviles. Por su trabajo sobre los agentes móviles, General Magic recibió una patente en EE.UU. en el año de 1997 [White et al., 1997].

El siguiente hito en la investigación de los agentes móviles fue el papel de Chess, que describe un marco para agentes itinerantes como una extensión del modelo cliente-servidor.

Los agentes itinerantes se envían de un equipo de origen y luego vagan por una red de servidores hasta que hayan cumplido la tarea del usuario. Junto con una breve discusión de los beneficios de los agentes itinerantes, los autores describen posibles dominios de aplicación, la arquitectura de los puntos de encuentro, agente de idiomas para desarrollar tales agentes, y una discusión de los problemas de seguridad.

Un segundo documento importante, publicado casi al mismo tiempo, se refirió a las ventajas de los agentes móviles contra las técnicas basadas en cliente-servidor. El artículo fue publicado más tarde por Chess, y los autores concluyeron:

> Aunque ninguna de las ventajas individuales de los agentes móviles dados anteriormente es abrumadoramente fuerte, creemos que las ventajas agregadas de los agentes móviles son abrumadoramente fuertes, debido a que: Mientras alternativas a los agentes móviles pueden avanzar para cada una de las ventajas individuales, no existe una única alternativa a la totalidad de la funcionalidad soportada por un marco de agentes móviles. [Harrison, 1995, p. 17]

Desde el proyecto inicial de General Magic, la comunidad de investigación interesados en agentes móviles ha dejado de crecer, lo que implica muchas cuestiones interesantes de investigación, por ejemplo, en el ámbito de la seguridad, que son consecuencia de la simple idea de código en movimiento.

2.5.3.2 Estandarización

El primer enfoque de normalización a kits de herramientas de agentes móviles fue publicado por Milojicic en 1999 [20]. El Mobile Agent System Interoperability Facility (MASIF), anteriormente conocido como Mobile Agent Facility (MAF), fue respaldada por empresas y departamentos de investigación que estaban activos en la investigación de agentes móviles a principios de los años (por ejemplo, IBM, GeneralMagic, y GMD Fokus) y fue publicado como un estándar OMG en 1998.

Las bases MASIF en CORBA como la infraestructura del sistema. Varios conjuntos de herramientas de agente están disponibles, o al menos la reivindicación que son compatible por MASIF. Los dos más famosos son Herretes y Saltamontes.

MASIF es en realidad un conjunto de definiciones e interfaces para juegos de herramientas de agentes móviles interoperables. Consiste en una interfaz para la transferencia y gestión de agentes (MAFAgentSystem) y una interfaz para localizar y nombrar agentes móviles (MAFFinder).

El estándar define cómo entender el concepto como agente, sistema de agente, lugares, regiones, y varios otros conceptos básicos e ideas. MASIF no define nada relacionado con la comunicación entre agentes, ya que este problema se aborda ampliamente por CORBA.

Para hacer frente a los problemas de seguridad de los agentes móviles, MASIF también se basa en los principios de CORBA. No puede decirse que MASIF fracasó, pero se han desarrollado muy pocos juegos de herramientas para cumplir con esta norma. Una de las razones de esto podría ser la relación estrecha con CORBA.

Otro enfoque estándar en tecnología de agentes FIPA (www.fipa.org), que se centra principalmente en cuestiones relacionadas con la interoperabilidad de los agentes y, por lo tanto, define los temas de la comunicación de agentes, incluyendo lenguaje de comunicación (ACL), protocolos de transporte de mensajes y ontologías, pero no tiene en cuenta la movilidad del agente.

La especificación FIPA básica para una arquitectura abstracta (FIPA 00001) para un sistema de agente omite expresamente la movilidad del agente.

En 2000, un nuevo conjunto de especificaciones FIPA, que incluían FIPA 00087 para la movilidad agente, fue puesto en libertad.

2.5.3.3 ¿Por qué los Agentes Móviles son una buena idea?

Aunque los agentes móviles proporcionan un enfoque nuevo e interesante para sistemas distribuidos, tiene que haber pruebas claras a favor de los agentes móviles antes de que sean sustituidas por las técnicas más tradicionales. Sin embargo, aunque creemos que los agentes móviles son la tecnología más prometedora para resolver la mayoría de los problemas del futuro en red, hay que decir que también creemos que los agentes móviles complementan muchas técnicas más antiguas en lugar de reemplazarlas.

Presentamos cuatro importantes ventajas técnicas en detalle. Es este conjunto de ventajas técnicas básicas que se abre la posibilidad para aplicaciones mejoradas y típicas.

1. La delegación de tareas. Debido a que los agentes móviles son simplemente un tipo más específico del agente de software, un usuario puede emplear un agente móvil como representante al que el usuario puede delegar tareas.

   En lugar de utilizar los sistemas informáticos como herramientas interactivas que son capaces de trabajar sólo bajo el control directo de un usuario, los agentes de software autónomos tienen como objetivo el cuidado de tareas enteras y trabajar sin contacto y control permanente.

   Como resultado, el usuario puede dedicar tiempo y atención a otras cosas más importantes. Por lo tanto, los agentes de software móviles son un buen medio para hacer frente a la sobrecarga de información constante que experimentamos.

2. Procesamiento asíncrono. Una vez que los agentes móviles se han inicializado y configurado para una tarea específica, dejan físicamente el sistema informático de su propietario y de ahí deambulan libremente a través de Internet. Sólo para esta primera migración se debe establecer una conexión de red.

   Esta característica hace que los agentes móviles adecuados para la informática nómada, es decir, los usuarios móviles pueden iniciar sus agentes desde dispositivos móviles que ofrecen sólo limitado ancho banda y enlaces de red volátiles. Debido a que el agente es menos dependiente de la red, va a ser más estables que las aplicaciones basadas en cliente-servidor.

3. Interfaces de servicios adaptables. Las técnicas actuales en los sistemas distribuidos que ofrecen interfaces de servicios de aplicaciones, por lo general como una colección de funciones, constituyen sólo el mínimo común denominador de todos los clientes posibles.

   Como consecuencia, la mayoría de las funciones de interfaz son más o menos primitivas, y los clientes probablemente tendrán que utilizar un flujo de trabajo conectando estas funciones con el fin de ejecutar una operación compleja, guiada por el usuario. Si la sobrecarga de comunicación para el intercambio de mensajes entre el cliente y el servidor es alto comparado con el tiempo de ejecución de cada función, lo que tendría sentido para ofrecer funciones agregadas y más avanzadas como combinaciones de los primitivos.

Sin embargo, debido a que es difícil de localizar a todos los escenarios posibles por adelantado o incluso durante el tiempo de ejecución, estas funciones no suelen ser ofrecidos por la interfaz de usos múltiples del servidor. Los agentes móviles pueden ayudar en esta situación, ofreciendo una oportunidad de diseñar una interfaz orientada al cliente que está optimizada para el cliente (usuario), pero que es adaptable a diferentes interfaces de servidor.

La clave es el uso de un agente móvil para traducir las funciones más complejas y controladas por el usuario de cara al cliente internacional en las funciones primitivas de ajuste que se ofrecen en el nodo de servidor.

El agente móvil simulará una interfaz constante y altamente especializada para el cliente (usuario) mientras habla con cada servidor en su propio idioma, lo que permitirá a los servidores que se vuelven más simples y más generalizables.

4. Envío de código frente a envío de datos. Esta es probablemente la ventaja más citada de los agentes móviles, y está en estrecha relación con las interfaces de servicios adaptables.

Las interfaces de servicio con frecuencia sólo ofrecen funciones primitivas para acceder a bases de datos. Una sola llamada, por tanto, puede dar lugar a una enorme cantidad de datos que se envían de vuelta al cliente debido a la falta de precisión en la solicitud.

En lugar de transferir datos al cliente, donde se procesa, se filtra, y probablemente causa una nueva solicitud (datos de envío), este código puede ser transferido a la ubicación de los datos (código del envío) por medio de agentes móviles.

En este último caso, sólo los datos relevantes (es decir, los resultados después de procesamiento y filtrado) se envía de vuelta al cliente, lo que reduce el tráfico de red y ahorra tiempo si el código para el filtrado es menor que los datos que deben ser procesados. Esta ventaja se ha examinado en los últimos 5 años por diferentes grupos de investigación de diferentes dominios de aplicación, y por lo general ha sido verificado.

2.6 M-UML como metodología de desarrollo para Agentes Móviles

El Lenguaje de Modelado Unificado (UML) es un lenguaje para la especificación, visualización y documentación de los sistemas de software orientado a objetos [Guía del usuario del Lenguaje de Modelado Unificado, 1998]. Sin embargo, UML no puede describir de manera explícita las necesidades de movilidad requeridas para el modelado de sistemas de software basados en agentes móviles. En este trabajo, presentamos M-UML, una propuesta de ampliación al UML que cubre todos los aspectos de la movilidad en los diferentes puntos de vista y diagramas de UML.

El paradigma del agente móvil se hizo factible debido a los avances en el proceso de migración, evaluación remota, la computación de objetos distribuidos y la movilidad [8]. Los agentes móviles son objetos móviles o programas que llevan el código ejecutable y datos dentro de sí mismos. Tienen varias características que les ayudan a alcanzar sus objetivos o funciones de negocio tales como la negociación, y el pedido.

Los agentes inteligentes se ocupan de situaciones nuevas para la resolución de problemas automáticamente y así aprender de las experiencias pasadas. Presentan varias ventajas ya que son persistentes y en funcionamiento continuo. También reducen el tráfico en la red de manera significativa desde la computación, codificado en el agente móvil, es llevado a los datos en lugar de los datos a la computadora.

De hecho, hay beneficios muy claros y obvios de los agentes móviles, tales como la descarga de los enlaces de comunicación de red, la liberación de los recursos de computación locales, y el logro de la tolerancia a fallos mediante la migración a diferentes nodos de red. Sin embargo, para lograr la movilidad, hubo muchos problemas importantes, tales como, entre otras cosas, confiar en el software de agentes, confiar en el software de control remoto y el medio ambiente y el grado de autonomía que les damos.

En el comercio electrónico, el uso de herramientas de modelado para describir los procesos de negocio tiene un impacto decisivo en el éxito de la aplicación de negocios. Además, los agentes móviles son muy útiles para la realización de funciones de comercio electrónico [9,14] creando la necesidad de una herramienta de modelado para agentes móviles, como UML, crucial y altamente beneficioso.

Los estudios sobre el uso de UML en el modelado de los agentes móviles son raros y en extensión limitada. Klein [10] propuso una extensión de UML para los agentes móviles. Sin embargo, su enfoque se limita a una plataforma de agentes específicos y a la clase específica de aplicaciones de agentes móviles. Por otra parte, no proporcionan una descripción móvil de todos los puntos de vista y aspectos de los sistemas, por lo tanto, no cubre todos los diagramas UML. Bauer [11] introdujo Agente UML para especificar interacciones entre múltiples agentes.

En Agente UML, se desarrolló un nuevo tipo de diagrama llamado diagrama de protocolo de interacción. Sin embargo, esta extensión basada en agentes no incluía aspectos de la movilidad. También, Baumeister [12] introdujo una extensión del diagrama de actividad (ACD) para modelar sistemas móviles. Su extensión incluye la introducción de dos variantes de ACD para el modelado de la movilidad: una variante de responsabilidad centrada que se centra en el actor que realiza la acción, y una variante centrada en la ubicación que se centra en la topología de los lugares en los que las acciones están llevando a cabo.

En este trabajo, presentamos una extensión completa de UML 2.x estándar (M-UML) para hacer frente a todos los diagramas UML: Caso de uso, clase, objeto, diagramas de estado, de secuencia, de colaboración, de actividad, de componentes y de implementación.

En este trabajo, no describimos una metodología para el uso de nuestra extension propuesta, sin embargo, sólo proporciona un ejemplo sencillo con fines ilustrativos. El desarrollo de esta metodología es el objeto de nuestra investigación futura.

2.6.1 UML

En esta sección, proporcionamos alguna información de antecedentes sobre el Lenguaje de Modelado Unificado (UML 2.x)[5] normalizado por el Object Management Group (OMG), y agentes de software móviles y sus aplicaciones en diversos dominios.

2.6.1.1 ¿Qué es UML?

El UML es un software de lenguaje de modelado estándar para visualizar, especificar, construir y documentar los elementos de los sistemas en general y, en particular, los sistemas de software [7]. UML tiene una sintaxis y semántica bien definida.

Se proporciona un rico conjunto de objetos gráficos para ayudar a cumplir en la generación y perfeccionamiento de los sistemas de software orientado a objetos para el despliegue de captura de requisitos de componentes de software.

En UML, los sistemas pueden ser modelados considerando tres aspectos, los aspectos comportamentales, estructurales y arquitectónicos. Cada aspecto se basa tanto en los puntos de vista estático y dinámico del sistema. La visión estática representa una proyección sobre las estructuras estáticas de la descripción completa del sistema.

Sin embargo, la visión dinámica representa una proyección sobre el comportamiento dinámico del sistema. Por último, las vistas se comunican utilizando una serie de diagramas que contienen información haciendo hincapié en un aspecto particular del sistema. A continuación, se describen los aspectos diversos, opiniones y diagramas, y sus relaciones.

2.6.1.2 Aspectos

Tres aspectos fundamentales de los sistemas que se pueden reconocer son: los aspectos de comportamiento, estructurales y arquitectónicos. Los aspectos de comportamiento implican las funcionalidades proporcionadas por el sistema (aspectos estáticos de comportamiento) y sus interacciones asociadas, sus tiempos y sus estados (aspectos dinámicos de comportamiento).

Los aspectos estructurales implican las estructuras de clase del software orientado a objetos y sus interrelaciones (aspectos estructurales estáticas), y los detalles algorítmicos de los ciclos de objetos de la vida y sus interacciones (aspectos estructurales dinámicos). Por último, los aspectos arquitectónicos están relacionados con el despliegue y la distribución de software y hardware a través de los diversos componentes de sistemas distribuidos.

2.6.1.3 Puntos de vista

Los aspectos de los sistemas están cubiertos por diversos puntos de vista estáticos y/ó dinámicos, el caso de uso, diseño, proceso, aplicación y puntos de vista de implementación. La vista de casos de uso se centra en los aspectos del comportamiento de un sistema. En primer lugar, refleja las inquietudes y necesidades del usuario. Esta vista considera el sistema como una caja negra. Por lo general, los usuarios, los probadores y los requisitos de los analistas están interesados en esta vista.

Las vistas de diseño y procesos son la preocupación del equipo de desarrollo incluyendo los analistas, diseñadores e implementadores. Describen los aspectos estructurales estáticos y dinámicas del sistema.

La vista Diseño se ocupa de objetos, clases y colaboraciones, mientras que la vista de procesos se ocupa de temas de arquitectura de software, tales como problemas de concurrencia, multithreading, de sincronización, de rendimiento y escalabilidad. La vista de Implementación describe los archivos y componentes necesarios para montar y liberar el sistema. Finalmente, en la vista de implementación se describen las formas de distribuir, entregar e instalar los componentes y archivos del sistema a través de los nodos distribuidos del sistema.

2.6.1.4 Diagramas

Un diagrama contiene los elementos del modelo, tales como clases, objetos, nodos, componentes y relaciones, descritos por símbolos gráficos. Por otra parte, un diagrama se puede utilizar para describir ciertos aspectos del sistema en diferentes niveles de abstracción. Por ejemplo, un diagrama de estado puede describir las interacciones de entrada/salida a nivel sistema, y también puede ser utilizado para mostrar los cambios de estado de un objeto determinado del sistema a través de múltiples casos de uso.

Los nueve diagramas en UML se describen brevemente a continuación.

1. Diagrama de Caso de Uso (UCD): Este diagrama describe las funciones de un sistema y sus usuarios. Muestra una serie de actores externos y su relación con los casos de uso que representan los servicios prestados por el sistema. Los casos de uso pueden ser heredados por otro caso de uso, lo que permite la reutilización de los casos de uso. Además, los casos de uso se pueden asociar mediante la relación "uso", lo que permite la modularización de los casos de uso. De manera similar, los casos de uso se pueden asociar con la relación "extender" para hacer frente a ciertas condiciones o situaciones excepcionales o anormales. Este diagrama se utiliza para modelar los aspectos de comportamiento estático de la vista de casos de uso del sistema a modelar.

2. Diagrama de Clases (CLD): Este diagrama muestra la estructura estática de clases y sus posibles relaciones (es decir, asociación, herencia, agregación y dependencia) en el sistema. Este diagrama se utiliza para modelar los aspectos estructurales estáticos de los procesos de diseño y vistas del sistema a modelar.

3. Diagrama de objeto (OBD): Este diagrama es una variante de un CLD y utiliza la notación casi idéntica. Un sistema OBD puede ser un ejemplo de un CLD que muestra las posibles instancias de clases durante la ejecución del sistema. Este diagrama se utiliza para modelar los aspectos estructurales estáticos de los procesos de diseño y vistas del sistema a modelar.

4. Diagrama de Estados (STD): Este diagrama describe el posible comportamiento de un objeto utilizando los cambios de estado. También se puede utilizar como un diagrama a nivel del sistema que muestra los cambios de estado del sistema durante el funcionamiento del mismo. Por lo tanto, este diagrama se puede utilizar para modelar los aspectos dinámicos de comportamiento de la vista de casos de uso, y los aspectos estructurales dinámicos de los procesos de diseño y vistas del sistema para modelar.

5. Diagrama de Secuencia (SQD): Este diagrama muestra las interacciones entre el número de objetos y su tiempo pidiendo. En otras palabras, este diagrama describe un escenario en el que varios objetos interactúan. Este diagrama se puede usar para modelar los aspectos de comportamiento dinámicos de la vista de casos de uso si los objetos que interactúan son objetos de la interfaz o actores. Sin embargo, si los objetos que interactúan son internos, este diagrama se utilizara para modelar los aspectos estructurales dinámicos de la vista de diseño del sistema a modelar.

6. Diagrama de la colaboración (COD): Este diagrama muestra los objetos y los mensajes que intercambian. Los mensajes se numeran para proporcionar un orden de tiempo del mensaje. Similar a SQDs, un COD se puede utilizar para modelar tanto el aspecto del comportamiento dinámico de la vista de casos de uso, además del aspecto estructural dinámico de la vista del proceso del sistema a modelar. Vale la pena mencionar aquí que se puede producir fácilmente un SQD del COD y viceversa. Tanto la secuencia y CODs también se conocen como diagramas de interacción.

7. Diagrama de Actividad (ACD): Este diagrama muestra un flujo secuencial o concurrente de una actividad a otra. Un ACD consiste en estados de acción, estados de actividad y las transiciones entre ellos, y por lo general se extiende por más de un caso de uso (o funcionalidad). Este diagrama utiliza símbolos de redes de Petri, diagramas de flujo y máquinas de estados finitos. Al igual que en los diagramas de interacción, un ACD se puede utilizar para modelar tanto el aspecto del comportamiento dinámico de la vista de caso de uso, además del aspecto estructural dinámico de la vista de diseño del sistema a modelar.

8. Diagrama de Componentes (CPD): Este diagrama muestra la estructura física del código en términos de componente de código. Un componente puede ser

un código fuente, un binario, o un ejecutable. Un componente contiene información acerca de la clase o clases de lógica que implementa, creando así un mapeo de la vista de diseño a la vista de componentes. Un diagrama de componentes se utiliza para modelar los aspectos arquitectónicos estáticos de la vista de implementación del sistema a modelar.

9. Diagrama de Despliegue (DPD): Este diagrama muestra la arquitectura física y la distribución de los componentes de hardware y software en el sistema. Un DPD se utiliza para modelar los aspectos arquitectónicos estáticos de la vista de despliegue del sistema a modelar.

Aspectos/Vistas	Vista estatica	Vista dinamica
Comportamiento (funcional) aspectos: Vista caso de uso	Casos de Uso Diagramas	Diagrama de Secuencia Diagrama de Colaboración Diagrama de Estados Diagrama de Actividades
Estructural (diseño) Aspectos: Diseño y vista de Procesos	Diagrama de Clases Diagrama de Objetos	Diagrama de Secuencia Diagrama de Colaboración Diagrama de Estados Diagrama de Actividades
Aspecto Arquitectónico: Vista de Implementación, Vista de Despliegue	Diagrama de Componentes, Diagrama de Despliegue	

Tabla 2.1 Aspectos, vistas y diagramas de soporte en UML

La Tabla 2.1 resume los aspectos, puntos de vista y sus esquemas de apoyo en UML. [7,12].

CAPÍTULO III DESARROLLO DE UNA APLICACION MÓVIL

3.1 Mobile UML como ejemplo

En esta sección, se describen las extensiones realizadas en cada uno de los diagramas UML para permitir la representación explícita de los aspectos de la movilidad [13,15]. En primer lugar, introducimos de manera informal nuestro sistema móvil que queremos modelar. A continuación, presentamos las modificaciones en el UML tradicional.

3.2 M-UML

En esta sección, describimos M-UML pasando a través de cada uno de los diagramas UML y explicando las modificaciones y ampliaciones necesarias para describir los aspectos de movilidad del sistema ejemplo de navegacion a modelar;

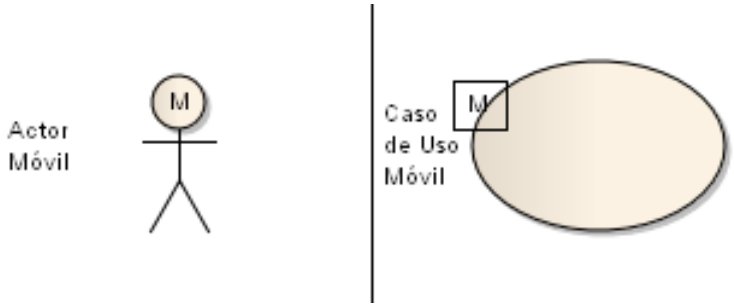
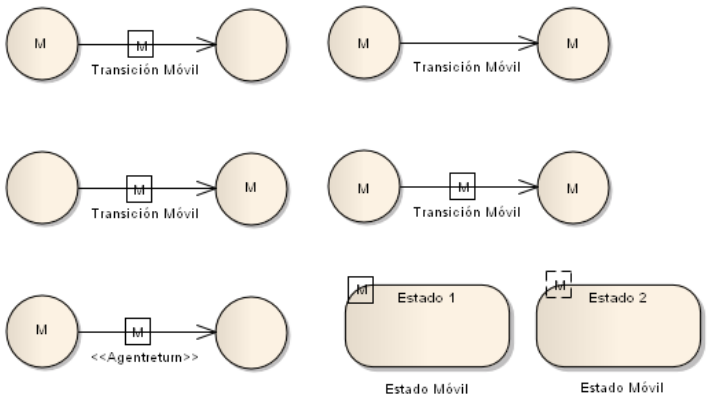
Diagrama de Casos de Uso (UCD)	
Diagrama de Estados (STD)	

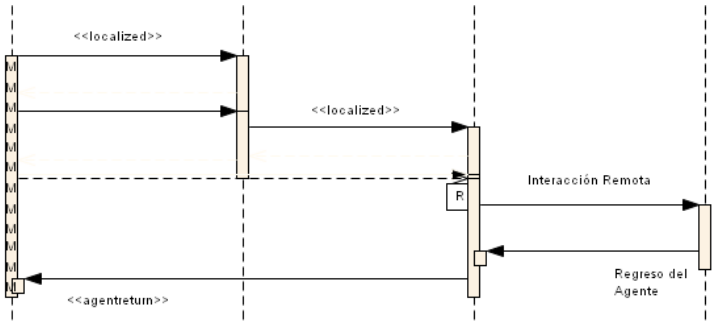
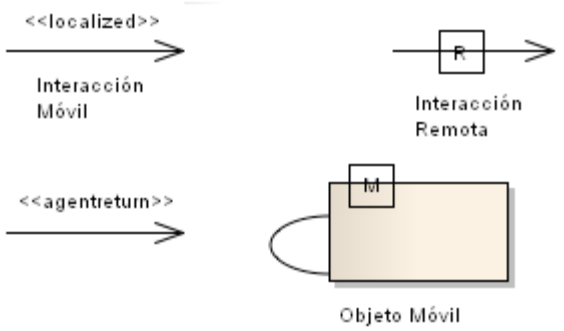
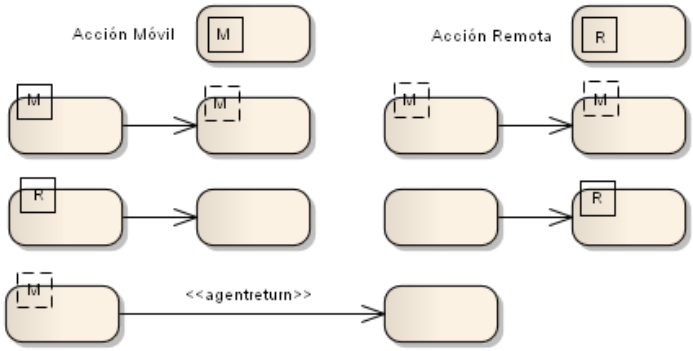
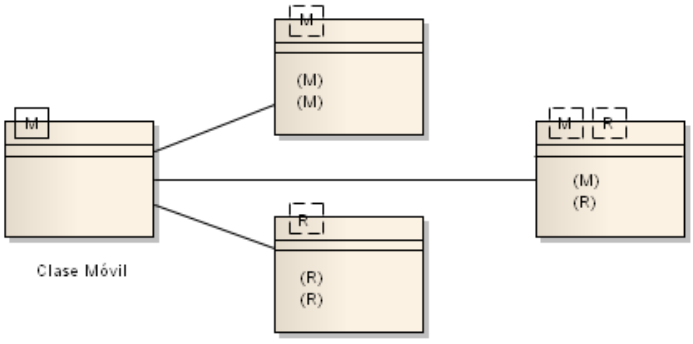
Diagrama de Secuencia (SQD)	
Diagrama de Colaboración (COD)	
Diagrama de Actividad (ACD)	
Diagrama de Clases (CLD) Diagrama de Objetos (OBD)	

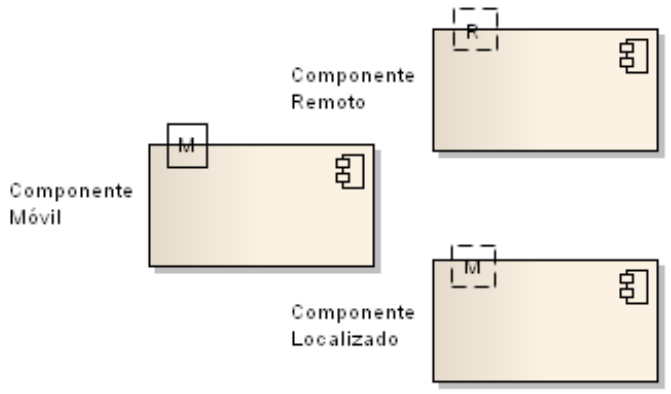
Diagrama de Componentes (COD)	
Diagrama de Despliegue (DPD)	

Tabla 3.1 Resumen de diagramas M-UML

La Tabla 3.1 resume las extensiones de movilidad propuestas en esta sección.

3.2.1 ¿Qué sucede después de que se reúnen los casos de uso?

Hoy en día, hay una variedad de herramientas UML y de desarrollo no basado en proceso UML como Extreme Ken Beckett programación o RUP de Rational Software. Todos los procesos de desarrollo siguen los mismos procesos y flujo de acuerdo al siguiente diagrama:

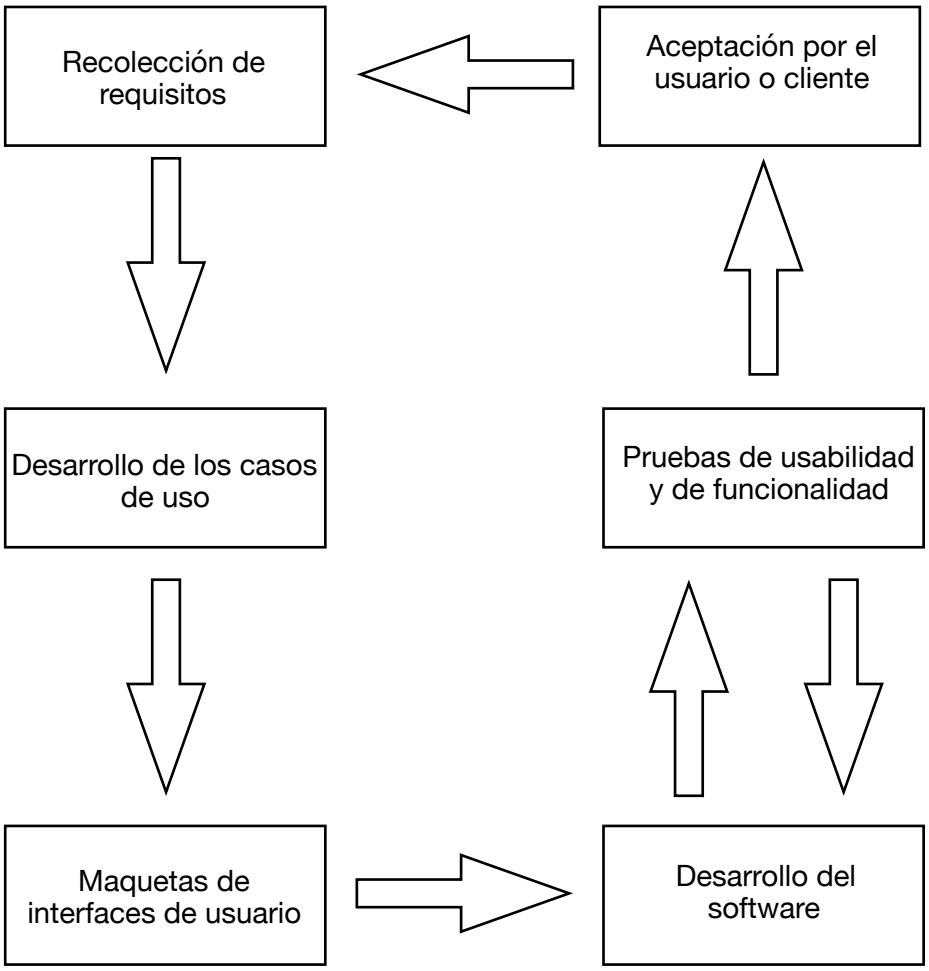


Figura 3.1 Ciclo de desarrollo de software para aplicaciones de escritorio

Por el contrario, en el siguiente diagrama se muestra este proceso de alto nivel, modificado para el desarrollo de aplicaciones móviles.

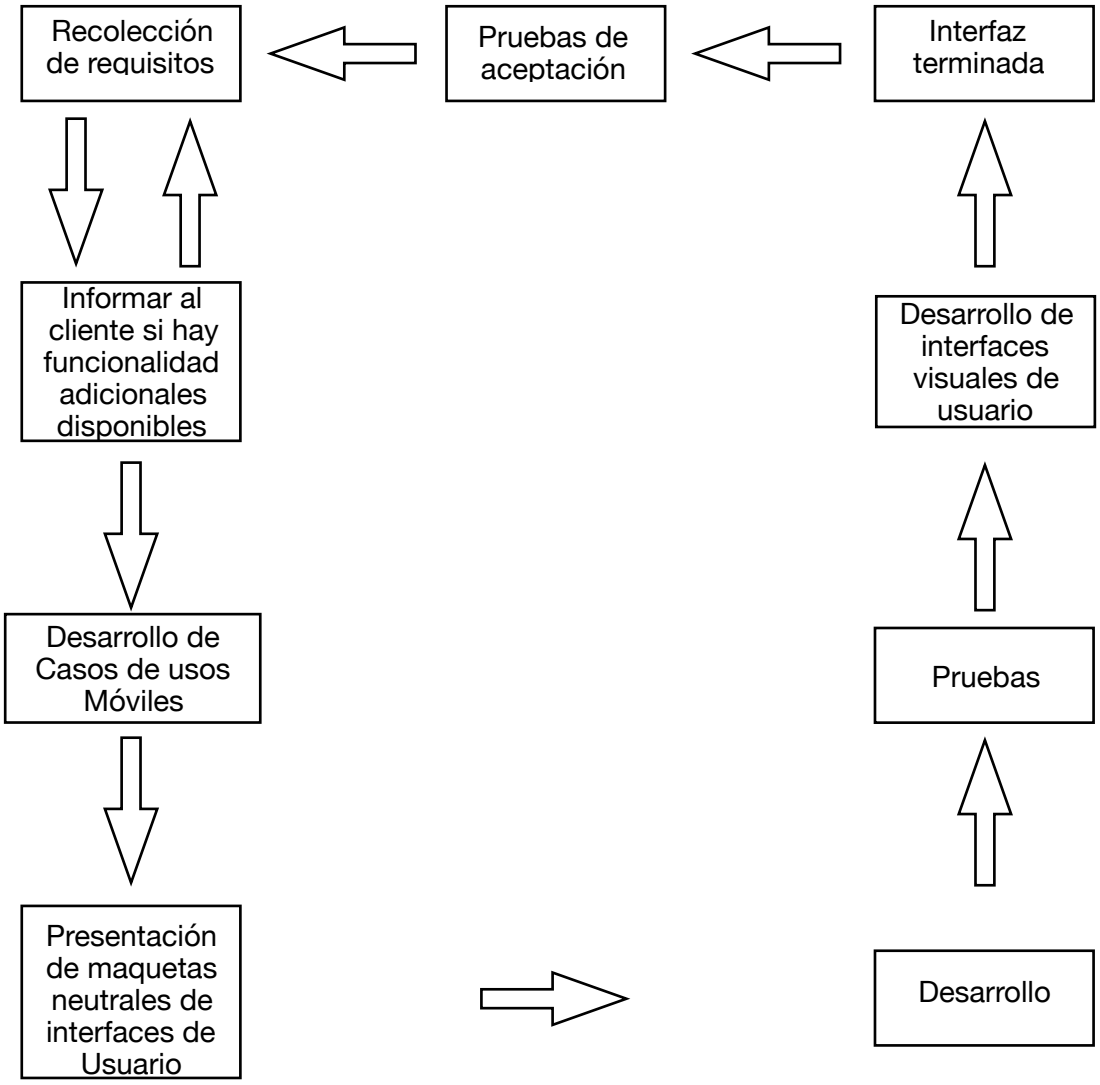


Figura 3.2 Ciclo de desarrollo de software para aplicaciones móviles

3.2.2 Diagrama de Casos de Uso

Un Diagrama de casos de uso (UCD por sus siglas en ingles) encapsula actores y casos de uso. Los actores son entidades externas que interactúan con el sistema a modelar mediante casos de uso. Un actor puede ser un agente de software que interactúa con el sistema. Un actor móvil es un agente de software móvil que puede interactuar con el sistema, mientras se encuentre lejos de donde fue creado (es decir, la plataforma de base). La funcionalidad de un sistema o subsistema a modelar es descrito por un grupo de casos de uso. Un caso de uso representa un requisito funcional que puede conllevar uno o más actores que interactúan. Un caso de uso móvil representa un requisito funcional que consiste en uno o más actores móviles. Por lo tanto, al menos un agente móvil está implicado en un caso de uso móvil. Un sistema basado en agentes móviles contiene casos móviles y casos no móviles.

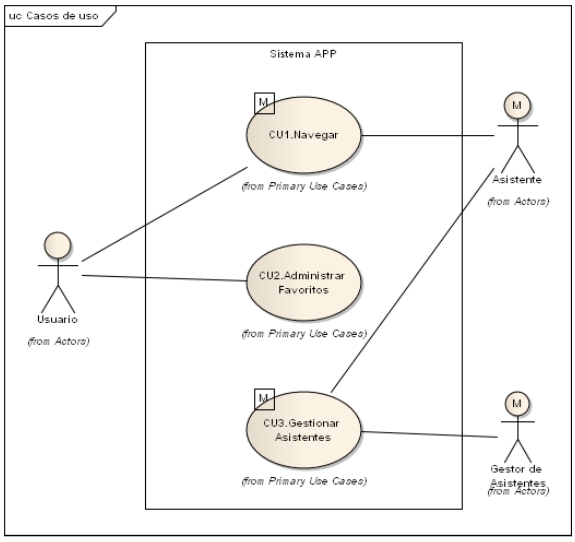


Figura 3.3 Diagrama de casos de uso

La figura 3.3 muestra a un UCD que contiene dos actores móviles y tres casos de uso, dos móviles y uno normal. En esta figura, tenemos un actor móvil *Asistente* y otro actor movil *Gestor de Asistentes,* un actor no movil *Usuario.* Nuestros dos actores móviles interactuan a través del caso de uso móvil *Gestionar Asistentes*. Del mismo modo, el Asistente está interactuando con el Usuario a través del caso de uso móvil *Navegar*.

3.2.3 Diagrama de Estados

Un Diagrama de estados (STD por sus siglas en ingles) es un diagrama UML que consiste en estados y transiciones que unen pares de estados. Normalmente, un STD describe el comportamiento de un actor en términos de cambio de estado. Un actor u objeto puede estar en un estado particular en un momento dado.

Un estado móvil es un estado alcanzable, mientras que el actor o el objeto está lejos de su plataforma base. Un estado móvil lleva una caja (M). Un enlace unidireccional que emana de un estado, el estado origen, y llega a otro estado, el estado final, a esto se llama transición. Una transición móvil es una transición entre dos estados que son accesibles por un agente, en dos plataformas diferentes. Por lo tanto, podemos tener una transición no móvil que une dos estados móviles, una transición móvil que une un estado móvil o vice versa, y una transición movil entre dos estados móviles. Las transiciones móviles llevan una caja (M). Sin embargo, si un agente puede llegar a un estado mientras que este ya sea en su base o a distancia, ese estado tendrá un cuadro de líneas discontinuas (M). Del mismo modo, si un agente móvil alcanza un estado al interactuar remotamente con otro agente, la transición llevara una caja (R). Por último, una transición se etiqueta con el estereotipo <<agentreturn>> si corresponde a la vuelta del agente a su base, durante el cambio de estado.

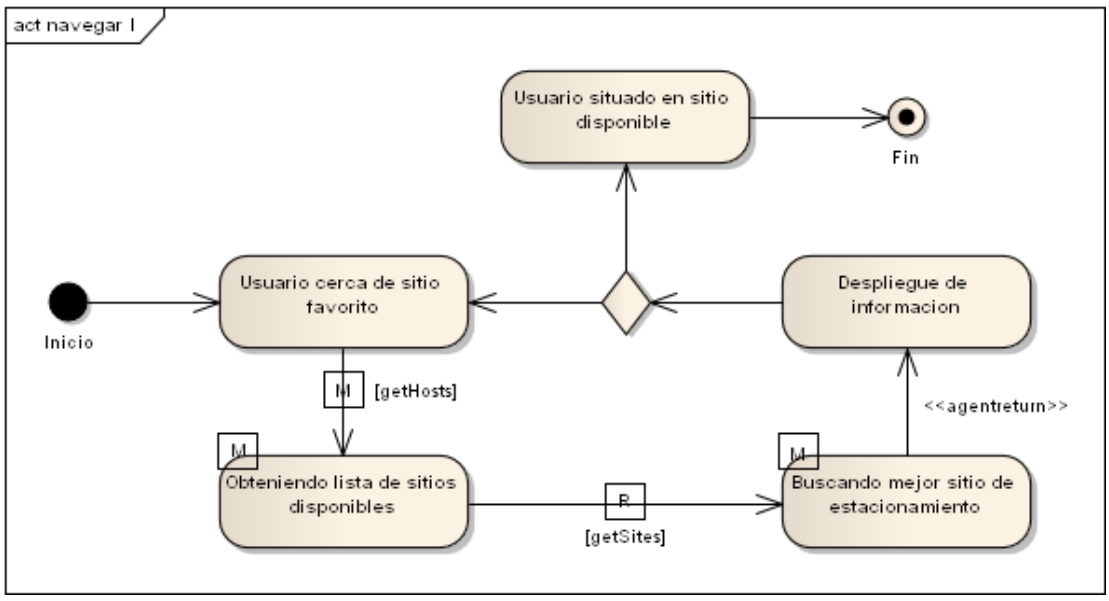


Figura 3.4 Diagrama de estados Navegar

La figura 3.4 muestra un STD que contiene ambos tipos de estados y transiciones en diferentes combinaciones. La figura muestra una parte del comportamiento de los estados del agente-móvil/actor del Asistente. En su plataforma base, el agente recibe el mensaje getHost y se mueve desde el estado Usuario cerca de sitio favorito al estado Obteniendo lista de sitios disponibles, en el que el agente ya está en una plataforma diferente. En este estado, el agente envía un mensaje remoto al gestor de asistentes y se mueve al estado Buscando mejor sitio de estacionamiento. Ambos son estados móviles.

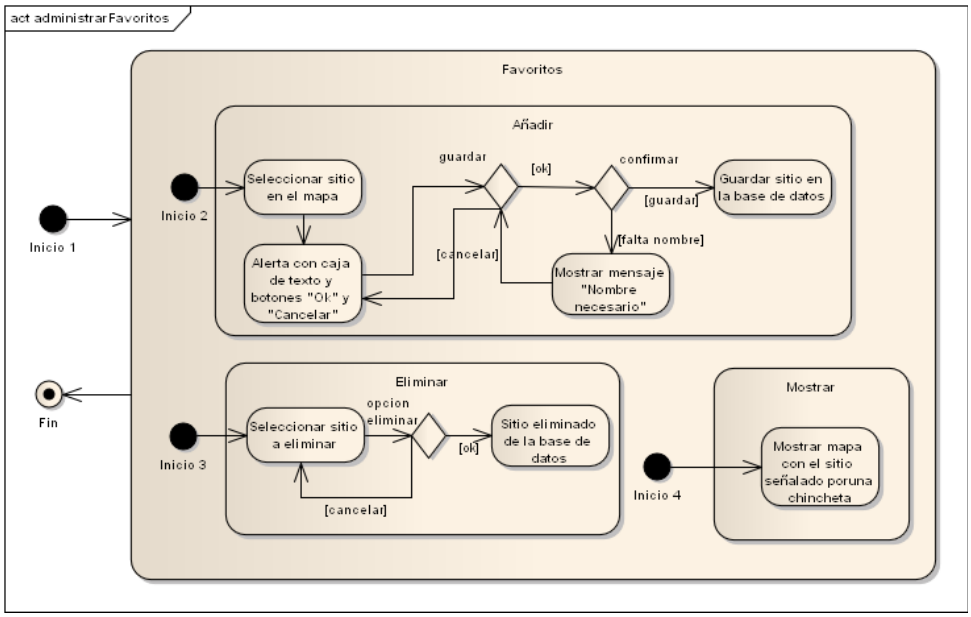


Figura 3.5 Diagrama de estados Administrar Favoritos

La figura 3.5 muestra un STD que contiene solo tipos de estados normales, ya que las tareas y procedimientos realizados son de carácter equivalente a cualquier otro tipo de software, se hacen operaciones administrativas de información necesaria para el correcto funcionamiento de la aplicación.

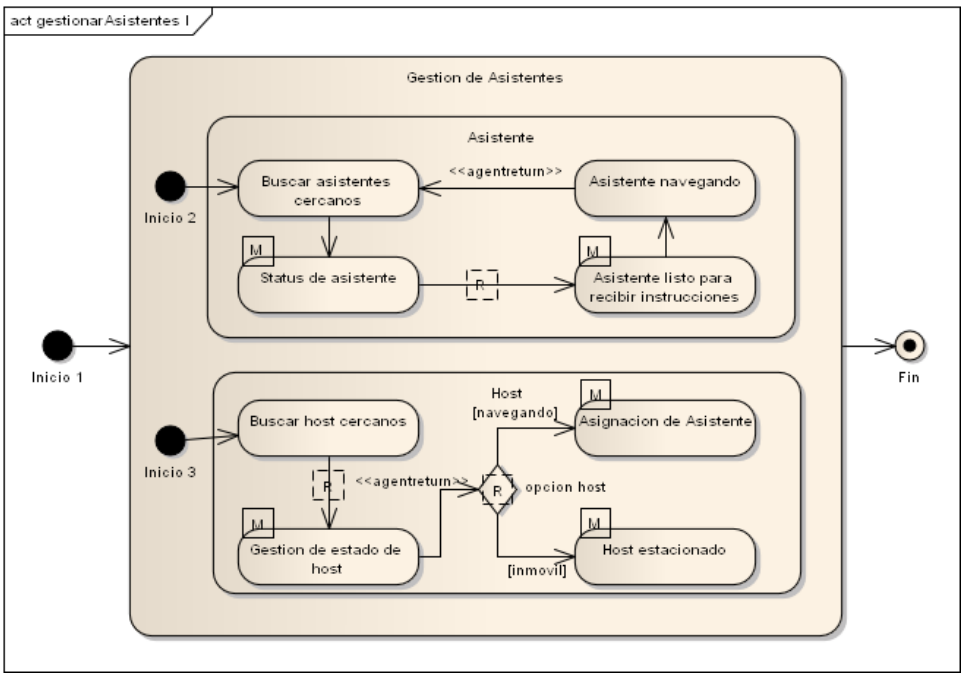


Figura 3.6 Diagrama de estados Gestionar Asistentes

La figura 3.6 muestra un STD que contiene todo tipos de estados y transiciones en diferentes combinaciones. La figura muestra una parte del comportamiento de los estados del agente-móvil/actor del Gestor de asistentes. En su plataforma base, el agente recibe el mensaje Buscar asistentes cercanos ó Buscar host cercanos y se mueve al estado Status de asistente o gestión de estado de host, en el que el agente ya está en una plataforma diferente.

3.2.4 Diagrama de Secuencia

Un Diagrama de secuencia (SQD por sus siglas en ingles) es un tipo de diagramas de interacción que muestra las interacciones entre los agentes, mientras avanza el tiempo. El tiempo se representa por una línea de tiempo vertical para cada agente. Una interacción se representa mediante un flecha desde el agente iniciador en dirección al agente receptor.

Si el agente iniciador está interactuando mientras no sea su plataforma base, la línea de tiempo desde el momento en que esta interactuando se produce a la vez que el agente vuelve a su plataforma, estará lleno de "M". Una interacción que emana de la línea de tiempo del agente se llama interacción móvil. Una interacción móvil que implica dos agentes situados en la misma plataforma se etiqueta con el estereotipo <<localized>>.

Sin embargo, una interacción móvil que implica un agente móvil y los dos agentes que interactúan no se encuentra en la misma plataforma llevará una caja (R) en la

intersección entre la línea de tiempo de la plataforma del agente iniciador y la interacción. Una línea discontinua en el SQD se utiliza para rastrear la plataforma de código de un agente móvil o bien iniciar una interacción localizada o remota. Esto significa que las interacciones móviles no tienen por qué ser colocados juntas (es decir, marcado con <<localized>>). Por ejemplo, un agente móvil se puede mover a una plataforma remota y comienza a interactuar con otros agentes remotos. Por último, el regreso del agente a su base está representado por una flecha dirigida con estereotipo <<agentreturn>>.

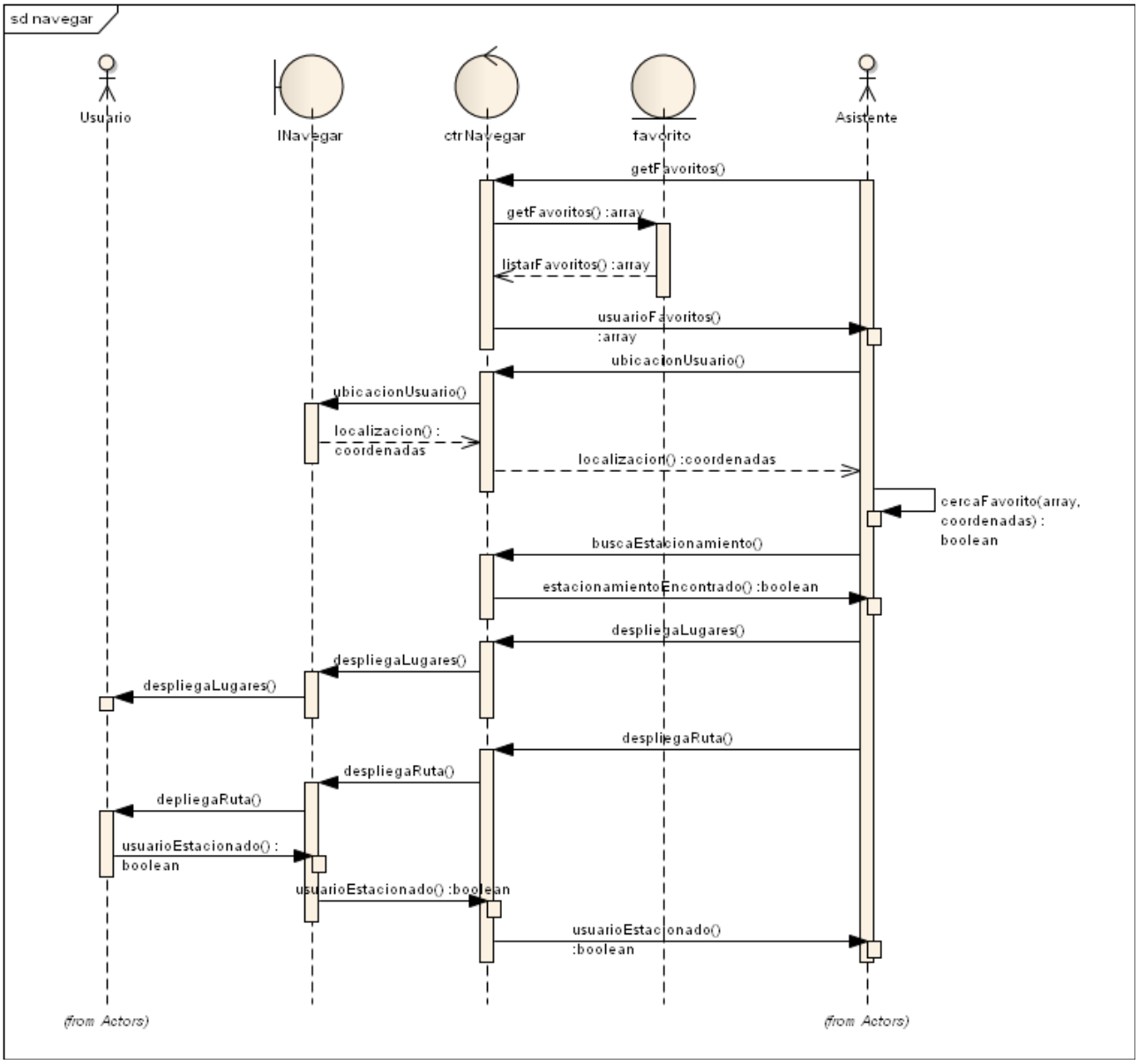


Figura 3.7 Diagrama de secuencia Navegar

La figura 3.7 muestra un SQD en la cual el usuario interactua de forma transparente con el asistente, el asistente que es el agente móvil, se encarga de realizar las tareas que el usuario va necesitando. El asistente al detectar que el usuario se encuentra en determinada zona se encarga de comunicarse con los host adyacentes al igual que con otros asistentes.

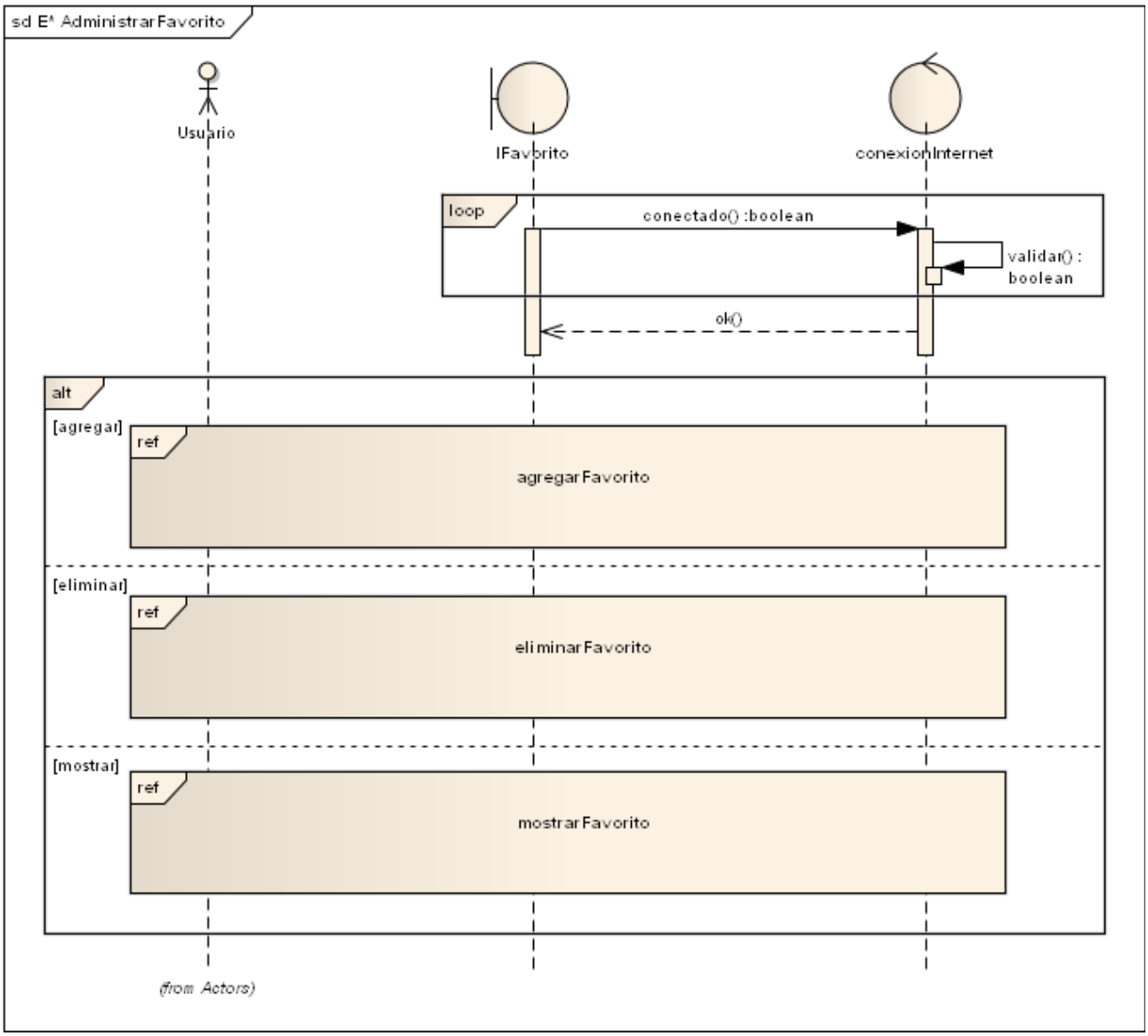


Figura 3.8 Diagrama de secuencia Administrar Favorito

La figura 3.8 muestra un SQD global el cual como podemos observar se va detallando en las figuras 3.9, 3.10 y 3.11. El usuario interactua con la aplicación agregando, eliminando o consultando zonas geográficas de su interés, es el único caso de uso en el cual el usuario puede interactuar con la aplicación de forma física.

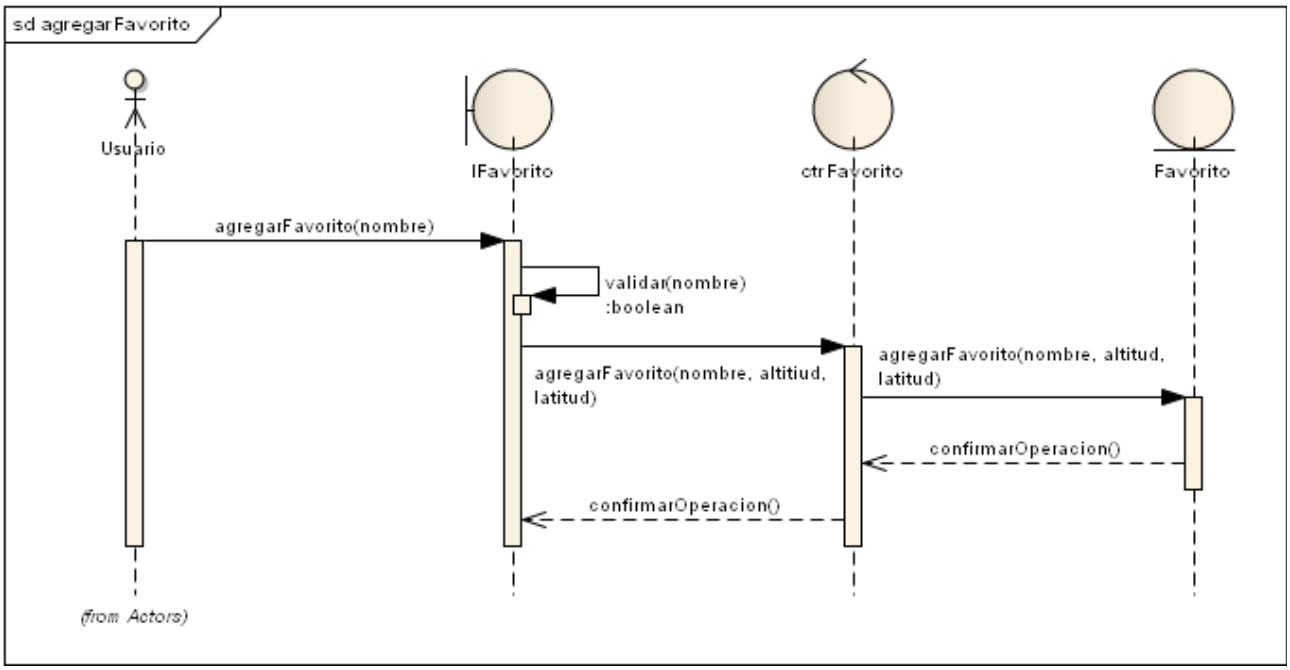


Figura 3.9 Diagrama de secuencia Administrar Favorito, Agregar Favorito

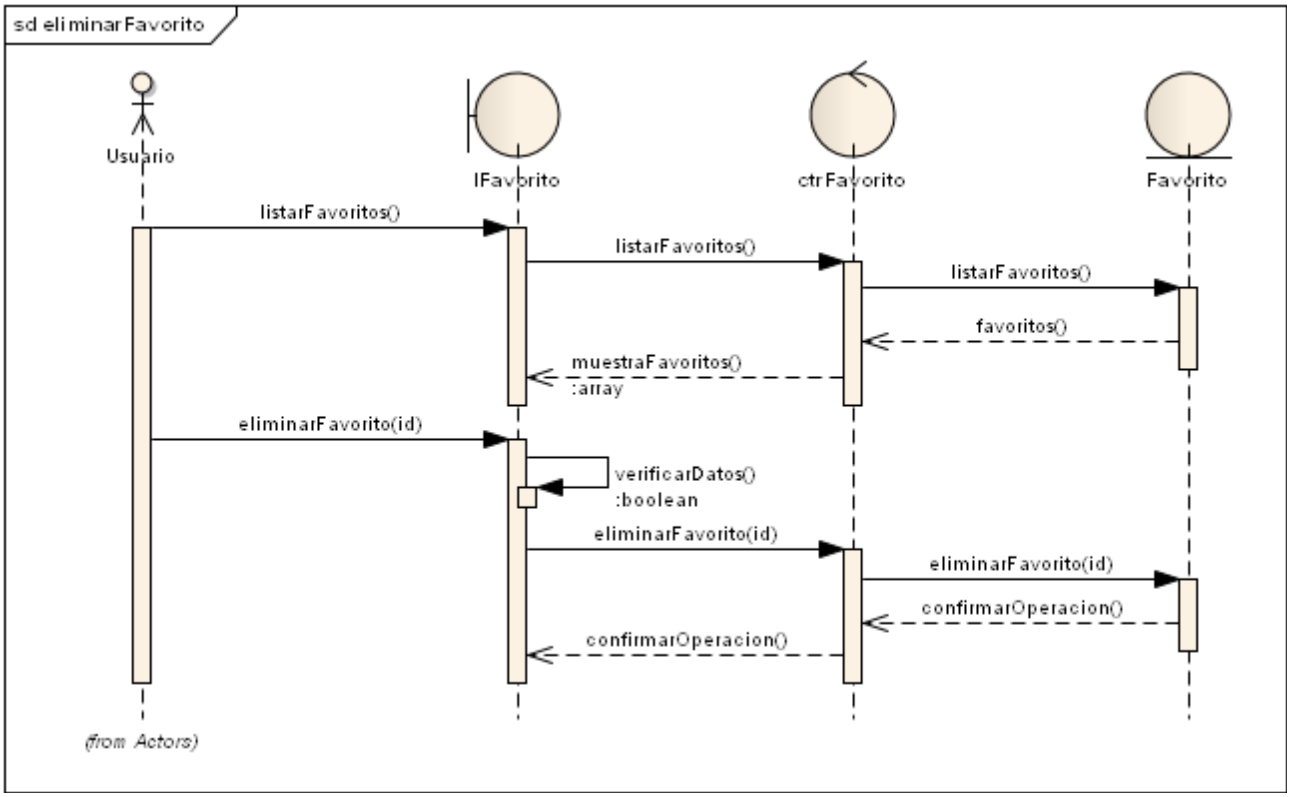


Figura 3.10 Diagrama de secuencia Administrar Favorito, Eliminar Favorito

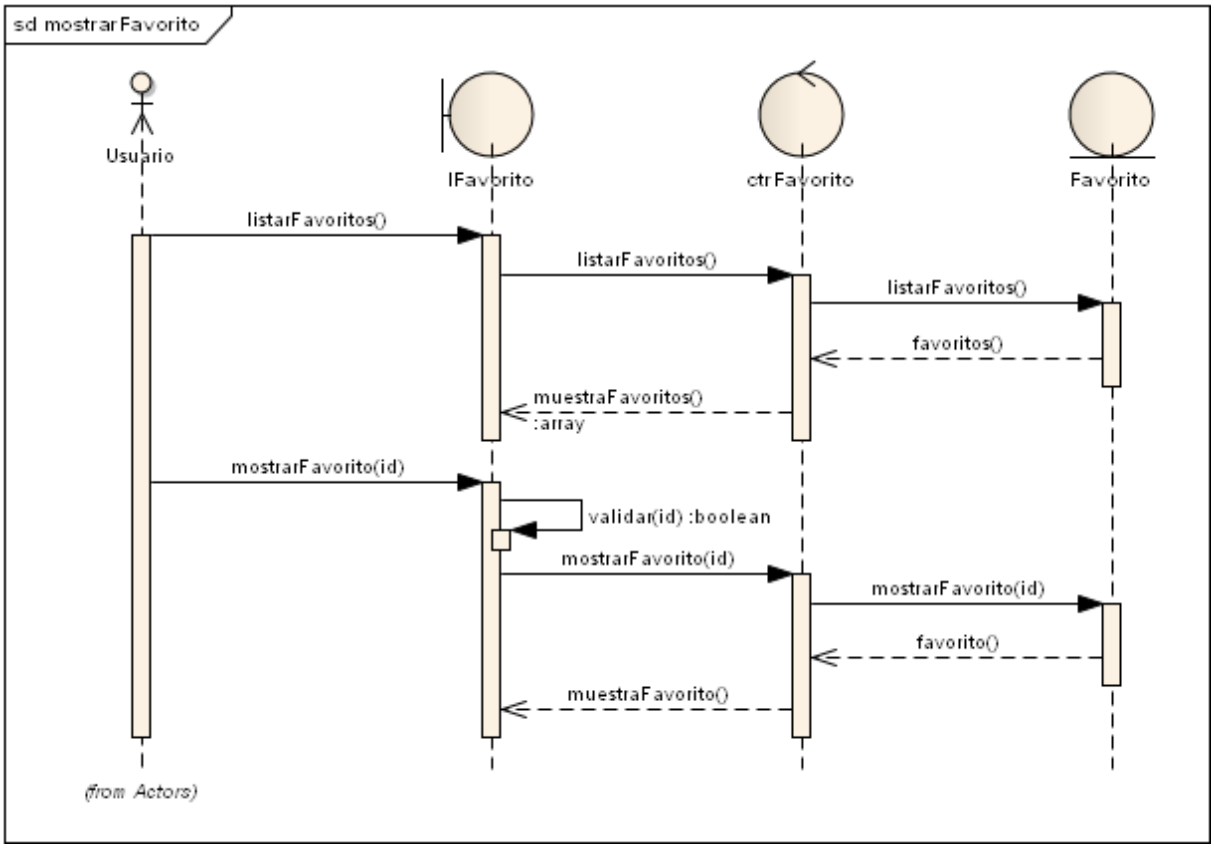


Figura 3.11 Diagrama de secuencia Administrar Favorito, Mostrar Favorito

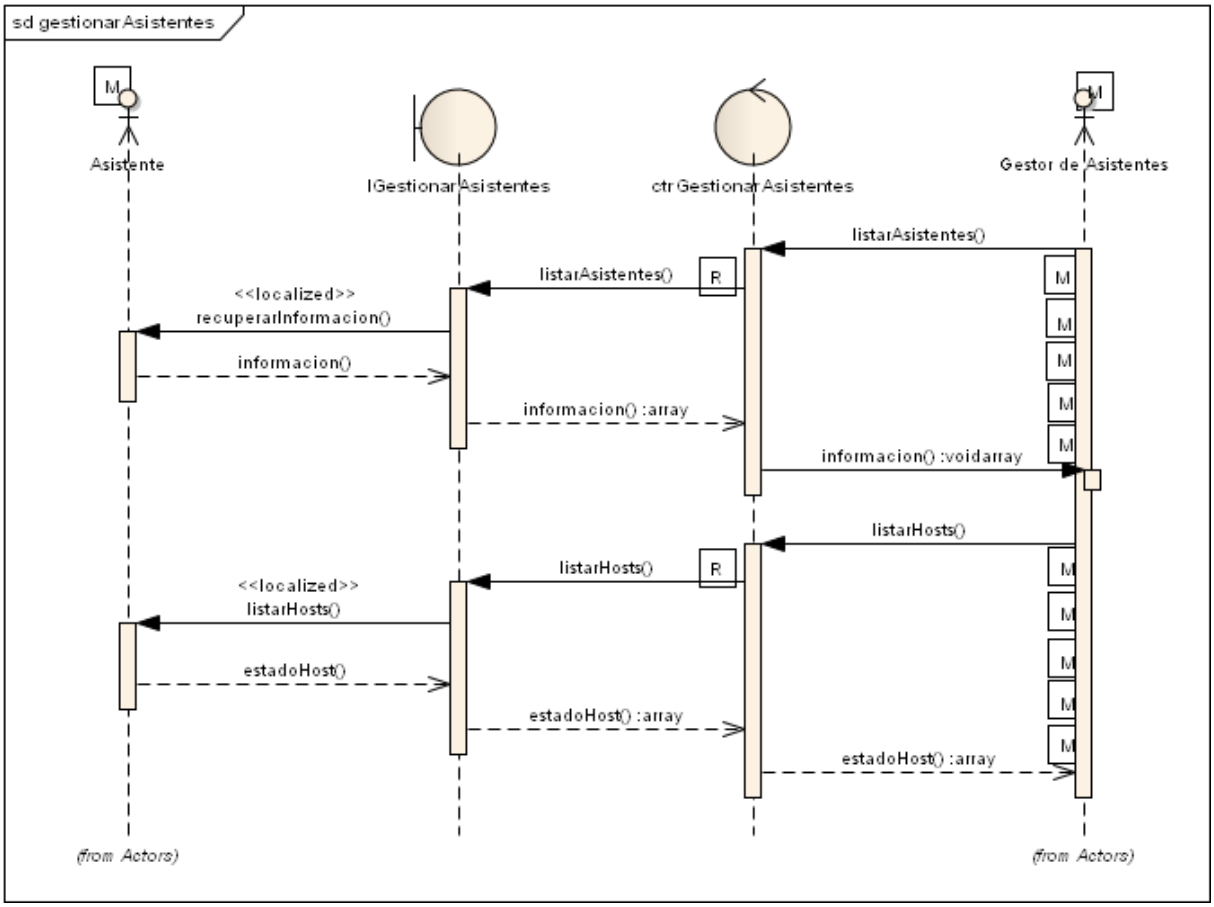


Figura 3.12 Diagrama de secuencia Gestionar Asistente

La figura 3.12 muestra un SQD móvil y la participación de ambos <<localized>> y las interacciones remotas. El gestor de asistentes envía un mensaje listar asistentes y listar hosts al asistente. A continuación, el asistente se desplaza desde su plataforma a una plataforma asignada por el gestor de asistentes en su lista de hosts. El gestor móvil envía un mensaje local listar host y se mueve al siguiente host después de recibir los parámetros. Finalmente, después de conseguir el listado desde el último asistente, vuelve a su plataforma base.

3.2.5 Diagrama de Colaboración

Un Diagrama de colaboración (COD por sus siglas en ingles) es un tipo de diagrama de interacción que muestra el comportamiento de objetos que interactúan usando una secuencia de intercambio de mensajes. La relación estática entre los objetos está bien definida.

Por lo tanto, un COD muestra las estructuras y las propiedades comportamentales de las interacciones entre objetos. Un COD contiene objetos (o instancias de clases) y líneas pares conectando a objetos o agentes. Cada interacción es etiquetado por el número de secuencia de mensaje, el identificador de mensaje, y una flecha que indica la dirección del mensaje. Funcionalmente, tanto la colaboración como SQDs son equivalentes.

En consecuencia, dado un SQD, podemos decir que es el equivalente de un COD y viceversa. Si un objeto móvil participa en la interacción, se agregara una caja (M) al objeto. Si la interacción se produce en la misma plataforma, la interacción línea llevará el estereotipo <<localized>>. Sin embargo, si la interacción implica dos agentes remotos, mientras que uno o ambos están lejos de su plataforma base, la línea de interacción llevará la caja (R).

Una interacción self-loop lleva el estereotipo <<agentreturn>>, en un objeto móvil representa la devolución del objeto móvil a su plataforma originaria, en particular orden con el numero de secuencia unido a la iteración.

3.2.6 Diagrama de Actividad

Un Diagrama de actividad (ACD por sus siglas en ingles) muestra el flujo de control entre las acciones o acciones complejas llamadas actividades. Se trata básicamente de una mezcla de máquinas de estado finitas/gráficos de estado, la sintaxis de Redes de Petri y la semántica. Sin embargo, a diferencia de un grafo

de estados, los estados que representan las actividades en un ACD normalmente pueden abarcar más de un objeto/agente.

Un ACD es capaz de modelar tanto el flujo secuencial y paralelo entre las acciones y actividades que intervienen en el modelado de un caso de uso. Por otra parte, una acción es un paso computacional atómico básico, a diferencia de una actividad que se puede describir por otro ACD por lo tanto, creando una jerarquía de ACD.

Una acción o actividad móvil implica al menos un agente móvil en ejecución en una plataforma diferente de su plataforma de origen. Un caso de uso móvil consiste en una o más acciones o actividades móviles. Si la acción/actividad se ejecuta por un agente fuera de su plataforma de origen, entonces, esta acción/ actividad móvil está representado por un óvalo con un cuadro (M) unido a su esquina superior izquierda.

Además, si se desencadena la acción (solicitada) por un agente móvil, esta acción llevará a un cuadro de líneas discontinuas (M). Sin embargo, esta acción/actividad móvil interactúa con un agente remoto, llevará una caja (R) en su lugar. El retorno de un agente para su plataforma de origen está representado por una acción móvil estereotipada con <<agentreturn>>.

Por último, para especificar explícitamente la ubicación en la que se está produciendo una acción móvil o remoto realizada por un agente móvil, la acción es estereotipada con <<location = agent>>. En resumen, el diagrama mostrará, además de la secuencia de las acciones/actividades, la colocación relativa de los agentes en el desempeño de esas acciones/actividades. Esta extensión de la ACD combina tanto la ubicación centrada y las extensiones de la responsabilidad centrada propuestas en la Ref [16].

3.2.7 Diagrama de Clase

Un Diagrama de clase (CLD por sus siglas en ingles) muestra la estructura estática de las clases de software del sistema y describe todas las relaciones entre las clases, incluyendo las relaciones de asociación, agregación y generalización. Una clase móvil es una clase de la cual los objetos/agentes móviles son creados. Una clase que hereda de una clase móvil hereda su característica de movilidad.

Las clases que forman parte de otra clase móvil (en relación de agregación) no son necesariamente clases móviles. Del mismo modo, clases asociadas con una clase móvil no son necesariamente clases móviles. Clases de las que se crean

instancias de objetos móviles se muestran con una caja (M) en la parte superior izquierda. Sin embargo, otras clases que contienen un comportamiento (métodos) afectado (comunicándose con) por objetos móviles se muestran con un cuadro de líneas discontinuas (M).

Por otra parte, un comportamiento de la clase que es afectado por un agente móvil a distancia se muestra con un recuadro de trazos (R). Por lo tanto, los métodos invocados por un agente móvil se etiquetan con (M) o (R), dependiendo de la localización del agente móvil que llama. Si una clase contiene métodos de ambos tipos, llevará ambas cajas discontinuas (M) y (R).

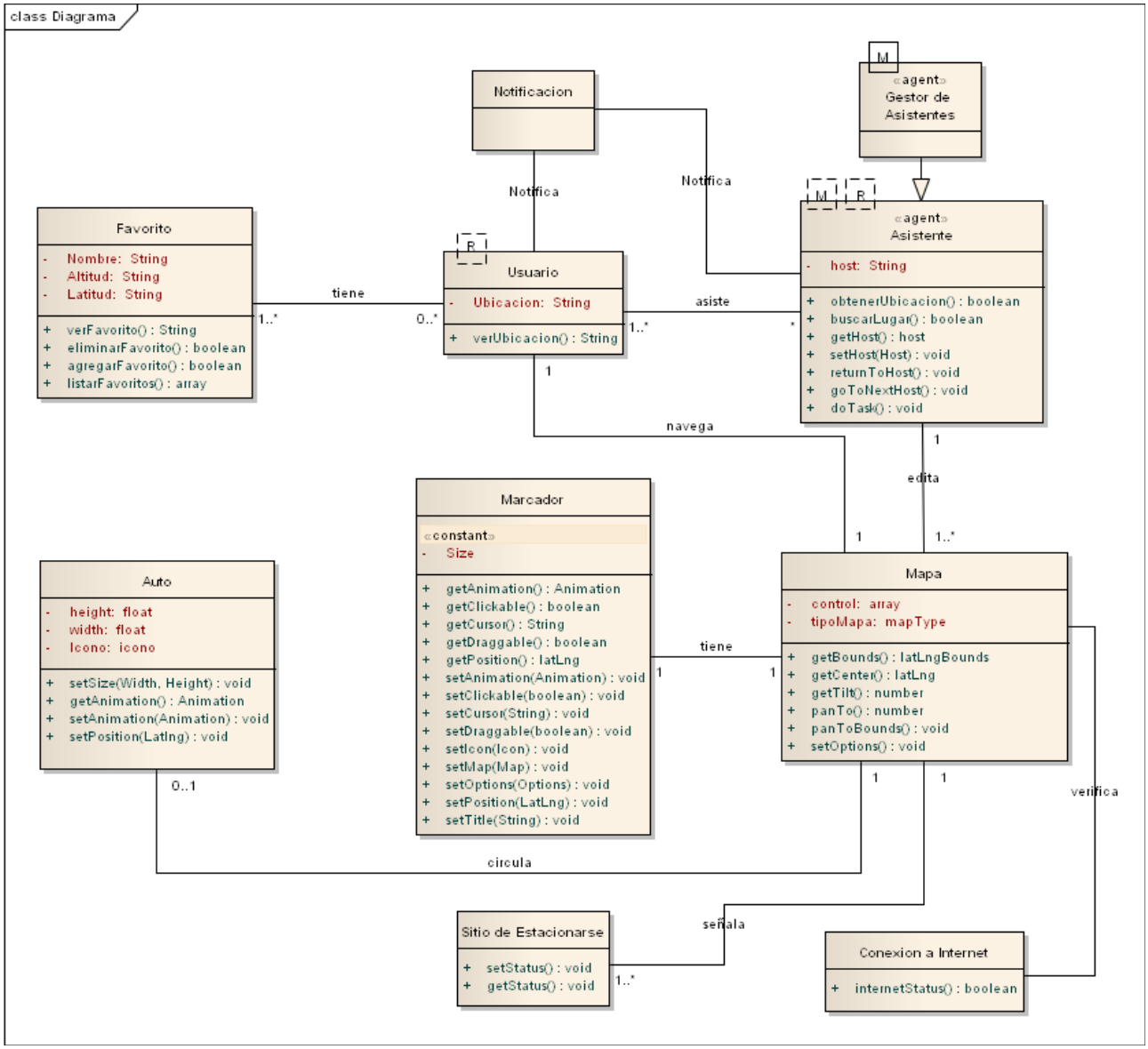


Figura 3.13 Diagrama de clases

La figura 3.13 muestra un CLD que implica clases móviles y no móviles. En esta figura, la clase asistente incluye los métodos obtenerUbicacion(), buscarLugar(), setHost() entre otros que son invocados por el agente móvil de clase agent,. Este es también el mismo para el gestor de asistentes, cuyas instancias no son móviles. La jerarquía de clases muestra las relaciones de asociación entre el asistente, gestor y los objetos de la aplicacion.

Finalmente, la clase usuario se asocia con la clase asistente y lleva una caja con lineas punteadas (R), ya que contiene el método buscarLugar() invocado de forma remota por un objeto móvil de clase asistente. Esto ocurre de forma predeterminada cuando se muestra la jerarquía de clases completo, que muestra las clases no móviles asociados a las clases móviles.

3.2.8 Diagrama de Objectos

Un Diagrama de objetos (OBD por sus siglas en ingles) muestra los objetos y las relaciones entre ellas. Un sistema OBD se deriva de un CLD y se basa en la estructura estática del modelo, pero también muestra varias instancias de algunos objetos como ilustraciones o escenarios de las relaciones de asociación de objetos hechas de una clase móvil que es de objetos móviles. Esos objetos móviles son considerados los agentes móviles en el sistema.

3.2.9 Diagrama de Componentes

Un Diagrama de componentes (CPD por sus siglas en ingles) contiene componentes, interfaces y varios tipos de relaciones, como las relaciones de dependencia, la generalización y asociación. Los componentes pueden ser código ejecutable, páginas web, documentación y componentes de una libreria.

Un componente ejecutable o un paquete que contiene el código del agente móvil lleva una caja (M), y un componente ejecutable que contiene un agente que interactúa con un agente móvil en la misma plataforma, y no es en sí móvil, lleva un cuadro de líneas discontinuas (M). Sin embargo, un componente ejecutable que contiene un agente que interactúa de forma remota con un agente móvil, y no es en sí móvil, lleva una caja de trazos (R).

Del mismo modo, cualquier otro componente relacionado con los agentes móviles en el sistema llevará la caja (M). Finalmente, similar a una clase, un componente puede llevar más de un tipo de cajas.

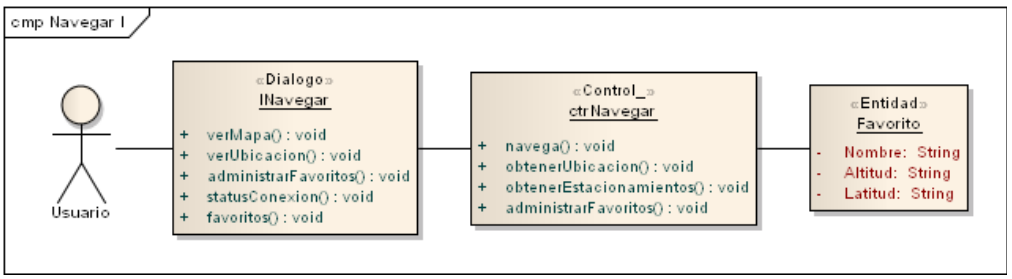


Figura 3.14 Diagrama de componente Navegar I

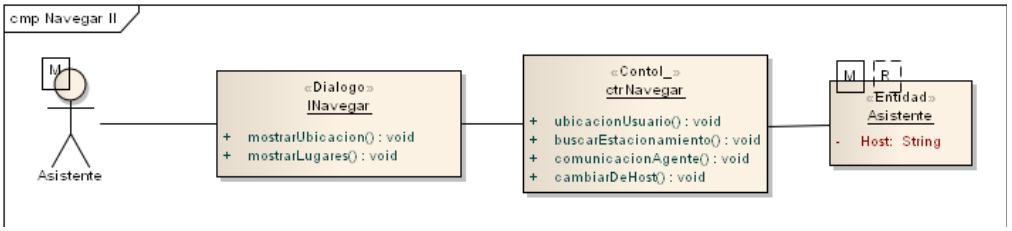


Figura 3.15 Diagrama de componente Navegar II

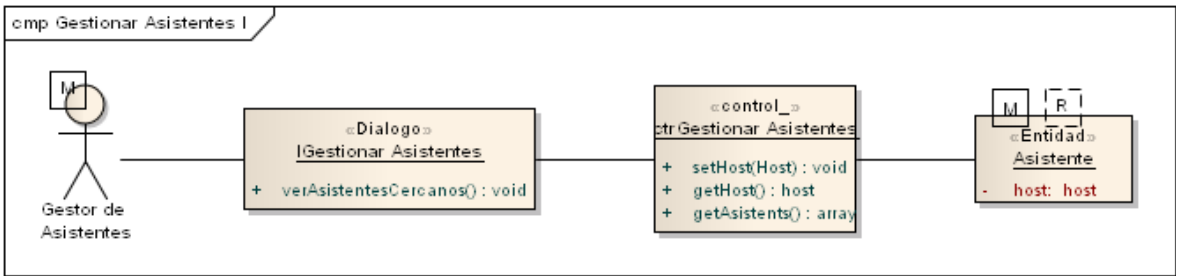


Figura 3.16 Diagrama de componente Gestionar Asistente I

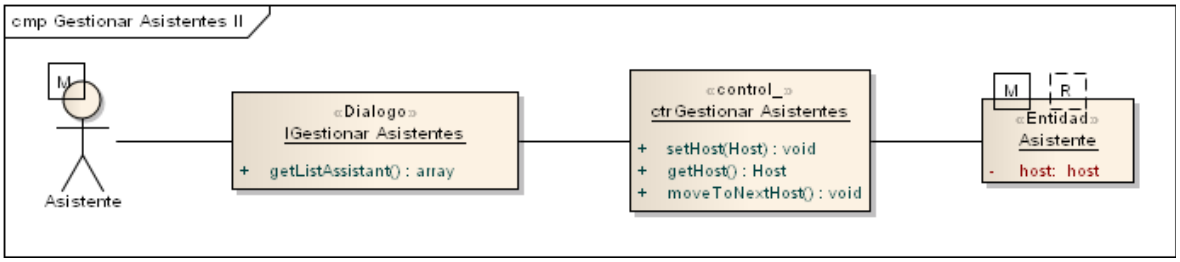


Figura 3.17 Diagrama de componente Gestionar Asistente II

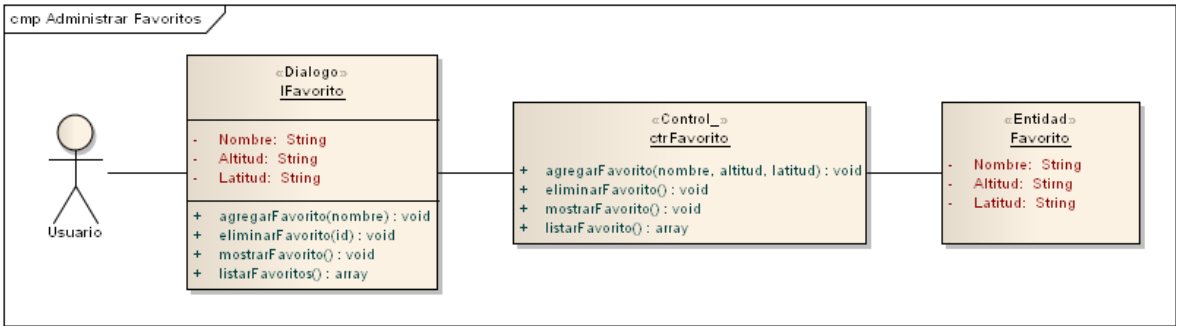


Figura 3.18 Diagrama de componente Administrar Favoritos

3.2.10 Diagrama de Despliegue

Un Diagrama de despliegue (DPD por sus siglas en ingles) es útil para modelar sistemas distribuidos y cliente/servidor. El diagrama muestra la distribución de las unidades físicas de procesamiento o nodos y componentes que residen en ellos. Un sistema móvil incluye una o más plataformas o nodos en los cuales se crean y expiden los agentes móviles. Dichos nodos tendrán una caja (M) en su esquina superior izquierda. Un nodo que recibe en un agente móvil llevará un cuadro de líneas discontinuas (M) en su esquina superior izquierda.

Por último, un nodo que interactúa de forma remota con un agente móvil llevará el cuadro de líneas discontinuas (R) en su esquina superior izquierda. Un nodo puede realizar cualquier combinación de cajas, por ejemplo, un nodo puede haber enviado un agente móvil (M), y todavía puede recibir otros agentes móviles (línea de punteada (m)), o interactuar con un agente móvil en otro nodo (línea de punteada (R)).

3.3 Diseño de la interfaz

Con el análisis de los diagramas anteriores se puede tener una primera aproximación de la interfaz final del usuario, se puede destacar que en los diagramas se hace referencia a una interfaz donde se elige una opción ya sea para dar de alta a un sitio nuevo, modificar el sitio seleccionado, eliminarlo y obtener el sitio de mejor posicionamiento para el usuario; en cada una de estas opciones notamos que todas regresan a la interfaz de usuario inicial.

Por lo que como interfaz de usuario principal destacamos la siguiente:

La interfaz principal del usuario al iniciar la aplicación, como se muestra en la figura 3.19 notamos que al abrir la aplicación se mostrara la vista donde siguiendo la navegación con los diagramas y los puntos a considerar para el desarrollo de interfaces nos dan como resultado la siguiente interfaz que cuenta con la geolocalizacion en tiempo real del usuario final, y en la parte superior derecha un botón con forma de estrella para poder administrar sus sitios favoritos.

Ahora como podemos observar en la figura 3.20 vemos la interfaz que se pretende mostrar en la opción cuando el usuario elija dar de alta a un nuevo usuario en nuestra base de datos. En esta vista el usuario gestionara la información pertinente a sus favoritos, con una interfaz entendible al usuario siguiendo los lineamientos para el diseño de aplicaciones en móviles.

Figura 3.19 Interfaz de usuario principal

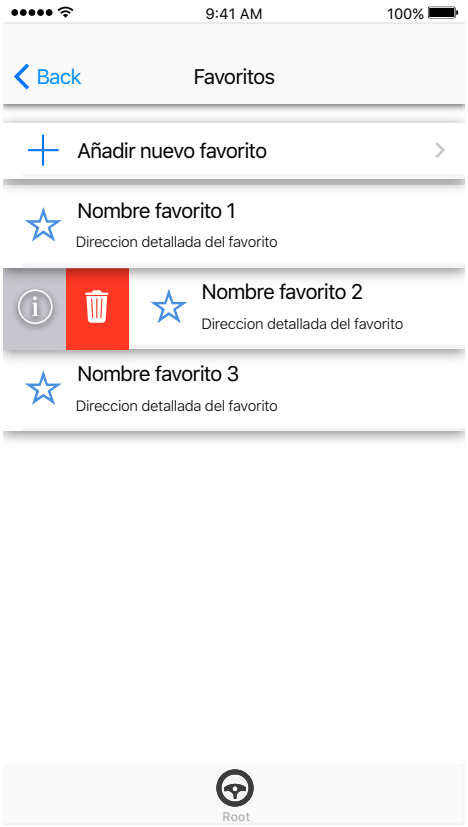


Figura 3.20 Interfaz de usuario agregar, eliminar, consultar

Como podemos observar, esta vista engloba las funciones que tiene permitido el usuario realizar para la gestión de sus sitios favoritos, los lineamientos para el correcto funcionamiento de aplicaciones móviles son proporcionados como apoyo por parte de Apple.

Estudios correspondientes a guías de desarrollo en el área de la interacción humano computador aconsejan el tamaño de botones, tipos de letras, combinación de colores, todo esto para que el usuario se sienta identificado con cada uno de los objetos dentro de la interfaz.

Podemos observar que la vista es muy interpretativa para el usuario, a continuación podemos observar mas detalladamente las vistas que se le presentaran al usuario para las distintas opciones que tiene a elegir.

Figura 3.21 Interfaz usuario Añadir Favorito

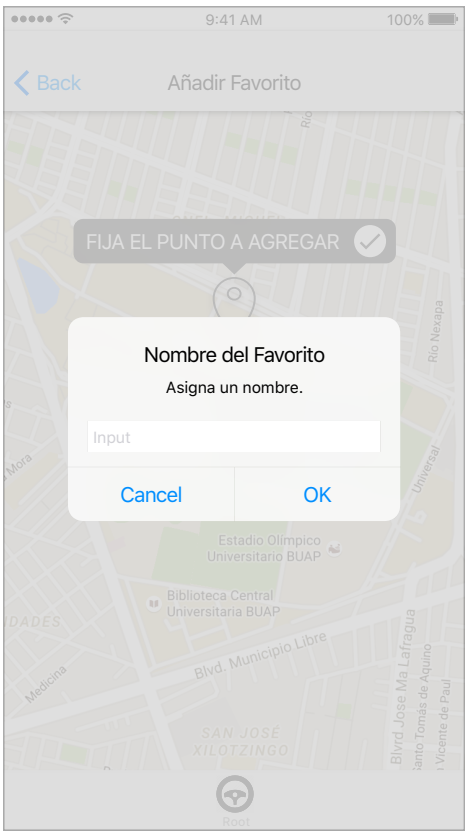


Figura 3.22 Interfaz usuario Asignar Nombre a Favorito

Figura 3.23 Interfaz usuario Consultar Favorito

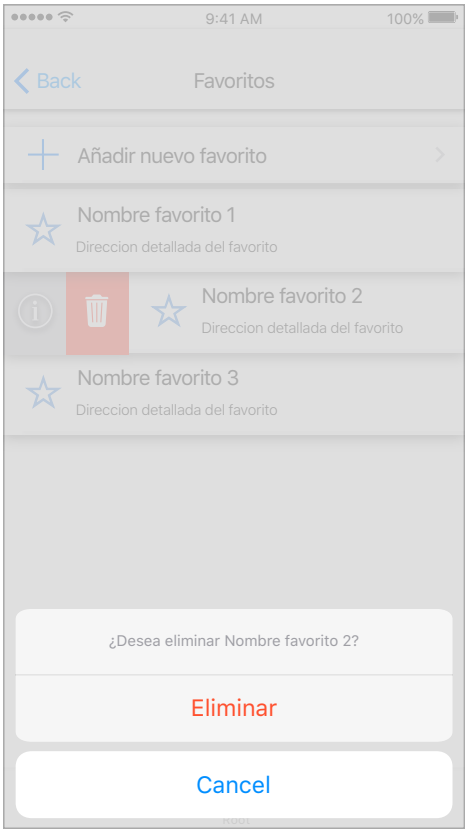


Figura 3.24 Interfaz usuario Eliminar Favorito

En la figura 3.21 podemos observar que nos aparece una chincheta la cual marca el sitio geográficamente y solo con presionar en la palomita el usuario estaría añadiendo el favorito, a continuación la aplicación nos pedirá asignarle un nombre de referencia.

Esto lo podemos observar en la figura 3.22 en la cual nos aparecerá una alerta con un cuadro de texto para escribir el nombre deseado, con dos botones, uno de cancelar y uno de confirmar.

En la figura 3.23 se muestra la interfaz del usuario para poder consultar el sitio añadido, esto con el fin de corroborar que el sitio es el correcto. En la figura 3.24 observamos que con presionar el botón rojo nos aparecerá un pop-up el cual nos pedirá eliminar o cancelar la acción.

3.4 Programación de la aplicación

Empezaremos con el desarrollo de la aplicación movil de forma física a partir de las interfaces definidas para el usuario y con ayuda de los casos de uso vistos anteriormente, para el desarrollo de la mismo se trabajara en el ambiente iOS a través de SDK Xcode, para el desarrollo de los servicios web se usara la plataforma brindada por google de nombre google maps.

3.4.1 Xcode

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Xcode trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Swift, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbón y Java. Otras compañías han añadido soporte para GNU Pascal, Free Pascal, Ada y Perl.

Entre las características más apreciadas de Xcode está la tecnología para distribuir el proceso de construcción a partir de código fuente entre varios ordenadores, utilizando Bonjour.

Xcode ofrece a los desarrolladores todo lo necesario para crear aplicaciones para Mac, iPad y iPhone. La última versión cuenta con un diseño de interfaz de usuario que unifica la codificación, pruebas y depuración dentro de una única ventana.

La aplicación incluye el IDE de Xcode, el compilador LLVM, instrumentos, simulador de iOS, el SDK de los últimos iOS y OS X y muchas características más al alcance de los desarrolladores. Entre otras cosas, cuenta con una amplia variedad de herramientas innovadoras para crear aplicaciones, el editor profesional se mantiene enfocado en el código, se ha simplificado la interfaz para hacer que sea mucho más rápido y fácil de usar, así como varias herramientas para analizar el rendimiento visual.

3.4.2 Objective C

Objective-C es un lenguaje de programación orientado a objetos creado como un superconjunto de C para que implementase un modelo de objetos parecido al de Smalltalk. Originalmente fue creado por Brad Cox y la corporación StepStone en 1980. En 1988 fue adoptado como lenguaje de programación de NEXTSTEP y en 1992 fue liberado bajo licencia GPL para el compilador GCC.

Actualmente se usa como un lenguaje principal de programación para Mac OS X, iOS y GNUstep, además de swift.

3.4.3 Sqlite

Para almacenar los datos en la aplicación utilizaremos Sqlite dado que es la mejor opción para aplicaciones móviles que no manejan grandes cantidades de datos a almacenar

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una relativamente pequeña (~275 kiB) biblioteca escrita en C. SQLite es un proyecto de dominio público[1] creado por D. Richard Hipp.

A diferencia de los sistema de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones.

Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB.

3.4.4 Google Maps

Google Maps es un servidor de aplicaciones de mapas en la web que pertenece a Alphabet Inc. Ofrece imágenes de mapas desplazables, así como fotografías por satélite del mundo e incluso la ruta entre diferentes ubicaciones o imágenes a pie de calle con Google Street View.

Existe una variante a nivel entorno de escritorio llamada Google Earth que ofrece Alphabet Inc. también de forma gratuita. En 2014, los documentos filtrados por Edward Snowden revelaron que Google Maps es parte y víctima del entramado de vigilancia mundial operado por varias agencias de inteligencia occidentales y empresas tecnológicas.

Ahora ya que hemos mencionado las herramientas a utilizar y la arquitectura pasemos a la programación de la aplicación por lo que tenemos lo siguiente.

3.4.5 Implementación

Comenzaremos detallando nuestro modelo “DAO”, este contiene la función de crear la base de datos interna, dar las sentencias para dar de alta a un favorito, eliminarlo, editarlo y de mostrar la información que tengamos en nuestra base de datos.

La figura 3.25 especifica la ruta donde se almacenara nuestra base de datos, este método será nuestro intermediario para cualquier movimiento que queramos hacer en ella, se encuentra en nuestra clase llamada FavoritoDAO, si el archivo sql no existe lo creara.

```objc

#import "FavoritoDAO.h"
#import "Favorito.h"

@implementation FavoritoDAO

- (NSString *)obtenerRutaBD
{
    NSString *dirDocs;
    NSArray *rutas;
    NSString *rutaBD;

    rutas = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    dirDocs = [rutas objectAtIndex:0];

    NSFileManager *fileMgr = [NSFileManager defaultManager];
    rutaBD = [[NSString alloc] initWithString:[dirDocs stringByAppendingPathComponent:@"mapsdb.sql"]];

    if([fileMgr fileExistsAtPath:rutaBD] == NO){
        [fileMgr copyItemAtPath:[[[NSBundle mainBundle] resourcePath]
            stringByAppendingPathComponent:@"mapsdb.sql"] toPath:rutaBD error:NULL];
    }

    return rutaBD;
}
```

Figura 3.25 Creación de base de datos y acceso a la misma

```objc
- (BOOL)crearFavorito:(NSString *)nombre latitude:(double)latitude longitude:(double)longitude;
{
    NSString *ubicacionDB = [self obtenerRutaBD];

    if(!(sqlite3_open([ubicacionDB UTF8String], &bd) == SQLITE_OK)){
        NSLog(@"No se puede conectar con la BD");
        return false;
    } else {
        NSString *sqlInsert = [NSString stringWithFormat:@"INSERT INTO favorito (nombre, latitude, longitude)
            VALUES ('%@', '%f', '%f')", nombre, latitude, longitude];
        const char *sql = [sqlInsert UTF8String];
        sqlite3_stmt *sqlStatement;

        if(sqlite3_prepare_v2(bd, sql, -1, &sqlStatement, NULL) != SQLITE_OK){
            NSLog(@"Problema al preparar el statement");
            return false;
        } else {
            if(sqlite3_step(sqlStatement) == SQLITE_DONE){
                sqlite3_finalize(sqlStatement);
                sqlite3_close(bd);
            }
        }
    }

    return true;
}
```

Figura 3.26 Método de creación de Favorito

```objc
- (BOOL)actualizarFavorito:(NSInteger)favID nombre:(NSString *)nombre latitude:(double)latitude longitude:
    (double)longitude;
{
    NSString *ubicacionDB = [self obtenerRutaBD];

    if(!(sqlite3_open([ubicacionDB UTF8String], &bd) == SQLITE_OK)){
        NSLog(@"No se puede conectar con la BD");
        return false;
    } else {
        NSString *sqlUpdate = [NSString stringWithFormat:@"UPDATE favorito SET nombre = '%@', latitude = %f,
            longitude= %f WHERE favoritoID = %ld", nombre,  latitude, longitude, favID];
        const char *sql = [sqlUpdate UTF8String];
        sqlite3_stmt *sqlStatement;

        if(sqlite3_prepare_v2(bd, sql, -1, &sqlStatement, NULL) != SQLITE_OK){
            NSLog(@"Problema al preparar el statement.");
            return false;
        } else {
            if(sqlite3_step(sqlStatement) == SQLITE_DONE){
                sqlite3_finalize(sqlStatement);
                sqlite3_close(bd);
            }
        }
    }
    return true;
}
```

Figura 3.27 Método de actualización de Favorito

```objc
- (void)borrarFavorito:(NSInteger)favID
{
    NSString *ubicacionBD = [self obtenerRutaBD];

    if(!(sqlite3_open([ubicacionBD UTF8String], &bd) == SQLITE_OK)){
        NSLog(@"No se puede conectar con la BD");
        return;
    } else {
        NSString *sqlDelete = [NSString stringWithFormat:@"DELETE FROM favorito WHERE favoritoID = %ld", favID
            ];
        const char *sql = [sqlDelete UTF8String];
        sqlite3_stmt *sqlStatement;

        if(sqlite3_prepare_v2(bd, sql, -1, &sqlStatement, NULL) != SQLITE_OK){
            NSLog(@"Problema al preparar el statement.");
            return;
        } else {
            if(sqlite3_step(sqlStatement) == SQLITE_DONE){
                sqlite3_finalize(sqlStatement);
                sqlite3_close(bd);
            }
        }
    }
}
```

Figura 3.28 Método de borrado de Favorito

Con los métodos mostrados en las figuras 3.26, 3.27 y 3.28 nos encargaremos de almacenar la información necesaria del sitio que el usuario esta marcando como favorito, son los métodos de nuestra clase de modelo “DAO” llamados en el caso de uso Administrar Favoritos mostrado con anterioridad.

El agente móvil hace uso de servicios web de Google Maps para llevar a cabo sus tareas, uno de estos métodos en llamado por el método de la figura 3.29. Obtiene las coordenadas de geolocalizacion del usuario, las marca como origen, el destino es proporcionado por el sitio favorito cercano, concatena toda esta información junto con la dirección de la API de google y crea una sesión segura para intercambiar la información en formato json.

Recibida la información con ayuda de métodos de la misma API crea una polylinea para mostrarle al usuario la ruta. El agente es capas de interpretar varias rutas, sin embargo la primera recibida es la mostrada al usuario.

La aplicación por si misma proporciona información al agente móvil, tal es el caso de la localización geográfica en tiempo real la cual es obtenida por el método de la figura 3.30.

La figura 3.31 es un ejemplo de como es nuestro servicio web, este servicio web es llamado por nuestra aplicación como se muestra con anterioridad, nuestro método en esta figura llama al servicio web proporcionado por Google Maps el cual nos responde con el archivo XML, posteriormente lo parseamos y el agente se encarga de mostrar la ruta indicada al usuario, todo esto se hace de forma asíncrona y transparente al usuario.

```objc
- (void)fetchPolylineWithOrigin:(CLLocation *)origin destination:(CLLocation *)destination
    completionHandler:(void (^)(GMSPolyline *))completionHandler
{
    NSString *originString = [NSString stringWithFormat:@"%f,%f", origin.coordinate.latitude, origin.
        coordinate.longitude];
    NSString *destinationString = [NSString stringWithFormat:@"%f,%f", destination.coordinate.latitude,
        destination.coordinate.longitude];
    NSString *directionsAPI = @"https://maps.googleapis.com/maps/api/directions/json?";
    NSString *directionsUrlString = [NSString stringWithFormat:@"%@&origin=%@&destination=%@&mode=driving"
        , directionsAPI, originString, destinationString];
    NSURL *directionsUrl = [NSURL URLWithString:directionsUrlString];


    NSURLSessionDataTask *fetchDirectionsTask = [[NSURLSession sharedSession] dataTaskWithURL:
        directionsUrl completionHandler:
                                                ^(NSData *data, NSURLResponse *response, NSError *error)
                                                {
                                                    NSDictionary *json = [NSJSONSerialization
                                                        JSONObjectWithData:data options:kNilOptions
                                                        error:&error];
                                                    if(error)
                                                    {
                                                        NSLog(@"Error: %@",error);
                                                        if(completionHandler)
                                                            completionHandler(nil);
                                                        return;
                                                    }

                                                    NSArray *routesArray = [json objectForKey:@"routes"];

                                                    GMSPolyline *polyline = nil;
                                                    if ([routesArray count] > 0)
                                                    {
                                                        NSDictionary *routeDict = [routesArray
                                                            objectAtIndex:0];
                                                        NSDictionary *routeOverviewPolyline = [routeDict
                                                            objectForKey:@"overview_polyline"];
                                                        NSString *points = [routeOverviewPolyline
                                                            objectForKey:@"points"];
                                                        GMSPath *path = [GMSPath pathFromEncodedPath:
                                                            points];
                                                        polyline = [GMSPolyline polylineWithPath:path];
                                                    }

                                                    if(completionHandler)
                                                        completionHandler(polyline);
                                                }];
    [fetchDirectionsTask resume];
}
```

Figura 3.29 Método de invocación de servicio web

```objc
- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations {

    _location = [locations lastObject];
    if (_startStop) {
        [self drawRoute];
    }

    if (_locationMarker == nil) {
        _locationMarker = [[GMSMarker alloc] init];

        _locationMarker.position = _location.coordinate;
        NSArray *frames = @[[UIImage imageNamed:@"auto"]];

        _locationMarker.icon = [UIImage animatedImageWithImages:frames duration:0.8];
        _locationMarker.groundAnchor = CGPointMake(0.5f, 0.97f);
        _locationMarker.map = _mapView;
    } else {
        [CATransaction begin];
        [CATransaction setAnimationDuration:2.0];
        _locationMarker.position = _location.coordinate;
        [CATransaction commit];
    }

    GMSCameraUpdate *move = [GMSCameraUpdate setTarget:_location.coordinate zoom:20];
    [_mapView animateWithCameraUpdate:move];
}
```

Figura 3.30 Método de localización de usuario

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" xmlns:gpxx="http://www.garmin.com/xmlschemas/
    GpxExtensions/v3" xmlns:gpxtpx="http://www.garmin.com/xmlschemas/TrackPointExtension/v1" creator=
    "mapstogpx.com" version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd http://
    www.garmin.com/xmlschemas/GpxExtensions/v3 http://www.garmin.com/xmlschemas/GpxExtensionsv3.xsd
    http://www.garmin.com/xmlschemas/TrackPointExtension/v1 http://www.garmin.com/xmlschemas/
    TrackPointExtensionv1.xsd">
  <metadata>
    <link href="http://www.mapstogpx.com">
      <text>Sverrir Sigmundarson</text>
    </link>
    <!--desc>Map data copyright 2016 Google</desc-->
    <!--copyright author="Google Inc">
        <year>2016</year>
        <license>https://developers.google.com/maps/terms</license>
    </copyright-->
    <!--url>https://www.google.co.uk/maps/dir/19.0437437,-98.1979171/19.0437944,-98.1977991/
        @19.0432706,-98.1989169,18z/data=!4m2!4m1!3e0?hl=en</url-->
    <time>2016-07-15T17:59:43Z</time>
  </metadata>
  <wpt lat="19.0437437" lon="-98.1979171">
    <name>Avenida Don Juan de Palafox y Mendoza 232</name>
    <desc>Avenida Don Juan de Palafox y Mendoza 232, Centro, Heroica Puebla de Zaragoza, Pue., Mexico</
        desc>
    <time>2016-07-15T17:59:43Z</time>
  </wpt>
  <wpt lat="19.044" lon="-98.19778">
    <name>WP001</name>
    <time>2016-07-15T18:04:43Z</time>
  </wpt>
  <wpt lat="19.04397" lon="-98.19771">
    <name>WP002</name>
    <time>2016-07-15T18:09:43Z</time>
  </wpt>
  <wpt lat="19.0437944" lon="-98.1977991">
    <name>Avenida Don Juan de Palafox y Mendoza 232</name>
    <desc>Avenida Don Juan de Palafox y Mendoza 232, Centro, Heroica Puebla de Zaragoza, Pue., Mexico</
        desc>
    <time>2016-07-15T18:14:43Z</time>
  </wpt>
</gpx>
```

Figura 3.31 Servicio web

Ahora ya que hemos visto la programación de la aplicación pasamos a las pruebas, la ubicación de sitios de estacionamiento y mapeo de otras áreas se obtiene al momento en que el equipo móvil se sincroniza con el servicio web, esto se hace de manera nativa en la aplicación dado que el mapeo de estas áreas conlleva un tiempo considerable de localización, por lo que la aplicación de momento solo esta destinada a áreas adyacente a la facultad de computación.

La aplicación fue diseñada para equipos iOS, por lo que al final la aplicación se construyo para iPads, iPhones y iPods, así mismo la prueba será sobre equipos iPhone.

La principal finalidad de la aplicación es tener un inicio en la llamada inteligencia ambiental, atacando como principal objetivo la contaminación por vehículos. La mayoría de las veces uno no encuentra estacionamiento cercano a donde se transita con mayor frecuencia, originando así mayor contaminación en el medio ambiente.

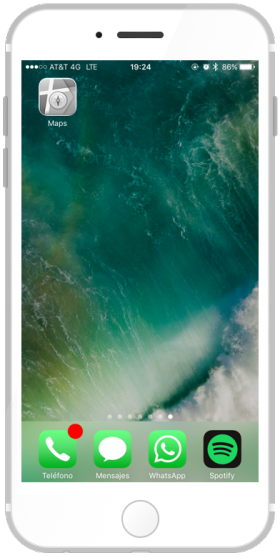


Figura 3.32 Aplicación instalada

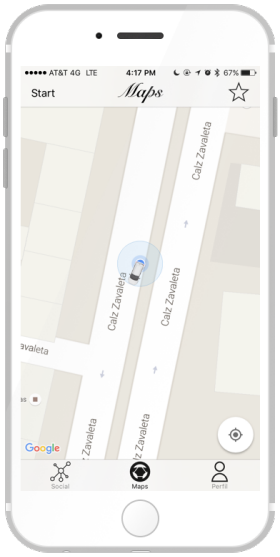


Figura 3.33 Interfaz principal

Podemos observar como nuestro usuario ya cuenta con la aplicación instalada en su smartphone (figura 3.32), al ingresar a ella se sitúa en una vista la cual le indicara su localización en tiempo real, todo esto con ayuda de google maps (figura 3.33), al presionar la estrella en la esquina superior derecha accede a su lista de sitios favoritos previamente añadidos.

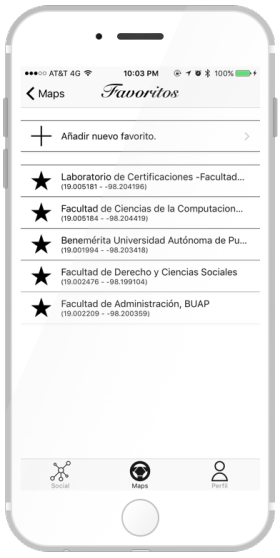


Figura 3.34 Interfaz gestión Favorito

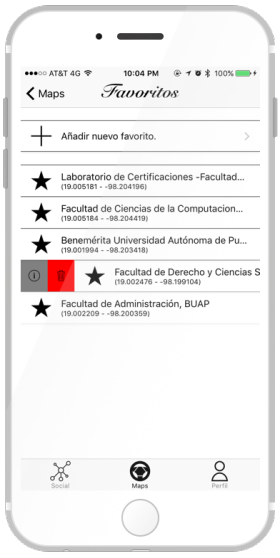


Figura 3.35 Interfaz menu emergente

El usuario desliza una de las celdas hacia la derecha mostrando el menú emergente (figura 3.34). El usuario presiona el botón gris del menú emergente mostrando una vista con las propiedades de la celda seleccionada (figura 3.35).

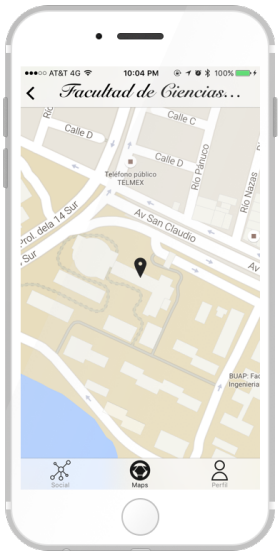


Figura 3.36 Vista de Favorito agregado previamente

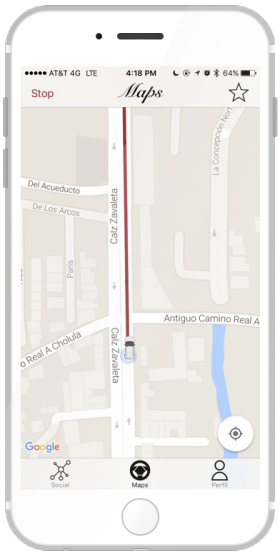


Figura 3.37 Vista camino

El usuario inicia la búsqueda de un sitio donde estacionarse, la ruta hacia el mejor sitio se marca con una linea de color rojo. El agente dirige al usuario al sitio mas cercano dependiendo del sitio almacenado (figuras 3.36 y 3.37).

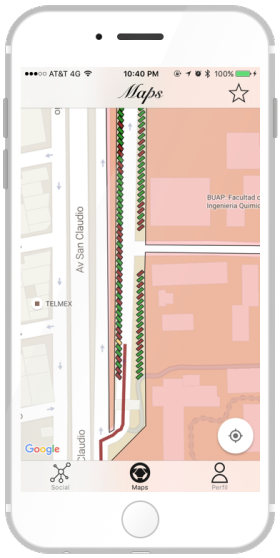


Figura 3.38 Navegación I

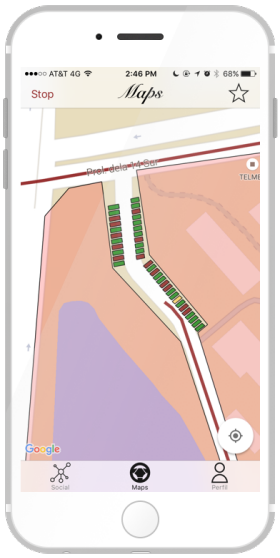


Figura 3.39 Navegación II

La aplicado cambiara el modo de vista dependiendo de la distancia que tenga con respecto al sitio seleccionado por el agente. El usuario visualiza los sitios con tres tipos diferentes de colores (figuras 3.38 y 3.39).

El agente se comunica con el gestor de agentes el cual avisara a los demás agentes de lo acontecido en los lugares donde tiene mapeado lugares. La comunicación se realiza por servicios web de google, los cuales comunican que sitios están siendo apartados por agentes remotos. De esta forma evitamos que exista duplicidad de información.

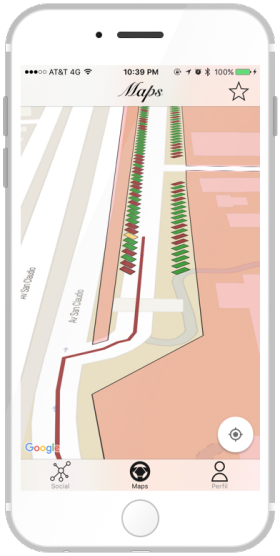


Figura 3.40 Llegada de usuario a destino I

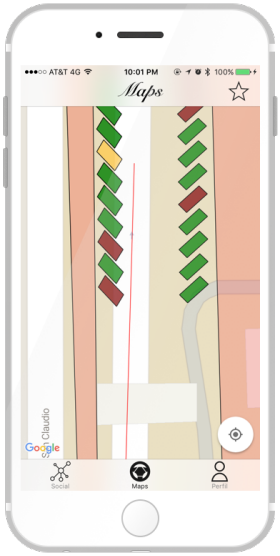


Figura 3.41 Llegada de usuario a destino II

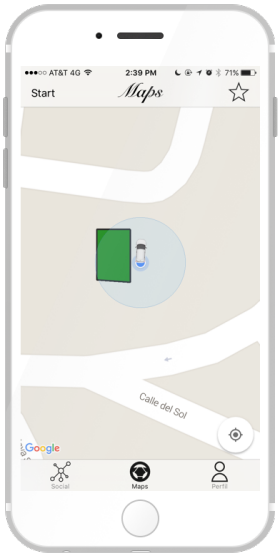


Figura 3.42 Cajón vacío

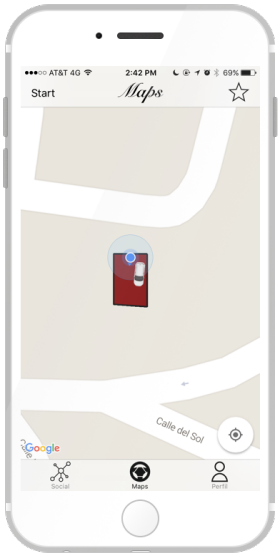


Figura 3.43 Cajón ocupado

El usuario aparca en el sitio indicado, el agente puede detectar cuando el usuario se localiza en un cajón y de esta manera cambiar el color, verde para el cajón desocupado (figura 3.42) y rojo para el cajón ocupado (figura 3.43), al hacer esto el agente no solo avisa mediante la pantalla a los demás usuarios, informa mediante el gestor de asistentes a todos los demás agentes cercanos que se encuentra ocupando un sitio.

CAPÍTULO IV RESULTADOS

Los resultados que se nos presentan, ya al haber hecho una investigación sobre los dispositivos móviles, sus aplicaciones diarias y el gran auge en el que se encuentran todos ellos, junto con la importancia, el incremento y el gran desempeño que tienen los agentes móviles, nos llevo a la propuesta de una metodología cuyo tiempo de investigación y maduración ya tenia tiempo en el ámbito de desarrollo de este tipo de tecnologías para la creación de aplicaciones para dispositivos móviles. Para todo esto tuvimos corroborar que esta metodología tiene un buen desempeño o nos puede dar una aproximación a la idea de cómo se deben crear dichas aplicaciones; se pueden ver como los pasos que se deben seguir para crear una aplicación móvil. Veamos lo que se creó y su funcionamiento.

Presentación de la interfaz principal de la aplicación.

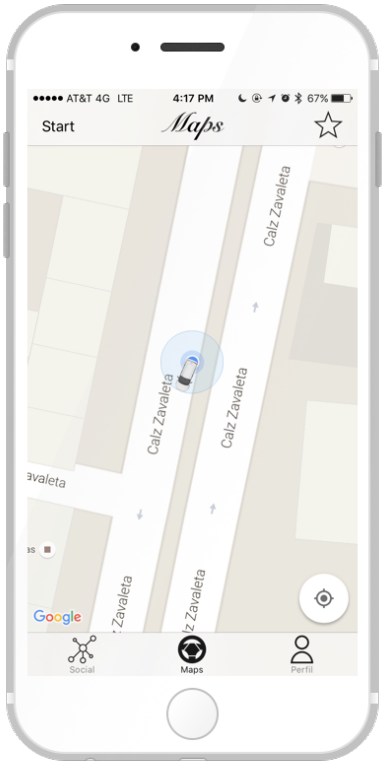


Figura 4.1 Interfaz principal

De la vista anterior podemos observar que nos muestra dos opciones en la parte superior una de comenzar, si deseamos que la aplicación nos encuentre un lugar de estacionamiento adecuado a nuestros sitios previamente almacenados en favoritos.

Otra opción es la de una estrella en la cual el usuario puede editar los sitios favoritos, tal es el caso de añadir, modificar, eliminar o simplemente consultar.

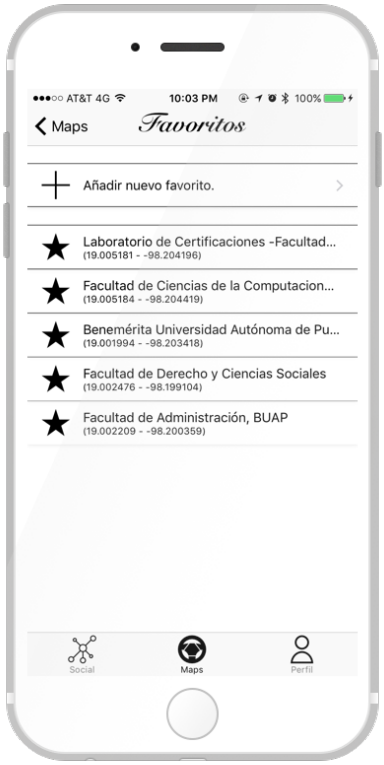


Figura 4.2 Interfaz de añadir, eliminar, consultar o modificar Favorito

Como se muestra en la figura 4.2 la vista despliega una tabla, la primera celda es para agregar un favorito las demás celdas nos muestra los favoritos almacenados, con su nombre asignado y su geolocalización.

Si el usuario desea agregar un nuevo favorito basta con presionar la primera celda, se nos desplegara una vista nueva con nuestra geolocalización, con ayuda de una chincheta podremos mover el mapa situando la chincheta en el sitio que deseemos añadir.

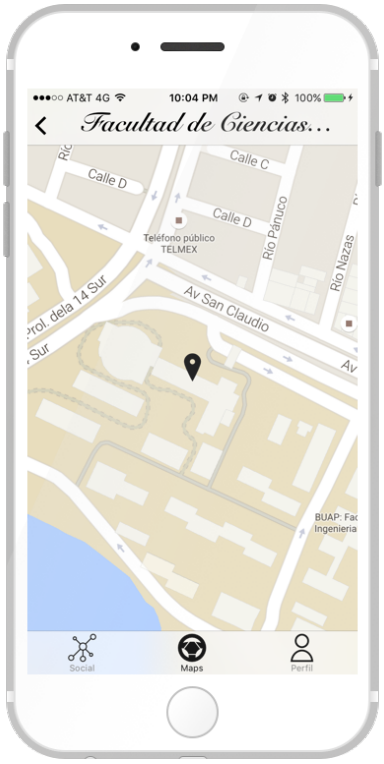


Figura 4.3 Interfaz de agregar y consultar Favorito

La figura 4.3 detalla la vistas correspondientes para agregar y consultar algún sitio.

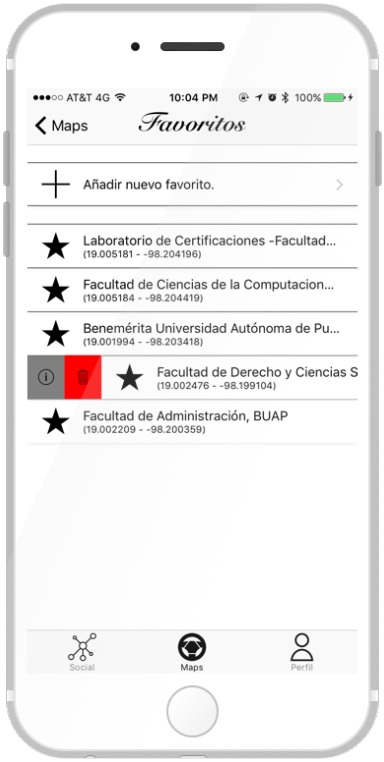


Figura 4.4 Interfaz de eliminar Favorito

Para eliminar algún favorito basta con deslizar la celda del favorito que deseemos eliminar hacia el lado derecho, esto nos mostrara un menú emergente con las opciones de obtener información detallada del sitio o eliminar con un botón rojo, al presionar sobre este nos pedirá confirmación, todo esto para mantener informado al usuario de si lo que se encuentra realizando es lo deseado.

Cabe resaltar que la vista de inicio de la aplicación es nuestra geolocalizacion resaltado por un vehículo, el usuario puede empezar la búsqueda de un sitio para estacionarse si así lo desea con el botón superior izquierdo.

Sin embargo cuando el agente detecta que el usuario se encuentra cerca de un sitio previamente añadido como favorito empezara el mapeo de la mejor ruta hacia el sitio de estacionamiento mas adecuado.

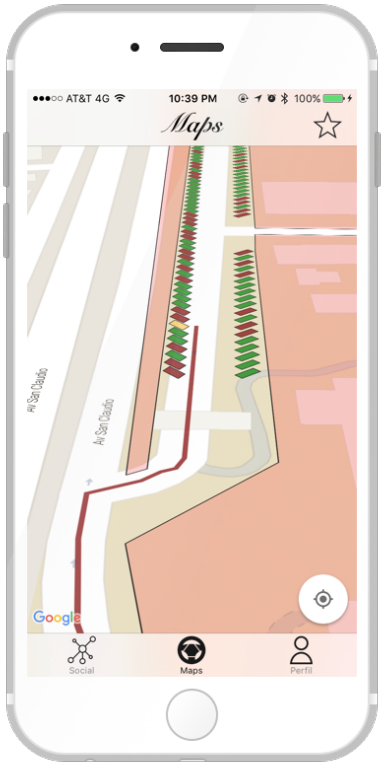


Figura 4.5 Interfaz de navegación

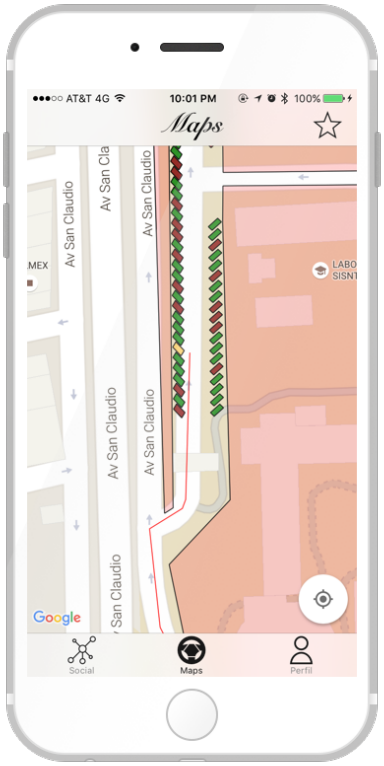


Figura 4.6 Interfaz de navegación vista superior

La posición de la cámara es definida por la distancia al objetivo, el usuario puede modificar la posición de la misma a su conveniencia, como se muestra en las figuras 4.5 y 4.6.

El usuario puede navegar por todo el mapa, sin embargo se le proporciona la posibilidad de volver a centrarse en su ruta con el botón de centrar ubicado en la parte inferior derecha.

Los sitios donde estacionarse son marcados por 3 colores distintos, el rojo que significa ocupado, el verde que significa libre, y el amarillo que significa como mejor opción para el usuario y que ningún otro agente o usuario puede hacer uso del mismo.

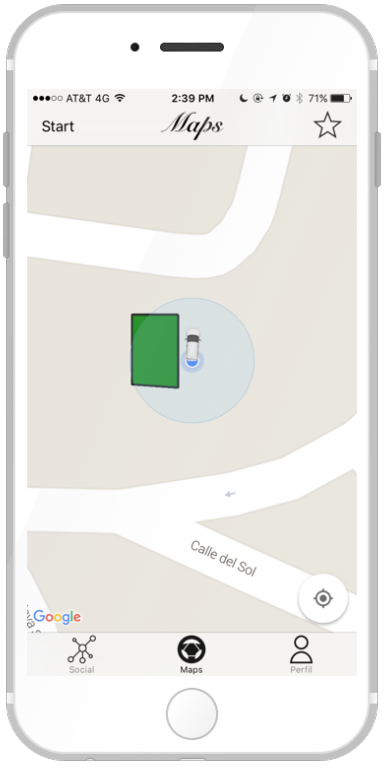


Figura 4.7 Sitio desocupado

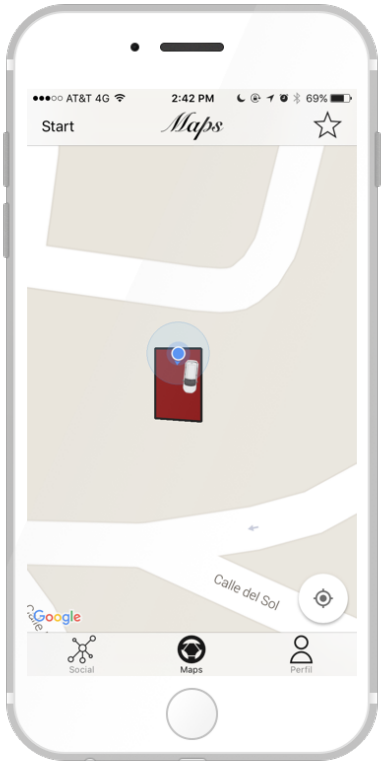


Figura 4.8 Sitio ocupado

Las figuras 4.7 y 4.8 indican la forma en que el agente sabe cuando un sitio ya fue ocupado y cuando no, si el usuario no entra en el perímetro marcado el cajón continuara de color verde, como se muestra en la figura 4.7.

Una vez que el usuario entra en el cajón, este cambiara a color rojo, indicando a los demás agentes y usuarios que el sitio ya fue ocupado, tal y como se muestra en la figura 4.8.

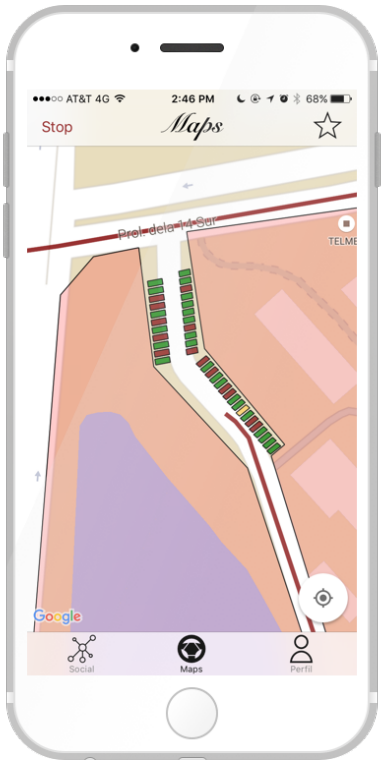


Figura 4.9 Vista superior de la ciudad mapeada

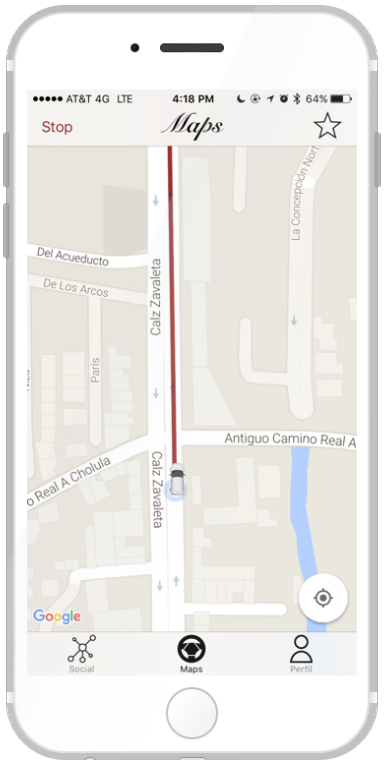


Figura 4.10 Vista superior de la ciudad sin mapear

Para concluir con la aplicación, dentro de la misma tiene mapeado los cajones para estacionarse, y las áreas donde un vehículo no puede entrar, como se muestra en la figura 4.9 los sitios donde un vehículo no puede pasar son coloreados de color melón, cuando el usuario deja su vehículo el agente detecta que esta entrando en una de estas áreas y detiene el procedimiento de rastreo, guarda la ultima localidad y cuando vuelve a estar próxima inicia sus tareas nuevamente.

En la figura 4.10 podemos observar que no todas las regiones están mapeadas debido a que esto conlleva un tiempo considerable de desempeño por el lado del servicio web.

CAPÍTULO V CONCLUSIONES Y TRABAJOS FUTUROS

Como se ha podido observar el principal objetivo de la elaboración de esta tesis fue mostrar las ventajas que ofrecen metodologías como la utilizada en este trabajo. La elaboración de aplicaciones móviles a diferencia de sus homologas las aplicaciones de escritorio, estas tienen un igual o mayor grado de complejidad en su creación. Una aplicación móvil es un super conjunto de una aplicación de escritorio, por lo cual debe ser desarrollada con metodologías que conlleven procesos para dicha construcción.

Resultados mostrados en este documento garantizan que el análisis, diseño y desarrollo de este tipo con base en lo que se propuso para su realización, se obtiene una aplicación que cumple con los requisitos y lineamientos que hasta el momento fueron presentados, el darle solución al principal problema que se quiere atacar será solucionado al momento que esta misma aplicación pueda expandirse en otros usuarios.

Al momento de crear este trabajo nos encontramos con aspectos demasiado curiosos, dado que en varios lugares no se diferencia entre metodologías propias para el desarrollo de este tipo de arquitecturas, desarrollando así aplicaciones con análisis y diseño de sistemas completamente diferentes. Lo que nos lleva a enfatizar la necesidad por establecer metodologías propias para cada entorno, metodologías especificas garantizan desarrollos eficientes.

Por el lado de los agentes móviles se pretende rescatar una rama de la inteligencia artificial la cual tiene varios puntos a favor a tener en consideración como se menciono en el capitulo II, sin embargo el campo de estudio en esta área ha sido un poco descuidada. Nuestra propuesta demuestra que tecnologías como estas aunadas a arquitecturas como lo son los agentes móviles tienen un optimo desempeño.

Como trabajo futuro queda un largo campo por explorar en ambas ramas, tareas principales como el mejor desarrollo de la metodología M-UML, enfoque mas conciso a los agentes utilizados y una expansión en el desarrollo de la aplicación que sin duda garantizara una mejor calidad de vida en aspectos tan sencillos del día a día.

BIBLIOGRAFIA

[1] F. Ahmad Hanandeh, I. Alsmadi, M. Yousef Al-Shannag, E. Al-Daoud. “Mobile agents modeling using UML”, International Journal of Business Information Systems, Vol. 19, Issue 4, pp. 419-432, 2015.

[2] R. B’Far, Roy T. Fielding. “Mobile computing principles: Designing and developing mobile applications with UML and XML”, Cambridge University Press, 2004.

[3] P. Braun, W. R. Rossak. “Mobile agents: Basic concepts, mobility models and the tracy toolkit”, Morgan Kaufmann, 2005.

[4] K. Saleh, C. El-Morr. “M-UML: An extension to UML for the modeling of mobile agent-based software systems”, *Information and Software Technology*, No. 46, pp. 219-227, 2004.

[5] R. Sidnei W. “Object-oriented analysis and design for information systems: Modeling with UML, OCL and IFML”, *Morgan Kaufmann*, 2014.

[6] M. Redha Bahri, R. Mokhtari, A. Chaoui. “Towards an extension of UML 2.0 to model mobile agent-based systems”, *International Journal of Computer Science and Network Security*, Vol. 9, No. 10, 2009

[7] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1998.

[8] D. Lange, M. Oshima, Seven good reasons for mobile agents, Communications of the ACM 42 (3) (1999) 88 – 99.

[9] C. Wagner, E. Turban, Are intelligent e-commerce agents partners or predators, Communications of the ACM 45 (5) (2002) 84 – 90.

[10] C. Klein, A. Rausch, M. Sihling, Z. Wen, Extension of the unified modeling language for mobile agents, in: K. Siau, T. Halpin (Eds.), Unified Modeling Language: Systems Analysis, Design and Development Issues, Idea Group Publishing, 2001, pp. 116 – 128, (Chapter VIII).

[11] B. Bauer, J. Muller, J. Odell, Agent UML: a formalism for specifying multiagent interaction, 22nd International Conference on Software Engineering (ICSE), Agent-Oriented Software Engineering, Springer, Berlin, 2001, pp. 91–103.

[12] H. Baumeister, N. Koch, P. Kosiuczenko, M. Wirsing, Extending, activity diagrams to model mobile systems, NetObjectDays, NODe, Erfurt, Germany, October, 2002.

[13] M. Fowler, K. Scott, UML Distilled, Addison-Wesley, Reading, MA, 1999.

[14] H. Nwana, et al., Agent-mediated electronic commerce: issues challenges and some viewpoints, Proceedings of The Workshop on Agent Mediated Electronic Trading (AMET'98), Minnesota, USA.

[15] P. Maes, R. Guttman, A. Moukas, Agents that buy and sell: transforming commerce as we know it, Communications of the ACM 42 (3) (1999).

[16] G. Vigna (Eds.), Mobile agents and security, Lecture Notes in Computer Science LNCS 1419, 1999.

[17] K. Saleh, C. E1-Morr, Specification and architecture of a mobile agent-based bartering business model, International Conference on Electronic Commerce (ICEC 2001), Austria, November (2001).

[18] D. Wong, N. Paciorek, D. Moore, Java-based mobile agents, Communications of the ACM 42 (3) (1999) 92 – 102.

[19] K. Saleh, C. El-Morr, A. Mourtada, Y. Morad, Specifications of a mobile electronic voting system and a mobile agent platform, IADIS e-Society 2003, Lisbon, Portugal, June 3 – 6, 2003.

[20] C. El-Morr, K. Saleh, MABASO: a mobile agent-based bartering software system, Second Proceeding of the ACS/IEEE International, Conference on Computer Systems and Applications, Tunis, 2003. [16] K. Saleh, A. Mourtada, Y. Morad, A prototype for a mobile-agent platform and a game application, Second International Conference on the Application and Development of Computer Games (ADCOG 2003), Hong Kong, China, January (2003).