UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

TÍTULO: Módulo de Control de Flotas para el Sistema de Información Geográfica GeoQ

.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor

Humberto Fernández Morris

Tutor

Ing. Claritza Sánchez Veranes

La Habana, Julio de 2011
"Año del 50 Aniversario de la Revolución"

"El triunfo es de los que se sacrifican."

"Solo perdura y es para bien, la riqueza que se crea y la libertad que se conquista, con las propias manos."

José Julián Martí Pérez (1853-1895)

Con todo el amor del mundo a mi papá Humberto, a mi mamá Irina y a mi hermana Anet…

a mi tío Jorge y a mi abuela Elsa …

a mi novia y amistades…

AGRADECIMIENTOS

Deseo mostrar mi agradecimiento a la Revolución y a Fidel por haberme dado la posibilidad de ser parte de este importante proyecto que es la UCI.

A mi tutora Claritza Sánchez Veranes por haber sido mi guía y apoyo principal todo este tiempo para el desarrollo exitoso de la tesis.

A los miembros del tribunal por sus precisas acotaciones que le dieron a este trabajo mayor calidad y rigor científico.

A mi mamá Irina, a mi papá Humberto y a mi hermana Anet, por apoyarme en cada momento de la vida y tener confianza en mí por sobre todas las cosas, por su comprensión y por el optimismo que me han transmitido siempre.

A mi tío Jorge por la atención que me ha brindado todo este tiempo, por ayudarme en lo que está a su alcance y por estar siempre tan pendiente de mí.

A toda mi familia, que siempre han estado pendientes de mí, en las buenas y en las no tan buenas.

A mis buenos amigos Jose Angel, Yasnaris, Yoandy, Gelsys, Alejandro y Leandro, por toda la ayuda y por compartir buenos y malos momentos conmigo, por ser más que mis amigos, mis hermanos.

A todos los que estuvieron a mi lado en esta etapa. A mis compañeros de grupo. Gracias a todos por haberme permitido entrar en sus vidas. Ha sido un privilegio haber contado con ustedes.

A mi novia, que con su forma de ser ha sido un ejemplo a seguir para mí, la cual me ha ayudado a lo largo de todo el desarrollo de este trabajo siendo la base que me ha sostenido y que me ha dado fuerzas para seguir adelante.

DECLARACIÓN DE AUDITORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____.

_____ _____

Humberto Fernández Morris. Ing. Claritza Sánchez Veranes.

RESUMEN

El presente trabajo, titulado: “Módulo de Control de Flotas para el Sistema de Información Geográfica GeoQ”, se realiza como alternativa de solución a la inexistencia de un módulo de control de flota en la plataforma GeoQ, de manera que permita realizar el monitoreo de objetos móviles de forma remota, la reconstrucción del comportamiento de los mismos en un determinado período de tiempo y la reelaboración de una trayectoria.

Está diseñado sobre una arquitectura tres capas y se implementó sobre plataformas de *software* libre, utilizando como gestor de base de datos PostgreSQL y sus utilidades están desarrolladas en el lenguaje de programación C++, teniendo como entorno de desarrollo integrado Qt. Con el objetivo de guiar el proceso de desarrollo se utilizó la metodología de *software* AUP (Proceso Unificado Ágil) y la modelación de la ingeniería se hizo mediante el lenguaje de modelado UML (Lenguaje Unificado de Modelado), asistido por la herramienta CASE Visual Paradigm.

Entre los resultados obtenidos se encuentran el módulo de control de flotas de la Plataforma de Información Geográfica GeoQ y la documentación UML de los artefactos resultantes del análisis y diseño del sistema de gestión de datos e información.

Palabras claves: control de flotas, objetos móviles, SIG.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con la disminución del tamaño, el aumento de velocidad y de memoria de las computadoras se ha experimentado enormes progresos en las Tecnologías de la Información y las Comunicaciones (TIC). Estas, son un conjunto de servicios, redes, *software* y dispositivos que tienen como fin la mejora de la calidad de vida de las personas dentro de un entorno, que se integran a un sistema de información interconectado y complementario, y están destinadas a optimizar la comunicación humana (ECHEVERRIA, 2007).

En Cuba, se aplican en todas las esferas de la vida, en los servicios, la salud, la educación, las investigaciones y la gestión económica. La Universidad de las Ciencias Informáticas (UCI) es un ejemplo vivo y motor impulsor de las mismas en el país debido al creciente auge y expansión del mundo de la información, que ha traído consigo la necesidad del manejo y control de cada vez mayores cantidades de información.

En la actualidad la universidad está estructurada en 7 facultades, las cuales están enfocadas en diferentes ramas de la producción de *software*. Específicamente en la facultad 6, se encuentra el centro GEySED que unifica los departamentos de Señales Digitales y Geoinformática, este último con el objetivo del desarrollo de los sistemas de manejo de información llamados Sistemas de Información Geográfica (SIG), los cuales no son más que una base de datos relacionada con un *software* gráfico, o un sistema que permite gestionar datos alfanuméricos espacialmente localizados (ECHEVERRIA, 2007). Su utilidad puede manifestarse en diversas actividades relacionadas con la tecnología de computadoras para integrar, manipular y visualizar una gran variedad de datos capaces de crear una imagen de la geografía, medio ambiente y características socioeconómicas de una zona determinada estudiada, sin dejar de mencionar uno de sus objetivos más importantes, que constituye la ayuda en la toma de mejores decisiones.

El proyecto SIG-DESKTOP perteneciente al departamento de Geoinformática se enfoca en desarrollar soluciones enteramente en plataforma de escritorio y utilizando herramientas libres como base. Las

aplicaciones de escritorio representan una herramienta muy poderosa y en complemento con las tecnologías libres, dan a la luz un producto potente y con muchas facilidades para los clientes.

Con el transcurso del tiempo el proyecto ha venido dando el frente a variados compromisos y llamando la atención de diferentes clientes, lo cual ha posibilitado la realización de varios productos. Entre estos productos se encuentra la plataforma de información geográfica GeoQ, la cual fue desarrollada sobre un producto de licencia GNU/GPL (*General Public License*) de código abierto. Esta brinda la posibilidad de realizar consultas, análisis, representación geoespacial de la información referente a un determinado negocio y además, una solución más rápida debido a las funcionalidades ya existentes en la aplicación base. Actualmente, dicha plataforma no realiza el monitoreo de objetos móviles de manera remota, además, resulta imposible la reconstrucción del comportamiento de los mismos en un determinado período de tiempo y reelaborar su trayectoria.

A partir del análisis de la situación problemática de la investigación se define como el **Problema a resolver** la siguiente interrogante: ¿Cómo lograr el control de flotas en el Sistema de Información Geográfica GeoQ?

Una vez identificado el problema reflejado con anterioridad, fue necesario centrar la investigación en el proceso de control de flotas, lo cual constituye el **objeto de estudio** de la misma. Además, se hizo necesario para resolver dicho problema, que el **objetivo general** esté dirigido a implementar el módulo de control de flotas de la plataforma GeoQ.

El conjunto de principios que serán planteados como solución de esta investigación tendrán su incidencia final en la automatización del proceso de control de flotas, lo cual determina el **campo de acción**.

En esta investigación se defiende la siguiente idea: Si se implementa un módulo para el control de flotas del Sistema de Información Geográfica GeoQ, se podrá contar con una aplicación de mayores prestaciones.

Se determinó también, que las **tareas a realizar** para darle cumplimiento al objetivo trazado estarían dirigidas a:

- Caracterizar el proceso de control de flotas.
- Caracterizar el proceso de representación de archivos de posicionamiento global.

- Determinar las herramientas y tecnologías a utilizar en el desarrollo de la aplicación.
- Elaborar el Diagrama del Modelo de Dominio.
- Especificar requisitos funcionales del sistema.
- Elaborar el Diagrama de Casos de Uso del Sistema.
- Elaborar los diagramas de clases del diseño.
- Elaborar el diagrama de implementación.
- Implementar los CU definidos.
- Diseñar los casos de prueba a emplear en la aplicación.

Entre los **posibles resultados** del trabajo se encuentran:

- Documentación UML de los artefactos resultantes del análisis y diseño del sistema de Gestión de datos e información.
- Modelo del Negocio.
- Requerimientos (Funcionales, No Funcionales).
- Modelo del Sistema.
- Modelo de Análisis.
- Módulo de Control de Flotas para el Sistema de Información Geográfica GeoQ.

La realización de la investigación está dirigida mediante **métodos científicos** que se fueron aplicando durante su desarrollo.

Teóricos:

- **Analítico-Sintético**: Se realizó un detallado estudio de toda la bibliografía, conformada por documentos, sitios Web y tesis, lo que posibilitó obtener una síntesis detallada de esta con los aspectos más importantes referentes al tema.

- **Análisis Histórico Lógico**: Se investigó y se hizo el análisis de las aplicaciones existentes hasta la actualidad que implementan el control de flotas.

- **Modelación**: Se modelaron las actividades que se desarrollan en el transcurso de la elaboración del problema a través de diagramas.

Empíricos:

- **Entrevistas**: Se realizaron entrevistas de forma no estructurada a los dirigentes del centro de desarrollo GEySED de la facultad 6 para recoger toda la información necesaria sobre las posibles funcionalidades a desarrollar.

Estructura del Documento

El presente documento se encuentra dividido en cuatro capítulos.

Capítulo 1: Fundamentación Teórica, se realiza una caracterización de los principales conceptos asociados al dominio del problema. Además, se hace un análisis del estado del arte que precede a la realización de este trabajo, presentando algunas de las soluciones existentes en la actualidad.

Capítulo 2: Tendencias y tecnologías actuales, se describen las metodologías de desarrollo, lenguajes de programación y tecnologías actuales para el desarrollo de la solución presentada, justificando además la selección y el uso de estas.

Capítulo 3: Presentación de la solución propuesta, se propone la solución al problema planteado teniendo en cuenta la modelación del dominio del problema, la identificación de los requisitos funcionales y no funcionales de la aplicación, así como los casos de uso del sistema.

Capítulo 4: Construcción de la solución propuesta, se presentan elementos claves en el desarrollo del sistema propuesto como el diagrama de clases del diseño, el diseño de la base de datos a utilizar, el modelo de despliegue y el diagrama de componentes. Se ofrece una visión general de la implementación y de los estándares de diseño y codificación utilizados en la solución propuesta.

Finalmente se dan las conclusiones generales y un conjunto de recomendaciones con vistas a trabajos futuros.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1. Introducción

En el presente capítulo se comentan los aspectos más importantes que van a servir como soporte teórico para el desarrollo del trabajo. Se realiza una caracterización de temas fundamentales tales como los SIG, datos espaciales, metadatos, control de flotas y sistemas de posicionamiento global. Además, se caracteriza el proceso de control de flotas y se hace un análisis del estado del arte que precede a la realización de este trabajo y que contribuye a esclarecer su objeto de estudio.

1.1 Conceptos asociados al dominio del problema

Cada vez que se plantea la tarea de realizar una investigación, surgen conceptos esenciales asociados al problema abordado, los cuales constituyen la base fundamental del tema tratado y resulta de suma importancia su comprensión para el desarrollo del trabajo. A continuación se relacionan los principales conceptos asociados al dominio del problema, los cuales facilitarán la comprensión de la presente investigación.

1.1.1 Sistema de Información Geográfica (SIG)

Un Sistema de Información Geográfico (SIG) particulariza un conjunto de procedimientos sobre una base de datos no gráfica o descriptiva de objetos del mundo real que tienen una representación gráfica y que son susceptibles de algún tipo de medición respecto a su tamaño y dimensión relativa a la superficie de la tierra. A parte de la especificación no gráfica, el SIG cuenta también con una base de datos gráfica con información georeferenciada o de tipo espacial y de alguna forma ligada a la base de datos descriptiva. En un SIG se usan herramientas de gran capacidad de procesamiento gráfico y alfanumérico, las cuales van dotadas de procedimientos y aplicaciones para captura, almacenamiento, análisis y visualización de la información georeferenciada.

Un SIG se puede definir como una tecnología de manejo de información geográfica formada por equipos electrónicos (*hardware*) programados adecuadamente (*software*) que permiten manejar una serie de datos espaciales (información geográfica, datos geográficos) y realizar análisis complejos con estos siguiendo los criterios impuestos por el equipo científico (personal o equipo humano)(Rubi, 2008). En esencia, pueden concretarse como sistemas con la capacidad de manipular datos espaciales, brindándonos una herramienta que permite visualizar y analizar la información de forma versátil e intuitiva, agilizando la tan importante toma de decisiones (Farré, 2010).

1.1.2 Datos Espaciales

Los datos espaciales son el componente fundamental de cada proyecto o aplicación SIG. Contienen las ubicaciones y formas de características cartográficas. Son también conocidos como Datos Cartográficos Digitales. Dentro de su contexto, almacenan informaciones sobre la localización y las formas de un objeto geográfico y las relaciones entre ellos, normalmente con coordenadas y topología.

Los datos espaciales se refieren a entidades o fenómenos que cumplen los siguientes principios básicos (Franco, 2009):

- Tienen posición absoluta: sobre un sistema de coordenadas (x, y, z).
- Tienen una posición relativa: frente a otros elementos del paisaje (topología: incluido, adyacente, cruzado, etc.).
- Tienen una figura geométrica que las representan (punto, línea, polígono).
- Tienen atributos que los describen (características del elemento o fenómeno).

Espacialmente hay datos de n dimensiones (Franco, 2009):

- Puntuales: Asociados a un punto (Ver Figura 1).

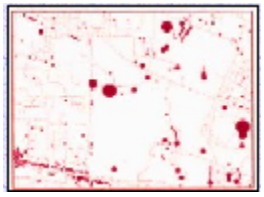


Figura 1: Representación de datos puntuales.

- Lineales: Asociados a una línea (Ver Figura 2).

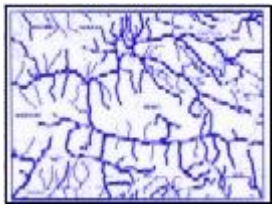


Figura 2: Representación de datos Lineales.

- Superficiales: Asociados a superficies (Ver Figura 3).

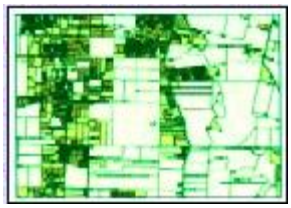


Figura 3: Representación de datos poligonales.

- Volumétricos: Asociados a un volumen (Ver Figura 4).

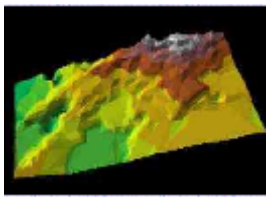


Figura 4: Representación de datos volumétricos.

1.1.3 Metadatos

El término metadatos no tiene una definición única. Según la definición más difundida de metadatos es que son “datos sobre datos”. También hay muchas declaraciones como “informaciones sobre datos”, “datos sobre informaciones” e “informaciones sobre informaciones” (Cid, y otros, 2008).

Otra clase de definiciones trata de precisar el término como “descripciones estructuradas y opcionales que están disponibles de forma pública para ayudar a localizar objetos” o “datos estructurados y codificadas que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas” (Manuel, 2005). Esta clase surgió de la crítica de que las declaraciones más simples son tan difusas y generales que dificultarán la tarea de acordarse de estándares, pero estas definiciones no son muy comunes.

Los metadatos se clasifican usando tres criterios (Gómez, 2010):

- **Contenido.** Subdividir metadatos por su contenido es lo más común. Se puede separar los metadatos que describen el recurso mismo de los que describen el contenido del recurso. Es posible subdividir estos dos grupos más veces, por ejemplo para separar los metadatos que describen el sentido del contenido de los que describen la estructura del contenido o los que describen el recurso mismo de los que describen el ciclo vital del recurso.
- **Variabilidad.** Según la variabilidad se puede distinguir metadatos mutables e inmutables. Los inmutables no cambian, no importa qué parte del recurso se vea, por ejemplo el nombre de un fichero. Los mutables difieren de parte a parte, por ejemplo el contenido de un vídeo.
- **Función.** Los datos pueden ser parte de una de las tres capas de funciones: subsimbólicos, simbólicos o lógicos. Los datos subsimbólicos no contienen información sobre su significado. Los simbólicos describen datos subsimbólicos, es decir, añaden sentido. Los datos lógicos describen cómo los datos simbólicos pueden ser usados para deducir conclusiones lógicas, es decir, añaden comprensión.

1.1.4 Flota

Una flota es un conjunto de embarcaciones que tienen un destino común, realizan la misma actividad y normalmente son propiedad de una compañía ((Online, 2007); (Larousse, 2007)). Dentro de estas embarcaciones encontramos:

- Barcos mercantes de un país, de una compañía de navegación o de una línea marítima.
- Pueden existir flotas de guerra o flotas pesqueras, flotas de taxis, flotas de camiones, etc.

Cuando las embarcaciones comparten unas mismas características se dice que la flota es homogénea, y si son diferentes la flota es heterogénea (Rodríguez, 2007).

1.1.5 Control de flotas

El Sistema de Gestión y Control de Flota (SGCF) está constituido por todos los equipos, infraestructura, aplicativos informáticos y procesos que permiten realizar las actividades de planeación, programación y control de la operación, entendiendo por planeación y programación la especificación de las rutas, servicios y frecuencias del sistema; y por control, aquellas actividades que tienen como fin coordinar, vigilar, registrar y fiscalizar dicha operación, así como hacer seguimiento de los indicadores de servicio del sistema (Jurídicas, 2009).

El control de flotas es centrado en la localización, mensajería, telemetría e información de históricos con el de Gestión, relativo a la planificación de rutas, cálculo de costos y optimización de cargas. Permite la planificación de las rutas al detalle y la importación y exportación de datos (CyMSat, 2005).

1.1.6 Sistema de Posicionamiento Global o GPS

El GPS (*Global Positioning System*: sistema de posicionamiento global) o NAVSTAR-GPS (*NAVigation System and Ranging - Global Positioning System*) es un sistema global de navegación por satélite (GNSS) que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave,

con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión (Farré, 2010).

1.2 Funcionamiento de los sistemas de posicionamiento global

El GPS funciona mediante una red de 32 satélites (28 operativos y 4 de respaldo) en órbita sobre el globo, a 20 200 km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor localiza automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante "triangulación" (método de trilateración inversa), la cual se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o las coordenadas reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que llevan a bordo cada uno de los satélites ((FonoAmerica, 2009); (Ribeiro, 2008)).

1.2.1 Aplicaciones de sistemas de posicionamiento global

Actualmente, son múltiples los campos de aplicación de los sistemas de posicionamiento tanto como sistemas de ayuda a la navegación, como en la modelación del espacio atmosférico y terrestre o aplicaciones con requerimientos de alta precisión en la medida del tiempo. A continuación se detallan algunos de los campos donde se utilizan en la actualidad sistemas GPS (Ribeiro, 2008):

- Estudio de fenómenos atmosféricos.
- Localización y navegación en regiones inhóspitas.
- Modelos geológicos y topográficos.
- Ingeniería civil.
- Sistemas de alarma automática.

- Sincronización de señales.
- Sistemas de aviación civil.
- Navegación y control de flotas de vehículos.

1.3 Caracterización del proceso de control de flotas

Dentro de los sistemas de seguimiento y control de flotas se encuentra una amplia gama de sistemas de navegación y ayuda a la gestión de los transportes y su logística. Si bien los precios de los componentes necesarios para su construcción van bajando paulatinamente, la dificultad técnica para su desarrollo radica en el hecho de que este tipo de sistemas exigen al equipo de desarrollo conocimiento para la integración de diversas tecnologías: tarjetas con microprocesadores, integración de sensores, comunicaciones por radio terrestre o por satélite, comunicaciones entre redes de computadores, *software* de comunicaciones, sistemas de información geográfica, interfaces gráficas de usuario, acceso y gestión de bases de datos avanzadas, ingeniería del *software*, etc.

El proceso de control de flotas requiere una serie de servicios básicos comunes entre los que se encuentran: adquisición, análisis, visualización, comunicaciones, etc. (Farré, 2010).

1.4 Análisis de otras soluciones existentes

El avance de las tecnologías de la información y de las comunicaciones está permitiendo el desarrollo de sistemas a precios muy asumibles por una creciente gama de clientes institucionales, industriales y privados, con unas utilidades que hace muy pocos años solo podían ser desarrollados por empresas líderes en tecnología y comprados por clientes con grandes presupuestos. Dentro de este tipo de productos se encuentra una amplia gama de sistemas de navegación y ayuda a la gestión de los transportes y su logística.

Las aplicaciones de seguimiento de flotas de vehículos son un ejemplo prototípico en el que, un desarrollo basado en componentes, permite abordar las necesidades de una alta reusabilidad, interoperabilidad y escalabilidad del *software* desarrollado con el objeto de acelerar su proceso de desarrollo.

1.4.1 OODISMAL (Sistema de Información Distribuida Orientada a Objeto para la Localización Automática de Móviles)

OODISMAL está compuesto por un conjunto de componentes distribuidos en una red de área local(Ver Figura 5), que interoperan entre sí utilizando la infraestructura CORBA (Group, 1995), y tienen como propósito la adquisición, almacenamiento, procesamiento y visualización en tiempo real en mapas digitales de datos de localización provenientes de móviles que incorporen un dispositivo GPS (Muro, 2006).

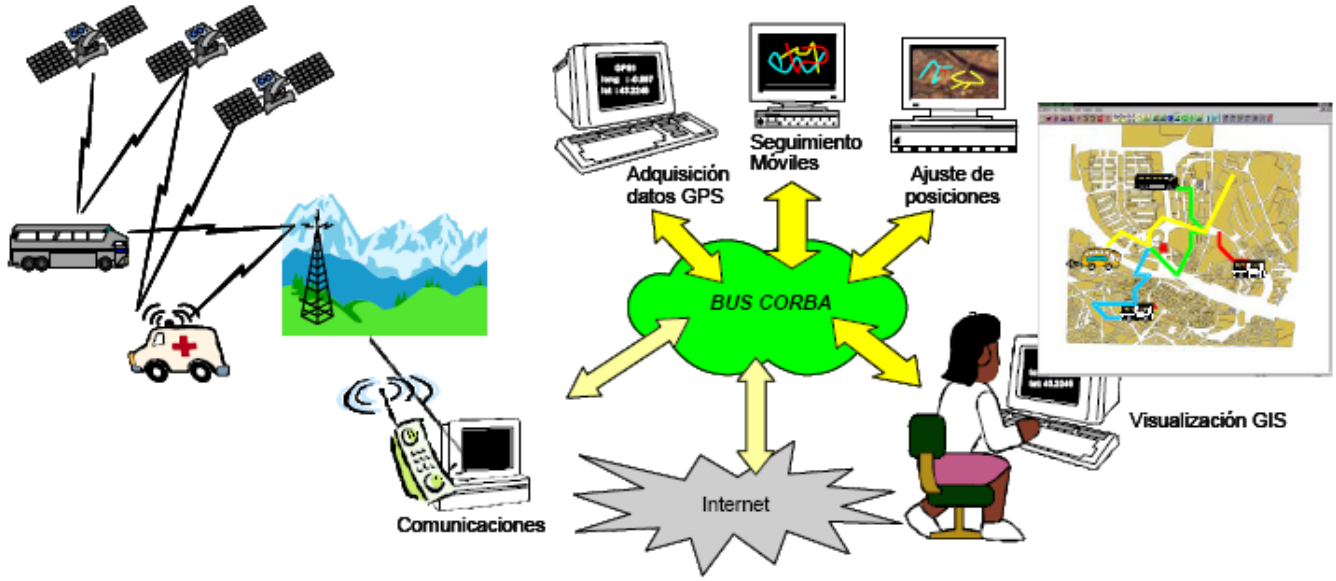


Figura 5: Arquitectura de OODISMAL

Los servicios básicos que presenta son:

- adquisición de datos de los móviles a través de enlaces de comunicación (actualmente se dispone del *software* para la adquisición de datos de GPS vía radiotelefonía en base al terminal T500 voz con opción GPS).
- utilidades para mejoras de precisión y de ajuste a rutas.
- análisis de rutas (relacionados con distancias, tiempos, velocidades, posiciones) y correlación entre rutas reales y previstas.

- gestión de persistencia.
- simulación de móviles para depurar otras aplicaciones y hacer demostraciones.
- interfaz gráfica de usuario para el acceso a los datos.
- visualización de datos en mapas digitales.

Estos servicios proporcionan tanto las utilidades para el funcionamiento en tiempo real como el basado en datos almacenados (Muro, 2006). En la figura 6 se muestra el aspecto de la interfaz de la aplicación de seguimiento de vehículos.

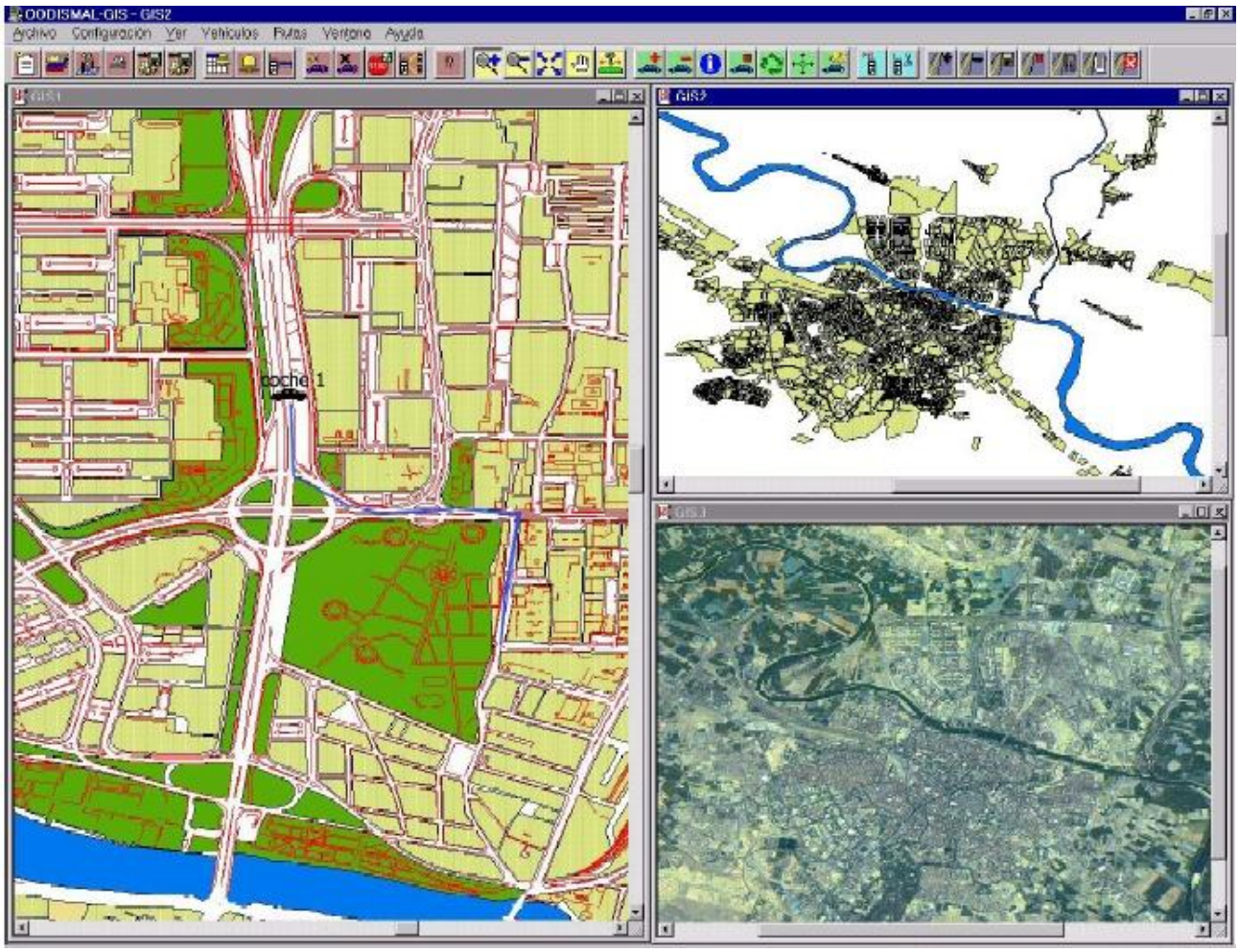


Figura 6: Aspecto de la interfaz de la aplicación de seguimiento de vehículos.

1.4.2 MovilWeb

MovilWeb es una aplicación de localización de vehículos basada en web para el seguimiento de móviles sobre cartografía vectorial y raster, diseñada para controlar flotas dentro de una arquitectura cliente-servidor.

Esta herramienta permite el monitoreo de móviles de manera remota sobre una red de comunicaciones, posibilitando reconstruir el comportamiento del vehículo en un determinado período de tiempo, reelaborando su trayectoria y analizando su velocidad, detenciones en destinos autorizados o no, etc., a través de la información almacenada en una base de datos histórica. Dota al usuario de un grupo de herramientas para el manejo de mapas, similar a las herramientas de los *software* profesionales para el manejo de Sistemas de Información Geográfica. Su arquitectura está orientada a servicios, en ella se produce la integración y encadenamiento de varios servicios de procesamiento y datos geoespaciales distribuidos sobre la web. Está implementada sobre plataformas de *software* libre, utiliza como gestor de base de datos PostgreSQL, *Business Intelligence and Reporting Tools* como herramienta para la generación de los reportes, GeoServer como servicio de mapas y todo se ejecuta sobre Apache Tomcat 6.0 como servidor para Internet. Las nuevas utilidades han sido implementadas sobre *Java* y se nutre de los servicios geoespaciales disponibles en la Infraestructura de Datos Espaciales de la República de Cuba (IDERC), como los servicios de imágenes satelitales y cartografía vectorial (Farré, y otros, 2010).

1.4.3 Conclusiones sobre las soluciones.

Las soluciones anteriores presentan diferentes funcionalidades que utilizan el control de flota en tiempo real. MovilWeb, las realiza también de manera diferencial, o sea, que logra obtener datos a partir de un archivo GPS con información sobre el recorrido obtenido a través de un dispositivo.

Se pueden reimplementar en la aplicación funcionalidades que ya están definidas en estas soluciones, entre las que se encuentran calcular la velocidad en diferentes tramos, mostrar la trayectoria de un vehículo en un mapa, etc. Se dice reimplementación de las funcionalidades y no reutilización de las mismas, ya que en el caso de OODISMAL, todos sus componentes son privativos, no expone su código fuente y está implementado para el uso en tiempo real. Además, MovilWeb como su nombre lo indica, es

una aplicación web, de ahí que sus funcionalidades tampoco se puedan reutilizar, debido a que la aplicación que se desea desarrollar es una aplicación de escritorio, que tiene como objetivo el mejoramiento de la capacidad gráfica visual y el tiempo de respuesta, o sea, una aplicación más rápida y con mayor personalización.

1.5 Conclusiones parciales.

Durante este capítulo se desarrolló un análisis de los elementos teóricos que sirven como base a la problemática y objetivos planteados en el presente trabajo. Para ello se realizó un estudio de los principales conceptos asociados al dominio del problema, caracterizando cada uno de ellos de forma tal que se logre comprender el entorno de la investigación y la teoría que la sustenta.

CAPÍTULO 2. TENDENCIAS Y TECNOLOGÍAS ACTUALES EN EL DESARROLLO DE *SOFTWARE*

2. Introducción.

La evolución de la informática ha sido posible por el perfeccionamiento de las herramientas y tecnologías para el desarrollo de *software*, que constituyen hoy, instrumentos fundamentales para garantizar la calidad de los procesos de desarrollo.

2.1 Metodologías para el desarrollo de *software*

El proceso de desarrollo de *software* es, sin lugar a dudas, una tarea complicada. Para aminorar este problema, en la actualidad se usan metodologías, logrando que el mismo sea un proceso disciplinado, predecible y eficiente. Una metodología de desarrollo es un proceso que engloba procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de *software* (Letelier, 2007). Existen dos corrientes de gestión del proyecto claramente diferenciadas una de otra, y que cada una de ellas ha comenzado a marcar territorio en el mercado.

2.1.1 Metodologías robustas

Las metodologías robustas son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también metodologías tradicionales o clásicas, donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

Proceso Unificado Racional (*Rational Unified Process* - RUP)

RUP es entre las metodologías una de las más significativas en la actualidad. Apareció en 1998 y fue creada por James Rumbaugh, Grady Booch e Ivar Jacobson para la *Rational Corporation*. Según sus

autores el proceso de desarrollo de *software* lo conforman el conjunto de actividades necesarias para transformar los requisitos funcionales de un usuario en un sistema de *software*.

Los aspectos más importantes que definen este proceso unificado son tres:

- Dirigido por casos de uso: Basándose en los casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo. Además, estos modelos se validan para que sean conformes a los casos de uso. Finalmente, los casos de uso también sirven para realizar las pruebas sobre los componentes desarrollados (Jacobson, y otros, 2000).

- Centrado en la arquitectura: En la arquitectura de la construcción, antes de construir un edificio este se contempla desde varios puntos de vista: estructura, conducciones eléctricas, fontanería, etc. Cada uno de estos aspectos está representado por un gráfico con su notación correspondiente. Siguiendo este ejemplo, el concepto de arquitectura de *software* incluye los aspectos estáticos y dinámicos más significativos del sistema (Jacobson, y otros, 2000).

- Iterativo e incremental: Todo sistema informático complejo supone un gran esfuerzo que puede durar desde varios meses hasta años, por lo tanto, lo más práctico es dividir un proyecto en varias fases. Actualmente se suele hablar de ciclos de vida en los que se realizan varios recorridos por todas las fases. Cada recorrido por las fases se denomina iteración en el proyecto en la que se realizan varios tipos de trabajo (denominados flujos). Además, cada iteración parte de la anterior incrementado o revisando la funcionalidad implementada (Jacobson, y otros, 2000).

RUP se divide en 4 fases (Ver Figura 7):

- Conceptualización (Concepción o Inicio): El objetivo de esta fase es determinar la visión del proyecto. Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema (Pressman, 1999).

- Elaboración: El objetivo es determinar la estructura óptima. Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la

arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido (Pressman, 1999).

- Construcción: El objetivo de esta fase es obtener la capacidad operacional inicial. Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene también uno o varios *release* del producto que han pasado las pruebas. Se ponen estos *release* a consideración de un subconjunto de usuarios (Pressman, 1999).

- Transición: El objetivo es obtener la primera versión del proyecto (*release*). Puede implicar reparación de errores (Pressman, 1999).

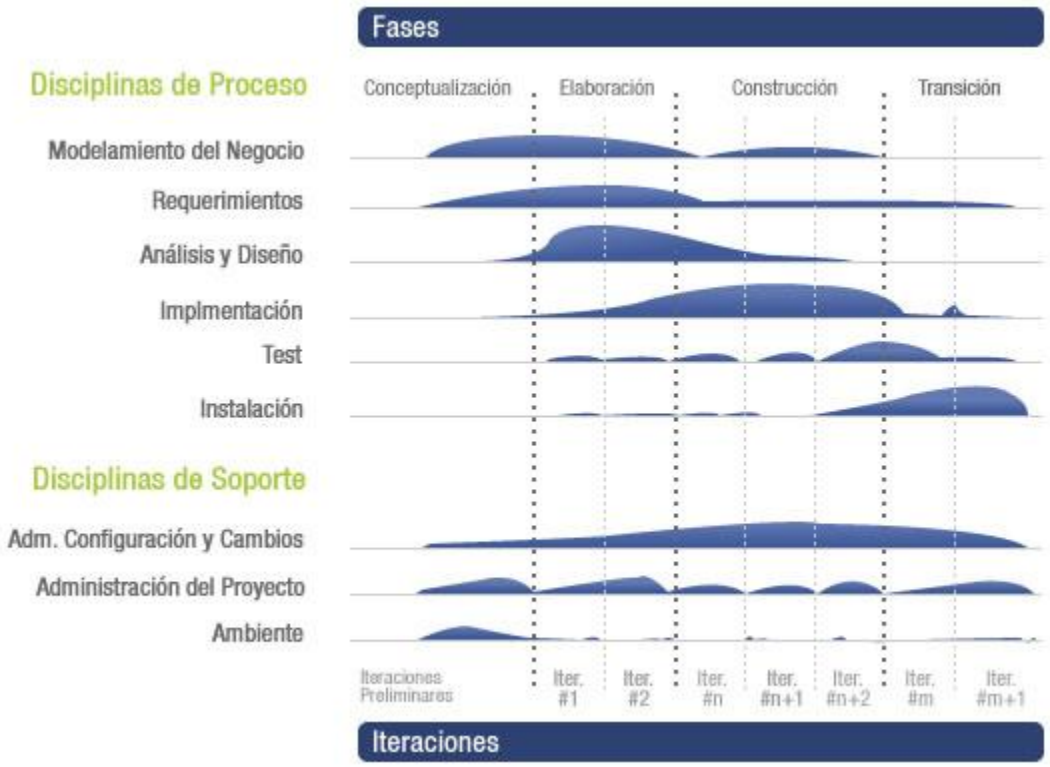


Figura 7: Imagen representativa de la estructura de RUP.

Metodología CASE

La metodología CASE (*Computer Aided Systems Engineering*) plantea una secuencia de etapas y proporciona para cada una de ellas su descripción, definición de objetivos y metas, productos, factores críticos de éxito, y la lista de tareas que conviene realizar (Río, 1995). Se basa en un análisis y desarrollo del tipo descendiente ("*top-down*") en que el ciclo de vida de un sistema se compone de las siguientes etapas:

1. Estrategia: Esta etapa tiene por objetivo lograr un entendimiento claro de las necesidades de la organización y del ambiente en que operará el sistema o sistemas a implantar. Con el fin de tener una visión desde los puntos de vista de la dirección corporativa, se analizan las diferentes funciones que realiza la organización y sus necesidades de información a todos los niveles. Durante esta etapa se realizan una serie de entrevistas con la dirección y los responsables de los departamentos, para realizar un primer modelado de los requerimientos del sistema de información adecuado a las necesidades de la organización. Posteriormente, se realiza una primera versión de la arquitectura del sistema. Los resultados de esta etapa son: un conjunto de modelos de la empresa**,** un conjunto de recomendaciones*,* y un plan acordado de desarrollo de los sistemas de información (Río, 1995)**.**

2. Análisis: Esta etapa toma y verifica los descubrimientos de la etapa de estrategia y expande estos en suficiente detalle para asegurar la precisión de los modelos de la empresa, posibilitando un fundamento sólido para el diseño dentro del alcance de la organización y tomando en cuenta sistemas existentes. Con el fin de obtener un refinamiento de los modelos, se realiza otra serie de entrevistas ya no a un nivel directivo como en la anterior, sino a un nivel operativo y técnico. Los modelos básicos de esta etapa son: el modelo de entidad-relación, el modelo funcional y la clasificación de las diversas funciones y subfunciones que fueron identificadas en el análisis. Además, se definen las restricciones que tendrá el sistema y la estrategia que se seguirá en la etapa de transición (Río, 1995).

3. Diseño: En esta etapa se toman los requerimientos y el modelado de la etapa de análisis y se determina la mejor manera de satisfacerlos, logrando niveles de servicios acordados, dados el

ambiente técnico y las decisiones previas en los niveles requeridos de automatización. El diseño final del sistema integra tres diseños, el de la base de datos, el de la aplicación y el de la red, además, se elaboran los planes de prueba y de transición y se realizan los diseños de los sistemas de auditoría y control, y el de respaldos y recuperación. Los resultados de esta etapa lo constituyen, la arquitectura del sistema, el diseño de la base de datos, la especificación de los programas y la especificación de los manuales de procedimientos (Río, 1995).

4.1 Construcción: A partir del diseño final generado en la anterior etapa, en esta se codificarán y probarán los nuevos programas usando herramientas apropiadas. Esta etapa involucra planeación, diseño de la estructura del sistema, codificación de abajo a arriba (prueba de unidades y enlaces), pruebas de arriba a abajo (prueba del sistema) y un enfoque disciplinado en la realización del trabajo y en el control de versiones del sistema y pruebas. Los resultados de esta etapa son los programas probados y la base de datos afinada (Río, 1995).

4.2 Documentación: Los productos fundamentales para el uso y el mantenimiento efectivo y eficiente de los sistemas programados son los manuales. En esta etapa se consideran el estilo de trabajo y las necesidades propias de los usuarios que utilizarán y mantendrán el sistema. Los manuales, resultados de esta etapa, se elaboran a partir de las especificaciones de diseño, de los programas realizados y del análisis del estilo de trabajo y nivel de competencia de los usuarios y operadores de los sistemas. Se realiza al mismo tiempo que la etapa de construcción (Río, 1995).

5. Transición: El desarrollo de un sistema no se termina con su programación; antes de su liberación para su uso, se debe prever un período de transición que deberá incluir la alimentación de las nuevas bases de datos, la capacitación de los usuarios y el desarrollo de pruebas. En esta metodología la transición conforma una de sus etapas y en ella se realizan todas las tareas necesarias para la implementación. Proporciona un período inicial de soporte al sistema. La transición debe llevarse a cabo con una interrupción mínima de la organización, y debe dejar a los usuarios confiados y listos para explotar el nuevo sistema (Río, 1995).

6. Producción: Finalmente, en esta etapa se asegura que el sistema funcione correctamente en la mayoría de los casos, y con intervención mínima de los administradores del sistema. Para esto se realizan nuevas pruebas, se reevalúan los resultados y se hacen refinamientos del sistema. Los cambios necesarios deberán ser introducidos sin afectar a los usuarios y deberá conseguirse la máxima confianza de los mismos. El resultado de esta etapa es un sistema listo para su operación (Río, 1995).

2.1.2 Metodologías ágiles

Las metodologías ágiles de desarrollo están especialmente indicadas en proyectos con requisitos poco definidos o cambiantes. Estas metodologías se aplican bien en equipos pequeños que resuelven problemas concretos, lo que no está reñido con su aplicación en el desarrollo de grandes sistemas, ya que una correcta modularización de los mismos es fundamental para su exitosa implantación (Carlos, 2007).

Programación Extrema (*eXtreme Programming*, XP)

XP es la primera metodología ágil y la que le dio conciencia al movimiento actual de metodologías ágiles. Fue desarrollada por Kent Beck (Carlos, 2007). Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP está guiada por una rápida programación y se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico ((Canós, y otros, 2009); (Carlos, 2007); (Rojas, 2008)). A continuación se presentan las características de esta metodología agrupadas en los cuatro apartados siguientes:

- Las historias de usuarios: es la técnica utilizada para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias

de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración (Carlos, 2007).

- Roles XP:

  Los roles de acuerdo con la propuesta original de Beck son (Carlos, 2007):

  a. Programador: El programador escribe las pruebas unitarias y produce el código del sistema.

  b. Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

  c. Encargado de pruebas: Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

  d. Encargado de seguimiento: Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

  e. Entrenador: Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

  f. Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

g. Gestor: Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

- Proceso XP: En todas las iteraciones del ciclo de desarrollo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el *software* o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración. El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (Carlos, 2007).

- Prácticas XP: La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del *software* e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo. Si bien XP es la metodología ágil de más renombre en la actualidad, se diferencia de las demás metodologías que forman este grupo en un aspecto en particular: el alto nivel de disciplina de las personas que participan en el proyecto (Carlos, 2007).

SCRUM

SCRUM define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto a los procesos definidos (cascada, espiral, prototipos, etc.) mediante la aceptación de la naturaleza caótica del desarrollo de *software*, y la utilización de prácticas tendientes a manejar la impredictibilidad y el riesgo a niveles aceptables. El mismo surge en 1986, de un artículo titulado "*The New Product Development Game*" de Hirotaka Takeuchi e Ikujiro Nonaka, que introducía las mejores prácticas más utilizadas en 10 compañías japonesas altamente innovadoras. A partir de ahí y tomando referencias al juego de rugby, Ken Scwaber y Jeff Sutherland formalizan el proceso conocido como SCRUM en el año 1995 (Carlos, 2007).

Esta metodología, enfatiza prácticas y valores del desarrollo de proyectos por sobre las demás disciplinas del desarrollo. Al principio del proyecto se definen todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir. Los mismos estarán especificados de acuerdo a las convenciones de la organización ya sea mediante: características, casos de uso, diagramas de flujo de datos, incidentes, tareas, etc. ((Carlos, 2007); (Rojas, 2008)).

Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos: el desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días y el resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente; la segunda característica importante de SCRUM son las reuniones a lo largo del proyecto, entre ellas destaca una reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración (Canós, y otros, 2009).

La intención de SCRUM es la de maximizar la realimentación sobre el desarrollo pudiendo corregir problemas y mitigar riesgos de forma temprana. Su uso se está extendiendo cada vez más dentro de la comunidad de metodologías ágiles, siendo combinado con otras - como XP - para completar sus carencias.

El ciclo de vida de SCRUM es el siguiente (Ver Figura 8) ((Carlos, 2007); (Rojas, 2008)):

- Pre-Juego: Planeamiento. El propósito es establecer la visión, definir expectativas y asegurarse la financiación. Las actividades son la escritura de la visión, el presupuesto, el registro de acumulación o retraso del producto inicial y los *ítems* estimados, así como la arquitectura de alto nivel, el diseño exploratorio y los prototipos. El registro de acumulación es de alto nivel de abstracción.

- Pre-Juego: Montaje. El propósito es identificar más requerimientos y priorizar las tareas para la primera iteración. Las actividades son planificación, diseño exploratorio y prototipos.

- Juego o Desarrollo. El propósito es implementar un sistema listo para entrega en una serie de iteraciones de treinta días llamadas “corridas”. Las actividades son un encuentro de planeamiento

de corridas en cada iteración y la definición del registro de acumulación de corridas y estimados, además de sus encuentros diarios.

- Pos-Juego: Liberación. El propósito es el despliegue operacional. Las actividades, documentación, entrenamiento, mercadeo y venta.

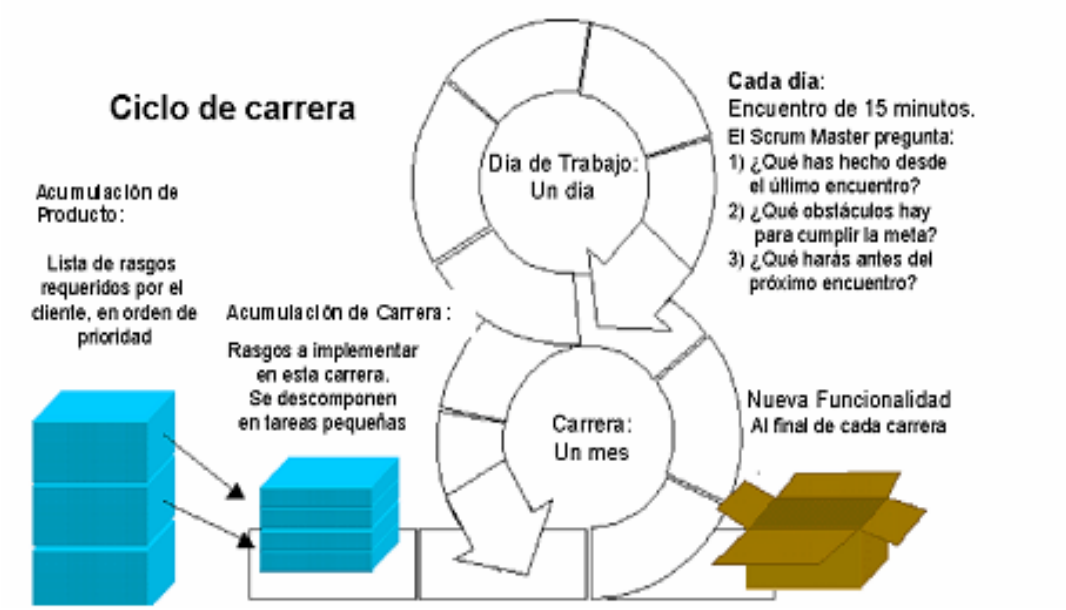


Figura 8: Ciclo de vida de SCRUM.

Método de Desarrollo de Sistemas Dinámicos (*Dynamic Systems Development Method*, DSDM)

Define el marco para desarrollar un proceso de producción de *software*. Propone cinco fases: estudio de viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente, la implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases (Letelier, 2007).

DSDM posee nueve principios esenciales para su implantación, ignorar uno solo de estos principios rompería con la filosofía de la metodología e incrementaría significativamente los riesgos del proyecto (Carlos, 2007):

- Imperativa participación del usuario activo: Este principio es considerado como el más importante porque la participación del usuario propicia la reducción de errores en términos de la percepción del mismo, lo que se traduce en la reducción del costo de errores. En vez de trabajar con una gran cantidad de usuarios, la guía de DSDM recomienda trabajar con un grupo pequeño continuamente.

- El equipo debe tener potestad para tomar decisiones: En aras de proceder de una manera más ágil debe evitarse la fricción de la comunicación entre los miembros del equipo de desarrollo y los directivos. Pedir autorización para acceder a modestos recursos o para hacer un simple cambio de requisitos retrasaría considerablemente a un proyecto.

- Centrarse en entregas continuas: Frecuentes entregas de resultados garantizan el descubrimiento temprano de errores, lo que permite su fácil corrección. Esto se aplica tanto al código elaborado por los programadores como a los documentos y diagramas elaborados por los analistas.

- Aptitud para el negocio es el criterio para entregas aceptables: El principal objetivo de DSDM es entregar productos *software* que satisfagan las necesidades del negocio y puedan ser enriquecidas con otras funcionalidades en futuras iteraciones. DSDM sugiere cumplir con las necesidades del negocio en primer lugar, y adquirir cajas de tiempo para refactorización y otras actividades conexas en una iteración posterior. Incluso en un proyecto DSDM es fundamental identificar las cuestiones claves, que requieren de un diseño robusto. Las metodologías ágiles requieren una buena comprensión de los patrones y la arquitectura; dependiendo mucho más de los primeros que los métodos de desarrollo tradicionales, porque es más probable tomar una decisión que ocasione la prohibición de entregas de dos o tres iteraciones en un futuro.

- Es obligatorio el desarrollo iterativo e incremental: Con el objetivo de mantener la complejidad de los proyectos, es necesario descomponerlos en pequeños paquetes de características; añadiendo funcionalidades a cada liberación hasta completar con todos los requisitos del negocio. Este principio está sujeto a la aceptación de que cualquier *software* puede estar sujeto a cambios.

- Todos los cambios realizados durante el desarrollo deben ser reversibles: Ser sensible al cambio exige que la configuración del sistema esté cambiando durante el desarrollo. Desde que DSDM aconsejó iterar pequeños incrementos, la pérdida total de trabajo es muy limitada.

- Línea base de requisitos a alto nivel: Para limitar el grado de libertad concerniente a los requisitos que pueden ser modificados durante el proceso de desarrollo, se establecen algunos de alto nivel. Esta línea base que puede ser interpretada como congelamiento de requisitos es acordada durante la fase de estudio del proceso de negocio.

- Pruebas integradas en todo el ciclo de vida: Muchos métodos de desarrollo claman por hacer las pruebas a partir de las fases de diseño o implementación. DSDM requiere pruebas en fases tempranas del proceso de desarrollo. Las pruebas incluyen arreglar los documentos por un grupo de control o técnicas similares.

- Enfoque colaborativo y cooperativo: Evitar la separación y fomentar la colaboración del personal técnico y el personal del negocio en un proyecto es crucial en el éxito de los proyectos DSDM. Sin un clima de confianza y honestidad, es difícil reunir los requisitos y más tarde obtener retroalimentación de los productos resultantes.

Agile Unified Process (AUP, Proceso Unificado Ágil o RUP Ágil)

AUP es una versión simplificada del Proceso Unificado de *Rational* (RUP) de IBM, pero se basa en disciplinas y entregables incrementales en el tiempo. En él se describe una forma sencilla, es fácil de entender en el enfoque de desarrollo de *software* de aplicaciones empresariales utilizando técnicas ágiles, conceptos y aún así, se mantiene fiel a RUP. El AUP aplica técnicas ágiles incluyendo el desarrollo de pruebas, gestión del cambio ágil y refactorización de base de datos para mejorar su productividad (Álvarez, y otros, 2009).

La disciplina de modelo de negocio abarca el modelado de RUP, requisitos y disciplinas de análisis y diseño. El modelo es una parte importante de la política de uso aceptable, pero no domina el proceso que desea permanecer ágil mediante la creación de modelos y documentos que son lo suficientemente

buenos. En segundo lugar, la configuración y la disciplina de Gestión del Cambio es ahora la disciplina de Gestión de la Configuración.

Las disciplinas que se llevan a cabo de manera iterativa por AUP son (Álvarez, y otros, 2009):

- Modelo: El objetivo de esta disciplina es entender el negocio de la organización, el dominio del problema del que se ocupa el proyecto y determinar una solución viable para hacer frente al dominio del problema.

- Implementación: El objetivo de esta disciplina es transformar el modelo en el código ejecutable para llevar a cabo un nivel básico de las pruebas.

- Prueba: El objetivo de esta disciplina consiste en realizar una evaluación objetiva para asegurar la calidad. Esto incluye encontrar defectos, validar que el sistema funcione como está previsto y verificar que se cumplan los requisitos.

- Gestión de la configuración: El objetivo de esta disciplina es administrar el acceso a los artefactos del proyecto. Esto incluye no solo el seguimiento de versiones de los artefactos a través del tiempo, sino también el control y la gestión de los cambios a los mismos.

- Gestión de proyectos: El objetivo de esta disciplina es dirigir las actividades que se llevan a cabo en el proyecto. Esto incluye la gestión de riesgos, la dirección de personas (la asignación de tareas, seguimiento de los progresos) y coordinar con la gente y los sistemas fuera del alcance del proyecto para asegurarse de que se entregue a tiempo y dentro del presupuesto.

- Medio ambiente: El objetivo de esta disciplina es apoyar el resto de los esfuerzos para garantizar la orientación adecuada del proceso y que las herramientas (*hardware*, *software*) estén disponibles para el equipo según sea necesario.

AUP presenta cuatro fases (Ver Figura 9) (Carlos, 2007):

- Creación: Identifica el alcance inicial del proyecto, una arquitectura potencial, obtiene la financiación inicial del proyecto y la aceptación de las partes interesadas.

- Elaboración: Prueba la arquitectura del sistema.

- Construcción: Construir el *software*.

- Transición: Valida y despliega el sistema en su entorno de producción.

AUP se centra en actividades de alto valor. Los productos AUP son fácilmente tolerables a través de cualquier herramienta de edición de HTML común (Álvarez, y otros, 2009).

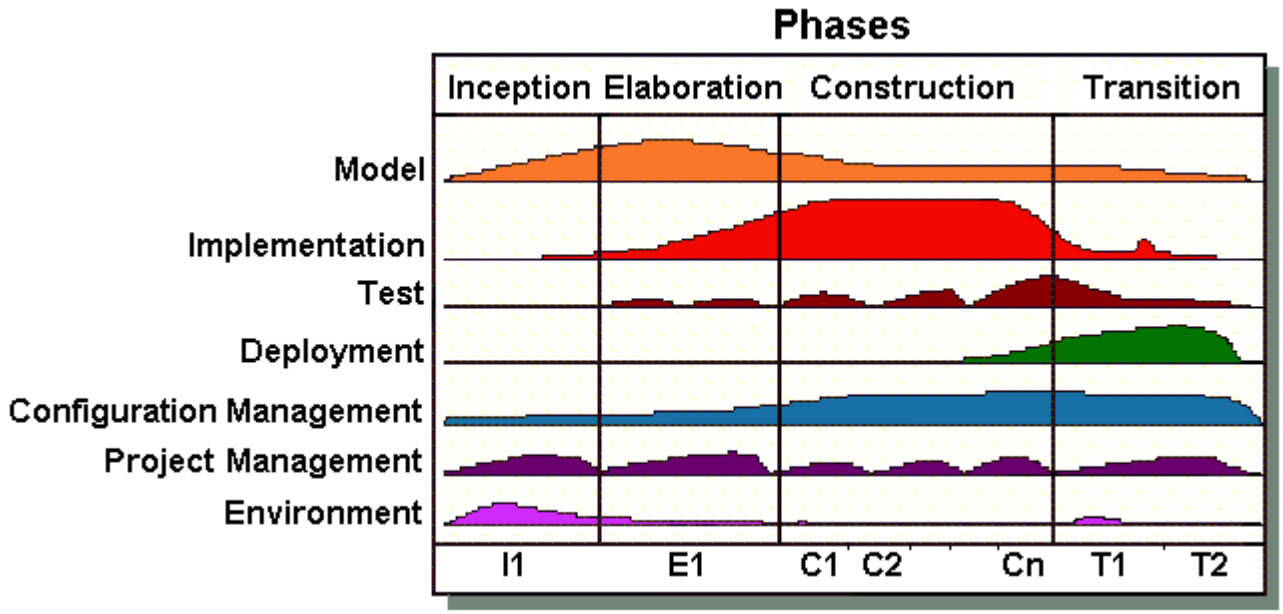


Figura 9: Esquema de trabajo de AUP.

2.1.3 Justificación de la metodología seleccionada

La metodología que se elige para llevar a cabo el desarrollo de la aplicación es AUP. La misma, contiene en sí la fortaleza de RUP y la flexibilidad de las metodologías ágiles. Debido a esto, se hace posible que

en cada iteración del desarrollo del *software* se mejore la funcionalidad, se solucione el problema de no poder retroceder a medida que se avanza en las fases y se proporcione un hilo conductor permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo. Además, ofrece una alta capacidad organizativa para grupos de trabajo.

AUP pone de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan. Esta flexibilidad es una ventaja competitiva porque estar preparados para el cambio significa reducir su costo. Permite retrasar las decisiones tanto como sea posible de manera responsable y esto es ventajoso tanto para el cliente como para la empresa. Reduce el número de decisiones de alta inversión que se toman y de cambios necesarios en el proyecto.

Además, permite realizar una planificación adaptativa que consiste en tomar decisiones a lo largo del proyecto, transformando el proyecto en un conjunto de proyectos pequeños. Esto permite tener el *software* disponible para los clientes a corto plazo (Álvarez, y otros, 2009).

2.2 Lenguaje Unificado de Modelado (*Unified Modeling Language,* UML)

Un modelo es una simplificación de la realidad. El objetivo del modelado de un sistema es capturar las partes esenciales del sistema. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica. Esto se conoce como modelado visual (Orallo, 2006).

Al principio de la década de 1990, James Rumbaught, Grady Booch e Ivar Jacobson comenzaron a trabajar en un "método unificado" que combinaría las mejores características de cada uno de sus métodos individuales y adoptaría características adicionales que propusieran otros expertos en el campo Orientado a Objetos. El resultado fue el lenguaje de modelado unificado (UML) que contiene una notación robusta para el modelado y el desarrollo de sistemas orientados a objetos (Pressman, 1999).

UML se centra en la representación gráfica de un sistema y permite la especificación, visualización, construcción y documentación de elementos de la Ingeniería del Software (Orallo, 2006). Está formado por elementos (abstracciones de cosas reales o ficticias), relaciones (relacionan los elementos entre sí) y diagramas (colecciones de elementos con sus relaciones) (Orallo, 2006). Estos últimos, muestran diferentes aspectos de lo que se quiere representar. Los diagramas de estructura enfatizan en los

elementos que deben existir en el sistema modelado y los de comportamiento enfatizan en lo que debe suceder también en el mismo. Los diagramas de interacción por su parte, son un subtipo de diagramas de comportamiento, que enfatizan sobre el flujo de control y de datos entre los elementos del sistema modelado.

Las ventajas que proporciona UML como lenguaje de modelado han hecho posible que se seleccione el mismo para modelar la aplicación, debido a que brinda mayor rigor en la especificación y permite realizar una verificación y validación del modelo realizado. Con UML se pueden automatizar determinados procesos, además de que permite generar código a partir de los modelos y a la inversa, de aquí que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño de la estructura de un proyecto.

2.2.1 Herramientas CASE

Hoy en día, muchas empresas se han extendido a la adquisición de herramientas de Ingeniería Asistida por Computadora (*Computer-Aided Systems Engineering*, CASE), con el fin de automatizar los aspectos claves de todo el proceso de desarrollo de un sistema e incrementar su posición en el mercado competitivo.

Una herramienta CASE es un producto computacional enfocado a apoyar una o más técnicas dentro de un método de desarrollo de *software* (Marcos, 2005).

Las herramientas CASE suponen una forma de abstracción del engorroso código fuente, a un nivel donde la arquitectura y el diseño se hacen más aparentes y fáciles de entender y modificar. Cuanto mayor es un proyecto, más importante es el uso de tecnología CASE.

También se puede definir a las herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. La gestión debe ser posible desde un nivel de abstracción alto, es decir, mirar una representación de un diseño y comprenderlo. Estas herramientas se acoplan con las metodologías para dar una forma de representar sistemas. Proporcionan un conjunto de herramientas semi-automatizadas y automatizadas que están desarrollando una cultura de ingeniería nueva para

muchas empresas. Uno de los objetivos más importantes es conseguir, a largo plazo, la generación automática de programas desde una especificación a nivel de diseño. Se han desarrollado como una de las soluciones para afrontar los problemas de calidad de *software* pobre y documentación inadecuada.

Rational Rose Enterprise Edition

Rational Rose Enterprise es el producto más completo de la familia *Rational Rose* de IBM. Todos los productos *Rational Rose*, como el *Rose Data Modeler* para el diseño visual de bases de datos, el *Rose Modeler*, para el análisis y la arquitectura de *software*, entre otros, tienen soporte para modelado UML.

Rational Rose Enterprise Edition es una herramienta *software* para el modelado visual mediante UML de sistemas *software* (Blanco, y otros, 2006) que soporta la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® y Visual Basic®. Como todos los demás productos *Rational Rose*, proporciona un lenguaje común de modelado para el equipo que facilita la creación de *software* de calidad más rápidamente (IBM, 2009).

Permite la modelación de los procesos del negocio y del sistema, además, resulta de gran utilidad para los desarrolladores de proyectos, pues cubre todo el ciclo de vida del mismo desde su fase inicial hasta su culminación. Esta herramienta posibilita establecer una trazabilidad entre los modelos y el código ejecutable. Propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando las vistas estática, dinámica, lógica y física de los modelos del sistema. Permite realizar y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de *software*. Cada analista, desarrollador o diseñador puede usar *Rational Rose* para definir y comunicar el negocio, el diseño y la arquitectura de la aplicación que se está desarrollando. Es una completa solución para mostrar de forma gráfica el análisis de los procesos del negocio y los requisitos del sistema. Permite que hayan varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo. Además, permite visualizar, entender y refinar los requisitos y la arquitectura antes del enfrentamiento al código, lo cual permite evitar esfuerzos innecesarios en el ciclo de desarrollo en caso de detectarse errores (IBM, 2009).

Visual Paradigm para UML

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El *software* de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (Alieri, 2009). La versión 6.4 incluye un paquete de aplicaciones ofreciendo varias herramientas que facilitan el desarrollo de proyectos de cualquier dimensión. Las principales características son (Sánchez, 2007):

- Soporte de UML versión 2.1.
- Modelado colaborativo con CVS y Subversión.
- Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas EJB - Visualización de sistemas EJB.
- Generación de código y despliegue de EJB´s - Generación de *beans* para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos.

- Soporte ORM - Generación de objetos *Java* desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de bases de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XMI.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.
- Integración con Visio - Dibujo de diagramas UML con plantillas (*stencils*) de MS Visio.

2.2.2 Justificación de la herramienta CASE utilizada: *Visual Paradigm*

Para el modelado de los artefactos y diagramas generados a lo largo del ciclo de vida del proyecto se decidió emplear *Visual Paradigm* en su versión 6.4, pues su uso está muy estandarizado a nivel mundial y constituye una herramienta multiplataforma muy madura y acabada. *Visual Paradigm* para UML es una herramienta profesional fácil de utilizar, que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad a un menor coste. Permite dibujar todos los tipos de diagramas de clases, permite la realización de ingeniería tanto directa como inversa, generar el código desde diagramas y generar la documentación automáticamente en varios formatos como web o Formato de Documento Portable (pdf). Permite el control de versiones. Además, la herramienta es colaborativa, o lo que es lo mismo, soporta múltiples usuarios trabajando sobre el mismo proyecto. No presenta ningún problema con su licencia para la elaboración de *software* comercial. Proporciona también abundantes tutoriales de UML y demostraciones interactivas de UML.

2.3 Justificación del lenguaje a utilizar: C++

Es un lenguaje orientado a objeto basado en C. Proporciona nuevas características útiles en diversos contextos como la sobrecarga de operadores. Su expresividad es elevada, pues la sintaxis de clases y objetos permite manipular convenientemente diversas estructuras de datos y operaciones. Las excepciones permiten procesar de un modo claro los casos de error (EIRL, 2008). Es un lenguaje muy difundido y popular. Puede ser usado para resolver diferentes tipos de problemas, desde los más simples hasta los más complejos y su código se puede compilar en diversas plataformas.

En C++, una clase es un tipo definido por el usuario, y sintácticamente, es una estructura con funciones de miembro. C++ incorpora una característica que, si bien no es específica de orientación a objetos, si produce un aumento de eficiencia en la llamada a funciones. Esta característica es la posibilidad de escribir funciones en línea (*inline*). *Inline* es una petición al compilador para sustituir la llamada a la función directamente por su código, es decir, genera código en línea.

Utiliza constructores y destructores para crear o destruir las instancias. Además, permite la conversión de tipo implícito, darle nombres a las funciones, tener argumentos de función por defecto y pasarlos por referencia (Kansas, 2007).

Debido a las ventajas y funcionalidades anteriormente mencionadas se propone como lenguaje de programación para el desarrollo de la aplicación el lenguaje C++. Producto de las características que manifiesta la aplicación que se llevará a cabo, C++ cumple con todos los objetivos perseguidos por los desarrolladores. Además, es el lenguaje en el cual fue desarrollada la plataforma GeoQ a la cual será añadida la aplicación como módulo, por lo que brindará reutilización de código. Al ser un leguaje liberado no presenta problema con las licencias a la hora de la distribución.

2.4 Justificación para elección del Entorno de Desarrollo Integrado (IDE): Qt

Qt posee un marco de trabajo multiplataforma para el desarrollo de aplicaciones e interfaces de usuario. Incluye librerías multiplataforma, herramientas de desarrollo integradas y un IDE multiplataforma. Presenta

un gran número de características que lo hacen altamente aplicable a cualquier aplicación de escritorio sin tener que utilizar librerías externas, entre las que se destacan:

- Es una intuitiva librería de C++.
- Garantiza la portabilidad entre sistemas operativos embebidos y de escritorio.
- Presenta herramientas de desarrollo integrado con IDE multiplataforma y un alto desempeño en dispositivos embebidos.
- Entre las posibilidades que brinda este marco de trabajo están la del trabajo con hilos independientemente del sistema operativo.
- Contiene un módulo para el trabajo con protocolos de red.
- Posee soporte para aplicaciones orientadas a componentes.
- Trabajo con los gestores de bases de datos más conocidos además de poder implementar soporte para otros.

Hasta el momento ha sido liberado bajo dos licencias, la LGPL y la comercial. La licencia comercial permite crear aplicaciones propietarias y la disponibilidad de soporte técnico, mientras que la licencia LGPL, además de permitir lo planteado anteriormente, también implica que los cambios al código de Qt deben de ser compartidos y debe de mantener las opciones de la licencia de distribución abierta.

Para desarrollar esta aplicación se ha elegido trabajar con la versión LGPL de Qt, puesto que provee un mayor intercambio con la comunidad de clientes y desarrolladores. Además, con ella se evita pagar una licencia de desarrollo.

2.5 Sistemas Gestores de Bases de Datos (SGBD)

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos.

Oracle

Oracle es básicamente una herramienta cliente/servidor para la gestión de bases de datos. Constituye un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que solo se vea en empresas muy grandes y multinacionales. Este SGBD está considerado como uno de los más fuertes y robustos para la elaboración de grandes sistemas donde el principal objetivo sea el manejo fácil y fiable de la información. Hoy Oracle tiene implementaciones sobre plataformas libres pero con los mismos derechos de licencias, que imposibilitan su implantación en soportes de pocos ingresos monetarios (Oracle, 2009).

MySQL

El *software* MySQL proporciona un servidor de base de datos SQL (*Structured Query Language)* muy rápido, multihilo, multiusuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos con alta carga de trabajo, así como para integrarse en *software* para ser distribuido. MySQL es una marca registrada de MySQLAB. Los usuarios pueden elegir entre usar el *software* MySQL como un producto de código abierto bajo los términos de la licencia GNU o pueden adquirir una licencia comercial estándar de MySQL AB (Company, 2010).

PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD12 y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilo

para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

Sus características técnicas lo hacen una de las bases de datos más potentes y robustas del mercado. Durante 15 años, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante el desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema (PostgreSQL.org, 2009).

2.5.1 Justificación de la elección del Gestor de Base de Datos: PostgreSQL

Los sistemas gestores de bases de datos por lo general presentan especificaciones muy peculiares de cada uno, por lo que para la implementación de un sistema con manejo de grandes cantidades de datos, la visualización de la información debe ser de fácil acceso. Para la utilización en el desarrollo de la base de datos se selecciona PostgreSQL, por estar licenciado bajo BSD y utilizar control de versionado concurrente y soporte para la implementación de consultas complejas. Con este SGBD se podrá mantener una base de datos actualizada, confiable y configurable, así como un alto nivel de integración con las demás aplicaciones que conforman el producto.

2.6 Conclusiones parciales

En este capítulo se desarrolló un análisis de las principales metodologías de desarrollo de *software* que se utilizan en la actualidad, lo que posibilitó la elección de la metodología AUP que guiará todo el proceso para el desarrollo de la aplicación garantizando de esta manera la adaptación al cambio y mayor agilidad en la construcción de la misma. Además, se seleccionó para el aprovechamiento de las ventajas que brindan el lenguaje de programación C++ utilizando como IDE Qt, la herramienta CASE Visual Paradigm, el lenguaje de modelado UML y el gestor de base de datos PostgresSQL.

CAPÍTULO 3. PRESENTACIÓN DE LA SOLUCIÓN PROPUESTA

3. Introducción

En este capítulo se da a conocer la propuesta de solución y en aras de describirla se realiza el modelo de dominio y la especificación de los requisitos funcionales y no funcionales que debe presentar el sistema a construir. Además, se determinan los casos de uso del sistema y los actores que interactuarán con este, elaborándose una descripción de cada uno de los mismos.

3.1 Modelo de dominio

El modelo de dominio es una representación visual estática del entorno real objeto del proyecto, o lo que es lo mismo, un diagrama con los objetos que existen (reales) relacionados con el proyecto que vamos a acometer y las relaciones que hay entre ellos (Ver Figura 10). Tiene como objetivo ayudar a comprender los conceptos que utilizan los usuarios, los conceptos con los que trabajan y con los que deberá trabajar la aplicación (Jacobson, 2000).

3.1.1 Diagrama de clases del Modelo de Dominio

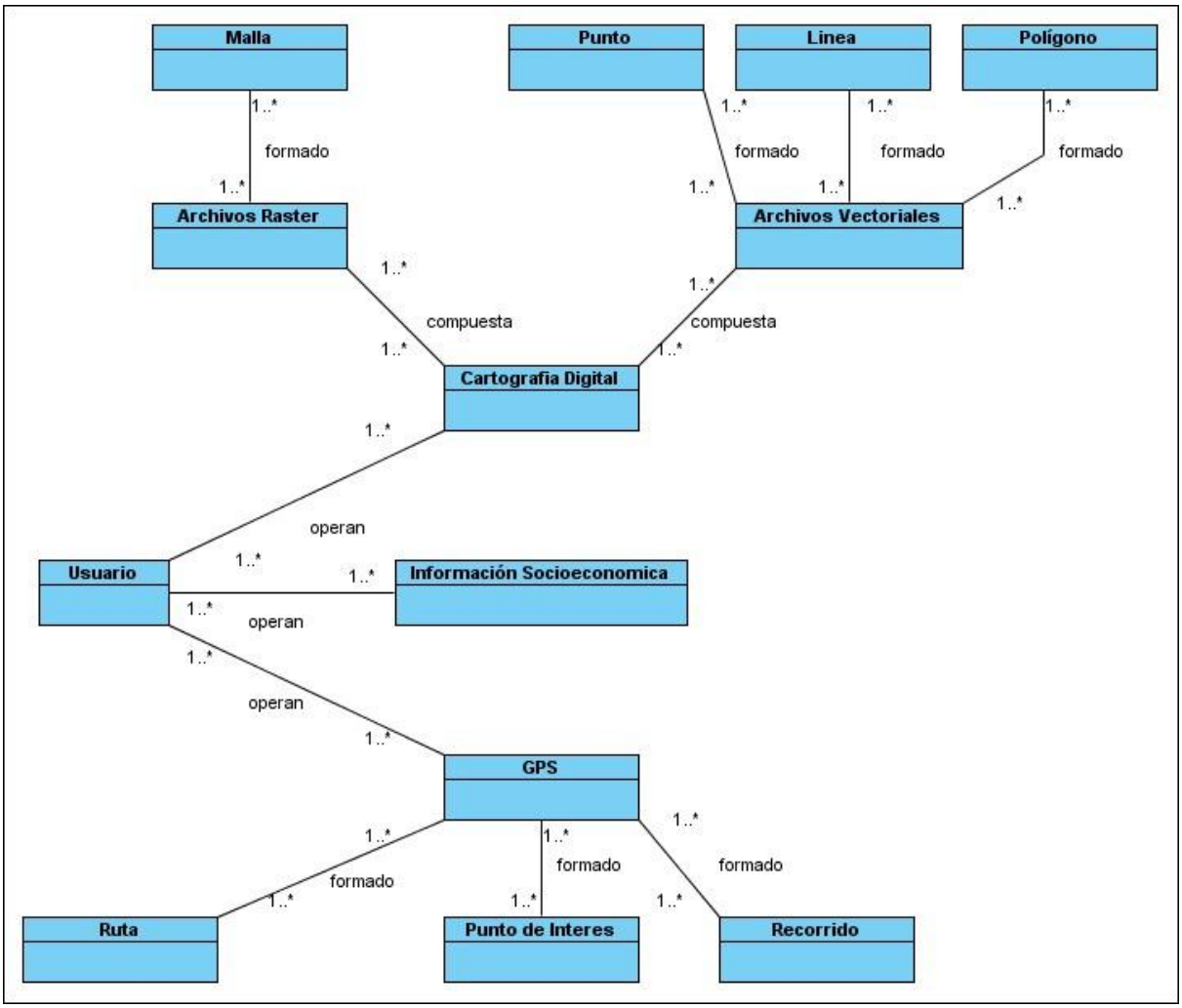


Figura 10: Diagrama de Modelo del Dominio.

3.1.2 Glosario de términos del dominio

- **Usuario:** puede ser tanto una persona como una computadora o un *software* que tiene una interacción directa con el producto.

- **Datos espaciales:** Contienen las ubicaciones y formas de características cartográficas. Dentro de su contexto, almacenan informaciones sobre la localización y las formas de un objeto geográfico y las relaciones entre ellos, normalmente con coordenadas y topología.

- **Modelo raster:** Es el modelo de datos complementario al modelo vectorial presentado anteriormente. Tiene como principal característica el llevar a cabo una representación "discreta" del mundo real, empleando una malla de rejillas regulares.

- **Malla:** Es una estructura donde en cada una de las celdas o píxeles se almacenan valores de un determinado aspecto o variable del mundo real.

- **Modelo vectorial:** Utiliza vectores definidos por pares de coordenadas relativas a algún sistema cartográfico. Con un par de coordenadas y su altitud gestionan un punto (un vértice geodésico), con dos puntos generan una línea y con una agrupación de líneas forman polígonos. En general, el modelo de datos vectorial es adecuado cuando se trabaja con objetos geográficos con límites bien establecidos.

- **Línea**: Las líneas están compuestas por puntos con sus correspondientes coordenadas X, Y y un valor Z global para el atributo de la línea.

- **Punto**: Los puntos son la estructura vectorial más simple, cuya información solo requiere una posición X, Y y un valor Z opcional para el atributo.

- **Polígonos**: Los polígonos bidimensionales se utilizan para representar elementos geográficos que cubren un área particular de la superficie de la tierra. Estas entidades pueden representar lagos, límites de parques naturales, edificios, provincias o los usos del suelo, entre otros. Los polígonos transmiten la mayor cantidad de información en archivos con datos vectoriales y en ellos se pueden medir el perímetro y el área.

- **Punto de interés**: Los GPS son capaces de almacenar una posición en su memoria, un lugar determinado. Cada una de esas posiciones almacenadas es lo que llamamos un punto de interés o

waypoint. Un *waypoint* incluye las coordenadas de la posición que define en qué parte del mundo está localizado, el nombre, la altitud y la fecha en que se realizó.

- **Recorrido**: En realidad, un recorrido o *track* no es más que una lista de *waypoint* sin nombres (llamados puntos de *tracks*) que están concatenados unos detrás de otros para definir un recorrido. Los puntos de *tracks* no contienen nombre, ni fecha de realizados.

- **Ruta**: son capaces de definir un recorrido (ya hecho o por hacer en un futuro inmediato). Las rutas están compuestas por un número determinado de *waypoint* y el orden en el que están concatenados para generar una ruta.

- **Información socioeconómica**: Son los datos asociados a los vehículos que controla el sistema.

3.2 Requerimientos de la Solución

El propósito de la definición de requisitos es especificar las condiciones o capacidades que el sistema debe cumplir y las restricciones bajo las cuales debe operar, logrando un entendimiento entre el equipo de desarrollo y el cliente y especificando las necesidades reales de forma que satisfaga sus expectativas.

3.2.1 Requisitos funcionales

Los requisitos funcionales constituyen las capacidades o condiciones, o sea, las utilidades expuestas por la solución para los usuarios (Jacobson, 2000). A continuación se listan estas funcionalidades:

RF1. Asignar recorrido dado los datos obtenidos de un dispositivo GPS.

RF2. Gestionar Vehículo.

2.1 Permitir adicionar los datos asociados a los vehículos en el sistema.

2.2 Permitir modificar los valores de los datos asociados a los vehículos ya existentes en el sistema.

2.3 Permitir eliminar los vehículos existentes en el sistema.

RF3. Realizar estadísticas sobre la información socioeconómica.

3.1 Permitir al usuario conocer el gasto de combustible que han tenido los vehículos en determinados recorridos.

3.2 Permitir al usuario conocer las distancias recorridas por los vehículos en determinados recorridos.

3.3 Permitir al usuario conocer el tiempo total realizado por los vehículos en determinados recorridos.

3.4 Permitir al usuario conocer la velocidad promedio alcanzada por los vehículos en determinados recorridos.

RF4. Detectar paradas en destinos no autorizados.

RF5. Gestionar Recorrido.

5.1 Permitir modificar los valores de los datos asociados a los recorridos ya existentes en el sistema.

5.2 Permitir eliminar los recorridos asociados a los vehículos ya existentes en el sistema.

5.3 Permitir mostrar los recorridos asociados a los vehículos ya existentes en el sistema.

RF6. Gestionar Parada.

5.1 Permitir adicionar datos asociados a un recorrido de paradas.

5.2 Permitir modificar un determinado recorrido de paradas ya existente en el sistema.

5.3 Permitir eliminar un recorrido de paradas de los ya existentes en el sistema.

5.4 Permitir editar las paradas de un determinado recorrido de los existentes en el sistema.

3.2.2 Requisitos no funcionales

Los requerimientos no funcionales hacen relación a las características del sistema que se aplican de manera general como un todo, más que a rasgos particulares del mismo. Estos requerimientos son adicionales a los requerimientos funcionales que debe de cumplir el sistema (Jacobson, 2000).

Requerimientos de interfaz externa

- La aplicación debe mostrar una interfaz sencilla y agradable para el usuario.

Requerimientos de usabilidad

- El sistema debe ser sencillo de usar, así como de fácil adaptación de los usuarios con el mismo.
- El sistema debe presentar mensajes de error que permitan al usuario identificar el tipo de error y comunicarse con el administrador del sistema.

Requerimiento de soporte

- El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes.

Requerimientos de software

- Es posible utilizar el sistema en los Sistemas operativos GNU/Linux, Windows XP, Windows 7. Se requiere PostgreSQL como Sistema Gestor de Base de Datos y PostGis como extensión de PostgreSQL como soporte de datos espaciales.

Requerimientos de hardware

- El sistema requiere un microprocesador igual o superior al Dual Core, con una velocidad de al menos 2.00 GHz en adelante, una memoria RAM de un 1 Gb en adelante y al menos 2 Gb libres en disco duro.

3.3 Descripción del sistema propuesto

Para cumplir con los objetivos del presente trabajo y teniendo en cuenta cada uno de los requerimientos anteriormente planteados se ha decidido que el sistema sea un módulo de la plataforma GeoQ desarrollado en el lenguaje propuesto C++, utilizando como entorno de desarrollo integrado Qt. También, se han identificado las principales funcionalidades que brindará el sistema, con las cuales el usuario podrá visualizar la reelaboración de una trayectoria recorrida por un vehículo dado los datos obtenidos de un dispositivo GPS, así como la realización de la gestión de la información socioeconómica de los vehículos que controla la aplicación. Brindará además, la opción de hacer estadísticas de cada trayectoria, permitirá el análisis de la velocidad de los vehículos en diferentes tramos de una ruta o recorrido y contendrá una interfaz agradable y de fácil uso para los usuarios.

3.4 Descripción de los actores del sistema

Tabla 1: Descripción de los actores del sistema.

Actor	Descripción
Usuario	El usuario, es el encargado de realizar cualquier operación de consulta sobre la información obtenida a partir del dispositivo GPS, además de la gestión sobre la información socioeconómica de los vehículos asociados y la realización de cualquier otra operación que el sistema brinde.

3.5 Diagrama de casos de uso del sistema.

Un diagrama de casos de uso del sistema describe parte del modelo de casos de uso y muestra un conjunto de casos de uso y actores con una asociación entre cada par actor/caso de uso que interactúan (Ver Figura 11) (Jacobson, 2000).

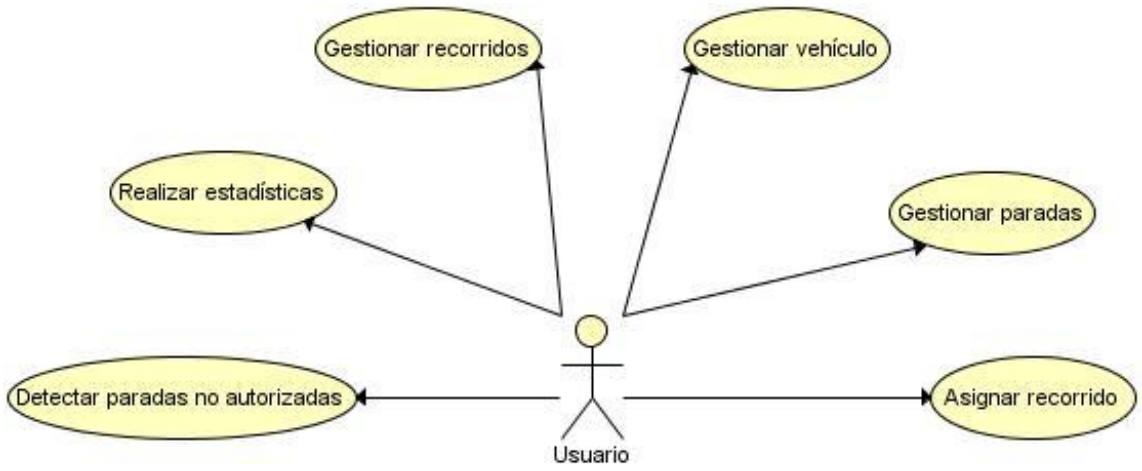


Figura 11: Diagrama del Modelo de CU del Sistema.

3.6 Descripción de los casos de uso del sistema.

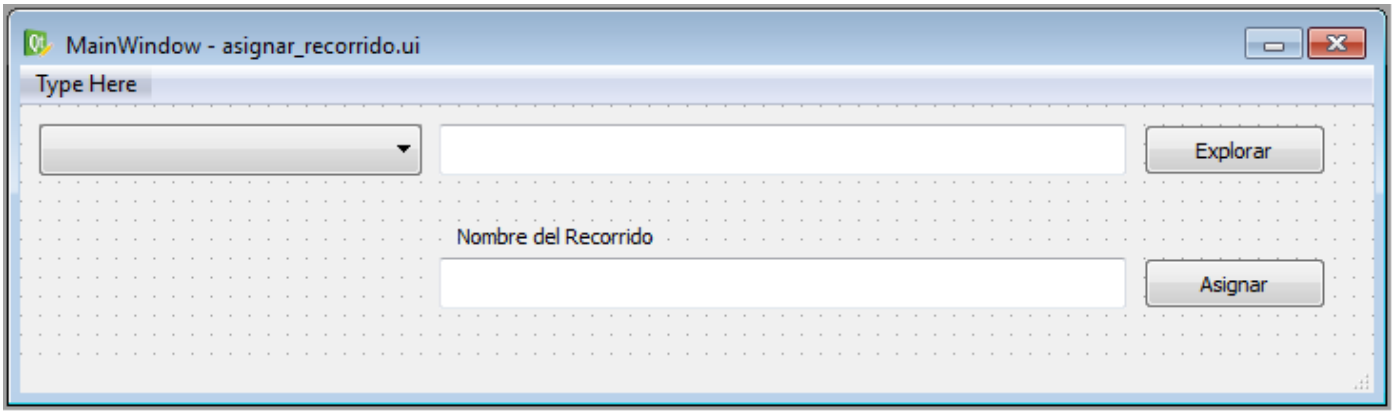
Un caso de uso es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por la unidad del sistema y uno o más actores. El propósito de un caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema. La definición de un caso de uso incluye todo el comportamiento que implica: las líneas principales, las diferentes variaciones sobre el comportamiento normal, y todas las condiciones excepcionales, que pueden ocurrir con tal comportamiento, junto con la respuesta deseada. Desde el punto de vista de los usuarios, estas pueden ser situaciones anormales. Desde el punto de vista de los sistemas, son las variaciones adicionales que deben ser descritas y manejadas (Jacobson, 2000).

Tabla 2: Descripción del caso de uso Asignar recorrido.

Caso de uso:	**Asignar recorrido.**
Actores:	**Usuario**
Resumen:	El CU se inicia cuando el usuario le desea asignar un recorrido a un vehículo determinado. El usuario carga un archivo GPS en el sistema y este le asigna el recorrido realizado al vehículo seleccionado.
Referencias	**RF 1.**
Prioridad	**Crítico.**
Precondición	Para realizar este caso de uso se debe haber ingresado vehículos en la base de datos.

Flujo normal de eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona la opción Asignar Recorrido.	2. El sistema muestra un formulario solicitando la chapa del vehículo al cual desea asignarle el recorrido, la ruta donde se encuentra el fichero GPS y el nombre del recorrido
3. El usuario inserta los datos solicitados y oprime el botón “Asignar”.	4. El sistema verifica la dirección y el estado del fichero.
	5. El sistema carga el fichero y asigna el recorrido al vehículo.

Flujo alterno	
Acción del actor	Respuesta del sistema
	5. El sistema muestra un mensaje de error informando que la ruta es incorrecta o que el fichero

	está en mal estado y pasa directamente a la acción 2.
Prototipo de interfaz	
	

3.7 Conclusiones

Con el desarrollo de este capítulo se ha logrado un mayor entendimiento de los procesos a automatizar, y las características y restricciones que debe poseer el sistema. Como resultado de su elaboración, se ha obtenido el modelo de dominio en el cual se presentan los conceptos que intervienen en los procesos que se realizan. Se definieron los requisitos funcionales y no funcionales para que el sistema sea eficiente y seguro. Se elaboró el diagrama de casos de uso del sistema, el cual presenta el actor y su relación con las funcionalidades que va a desempeñar en el sistema. Finalmente, para garantizar una mejor comprensión del funcionamiento del sistema fue elaborada una descripción de todos los casos de usos a tener en cuenta en este módulo.

CAPÍTULO 4. CONSTRUCCIÓN DE LA SOLUCIÓN PROPUESTA

4. Introducción

En este capítulo se construye la propuesta de solución para controlar y gestionar el flujo informativo que se genera como parte del proceso de control de flota. Dicha propuesta incluye los modelos de análisis y diseño, los cuales dejarán sentadas las bases para comenzar con la implementación de la primera versión de la solución del sistema. Esto se logrará a través de los diferentes diagramas de clases que incluyen dichos modelos. Además, se explican los patrones de diseño empleados para garantizar la viabilidad y calidad requeridas por el sistema a construir.

4.1 Análisis

Durante el análisis, se analizan los requisitos que fueron descritos en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar todo el sistema, incluyendo su arquitectura.

4.1.1 Diagrama de clases del análisis

Un diagrama de clases del análisis es un artefacto en el que se representan los conceptos en un dominio del problema. Representa además, el funcionamiento del mundo real, no de la implementación automatizada del mismo (Ver Figura 12).

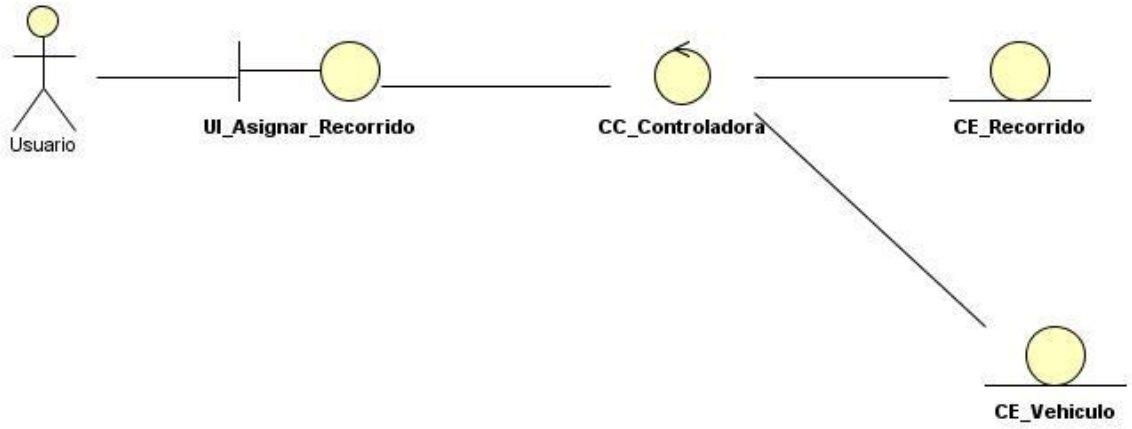


Figura 12: Diagrama de clases del análisis del CU Asignar recorrido.

4.1.2 Diagrama de Interacción

El término genérico interacción, se aplica a varios tipos de diagramas que hacen énfasis en las interacciones entre objetos.

El patrón de interacción entre objetos se muestra en un diagrama de interacción. Los mismos tienen diferentes formas, basadas todas ellas en una misma información subyacente, pero resaltando cada una un punto de vista diferente: diagramas de secuencia y diagramas de colaboración.

Diagrama de secuencia

Un diagrama de secuencia muestra una interacción que está organizada como una secuencia temporal (Ver Figura 13). En particular, muestra los objetos que participan en la interacción mediante sus líneas de vida y mediante los mensajes que intercambian, organizados en forma de una secuencia temporal. No muestra los enlaces existentes entre objetos. Presentan distintos formatos adecuados para propósitos diferentes.

Este tipo de diagrama representa una interacción como un gráfico bidimensional. La dimensión vertical es el eje de tiempo, que avanza hacia abajo de la página. La dimensión horizontal muestra los roles de clasificador que representan objetos individuales en la colaboración.

Cada rol de clasificador se representa mediante una columna vertical-línea de vida. Durante el tiempo que existe un objeto, el rol se muestra por una línea discontinua. Durante el tiempo que dura una activación de un procedimiento en el objeto, la línea de vida se dibuja como una línea doble.

Un mensaje se muestra como una flecha desde la línea de vida de un objeto a la del otro. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

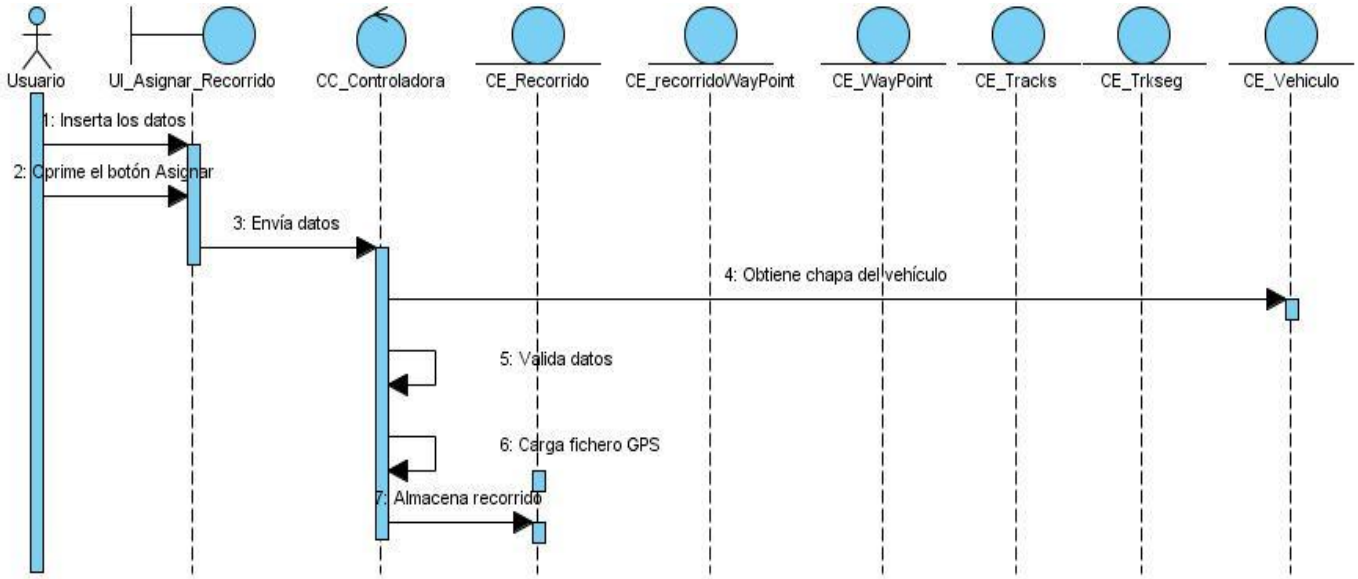


Figura 13: Diagrama de secuencia del CU Asignar recorrido.

Diagrama de colaboración

Un diagrama de colaboración muestra una interacción organizada en torno a los objetos que efectúan operaciones (Ver Figura 14). Es parecido a un diagrama de objetos que muestra los objetos y los enlaces existentes entre ellos, que se necesitan para implementar una operación de nivel más elevado.

Una colaboración modela los objetos y los enlaces significativos dentro de una interacción. Los objetos y los enlaces son significativos solamente en el contexto proporcionado por la interacción.

Un rol describe un objeto, y un rol en la asociación describe un enlace dentro de una colaboración. Los mensajes se muestran como flechas, ligadas a las líneas de la relación que conectan a los roles. La secuencia de mensajes, se indica con los números secuenciales que preceden a las descripciones del mensaje.

La colaboración muestra los parámetros y las variables locales de la operación, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mensajes corresponde a la estructura de llamadas anidadas y el paso de señales del programa.

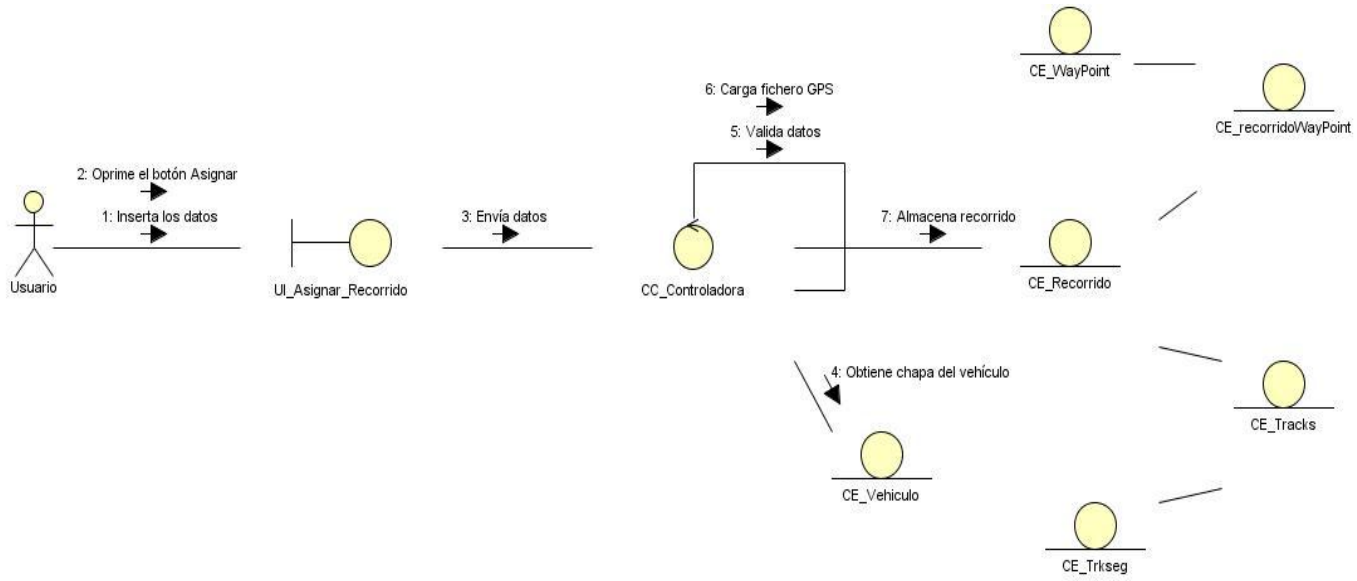


Figura 14: Diagrama de colaboración del CU Asignar recorrido.

4.2 Diseño

El diseño del *software* es realmente un proceso de muchos pasos pero que se clasifican dentro de uno mismo. En general, la actividad del diseño se refiere al establecimiento de la estructura de datos, la

arquitectura general del *software*, representaciones de interfaz y algoritmos. El proceso de diseño traduce los requisitos en una representación de *software* (Pressman, 1999).

4.2.1 Patrones de Diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de *software*. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de *software* que están sujetos a contextos similares. Se debe tener presente los siguientes elementos de un patrón: nombre, problema (cuando aplicar un patrón), solución (descripción abstracta del problema) y consecuencias (costos y beneficios) (Tedeschi, 2011).

Los patrones se dividen de la siguiente manera (Tedeschi, 2011):

- Patrones GRASP.
- Patrones GOF.

4.2.1.1 Patrones GRASP

GRASP es un acrónimo que significa *General Responsibility Asignment Software Patterns* (patrones generales de *software* para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el *software* orientado a objetos. Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

Patrón Experto

El patrón experto en información es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase

que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Patrón Creador

El patrón creador guía la asignación de responsabilidades relacionadas a la creación de objetos, una tarea muy común en sistemas orientados a objetos. El intento básico del patrón creador es encontrar un creador que necesite estar conectado al objeto creado en un evento en particular. Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

Patrón Controlador

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

Patrón Alta cohesión y bajo acoplamiento

Los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento. Maximizar el nivel de cohesión intermodular en todo el sistema resulta en una minimización del acoplamiento intermodular.

Alta cohesión

Plantea que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.

Bajo acoplamiento

Plantea tener las clases lo menos ligadas entre sí que se pueda, de tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las mismas.

Patrón Polimorfismo

Cuando las alternativas o comportamientos relacionados varían según el tipo (clase), se debe asignar la responsabilidad para el comportamiento utilizando operaciones polimórficas a los tipos para los que varía el comportamiento. El polimorfismo tiene varios significados relacionados. En este contexto significa "asignar el mismo nombre a servicios en diferentes objetos" cuando los servicios son parecidos o están relacionados. Los diferentes tipos de objetos normalmente implementan una interfaz común o están relacionados en una jerarquía de implementación con una superficie común.

4.2.1.2 Patrones GOF

Los patrones GOF (*Gang Of Four*) son dirigidos al desarrollo de sistemas orientados a objetos y se encuentran divididos en tres grupos: los de creación utilizados en la abstracción de cómo es creado un objeto, los de estructura que indican cómo se encuentran compuestas las clases y los de comportamiento utilizados en la asignación de responsabilidades en las clases.

Patrón Constructor

Aísla la construcción de un objeto complejo de su representación. El mismo proceso de construcción puede crear diferentes representaciones. Una clase “constructor” se aplica en la creación de otras clases cuando es necesario crear y agregar instancias de clases, contener múltiples instancias de clases y cuando la clase “constructor” posee los datos necesarios para instanciar la nueva clase.

4.3 Arquitectura de *software*

La arquitectura de *software* es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones (Clements, 1996).

Toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos. La arquitectura de las aplicaciones difiere según está distribuido este código (Cornejo, 2001).

Arquitectura 3 Capas

La arquitectura 3 capas consiste literalmente en separar un proyecto en capa de presentación, capa de negocio y capa de datos (Ver Figura 16). Esto permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles (Reynoso, y otros, 2004).

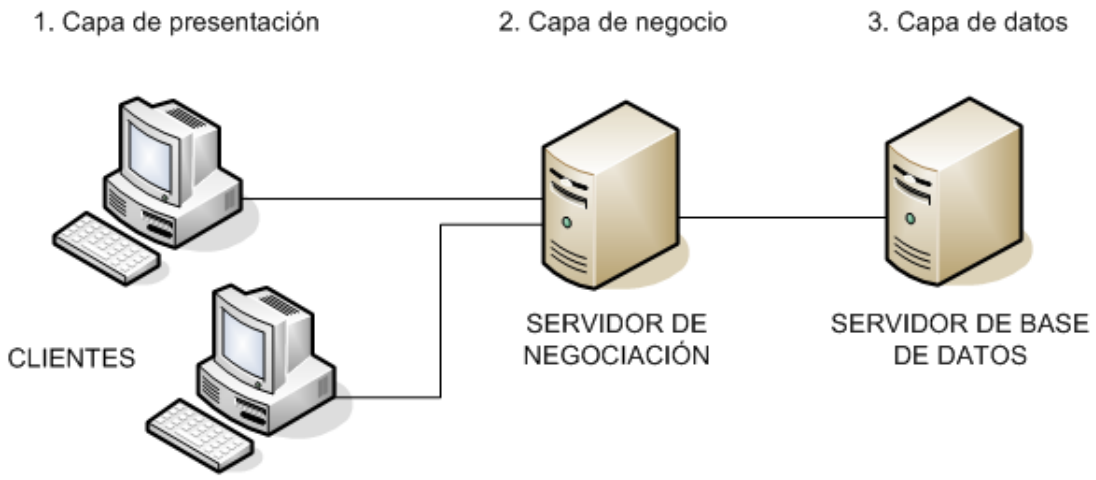


Figura 15: Arquitectura en Tres capas.

Ventajas de esta arquitectura (Reynoso, y otros, 2004):

- El desarrollo se puede llevar a cabo en varios niveles.
- Desarrollos paralelos (en cada capa).
- Aplicaciones más robustas debido al encapsulamiento.
- En caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado.
- Mantenimiento y soporte más sencillo (es más sencillo cambiar un componente que modificar una aplicación monolítica).
- Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- Alta escalabilidad, la principal ventaja de una aplicación distribuida bien diseñada es su buen escalado, es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más *hardware*.
- El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

Capas y niveles (Reynoso, y otros, 2004):

- Capa de presentación: Es la parte que ve el usuario, las pantallas que se le muestra para que interaccione con el programa (también se le conoce como “capa de usuario”), comunicándole la información y recolectando la información suministrada por el usuario en un mínimo de proceso (realiza validaciones para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio llevando y trayendo los datos o registros necesarios. Es la interfaz gráfica del programa y debe ser lo más amena posible para una mejor comunicación con el usuario.

- Capa de negocio: Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todos los procesos que deben realizarse.

- Capa de datos: Es donde residen los datos y la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos y reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Teniendo en cuenta las ventajas y la flexibilidad al cambio que proporciona esta arquitectura se decidió utilizarla para llevar a cabo la implementación del sistema a construir.

4.4 Diagramas de Clases del Diseño

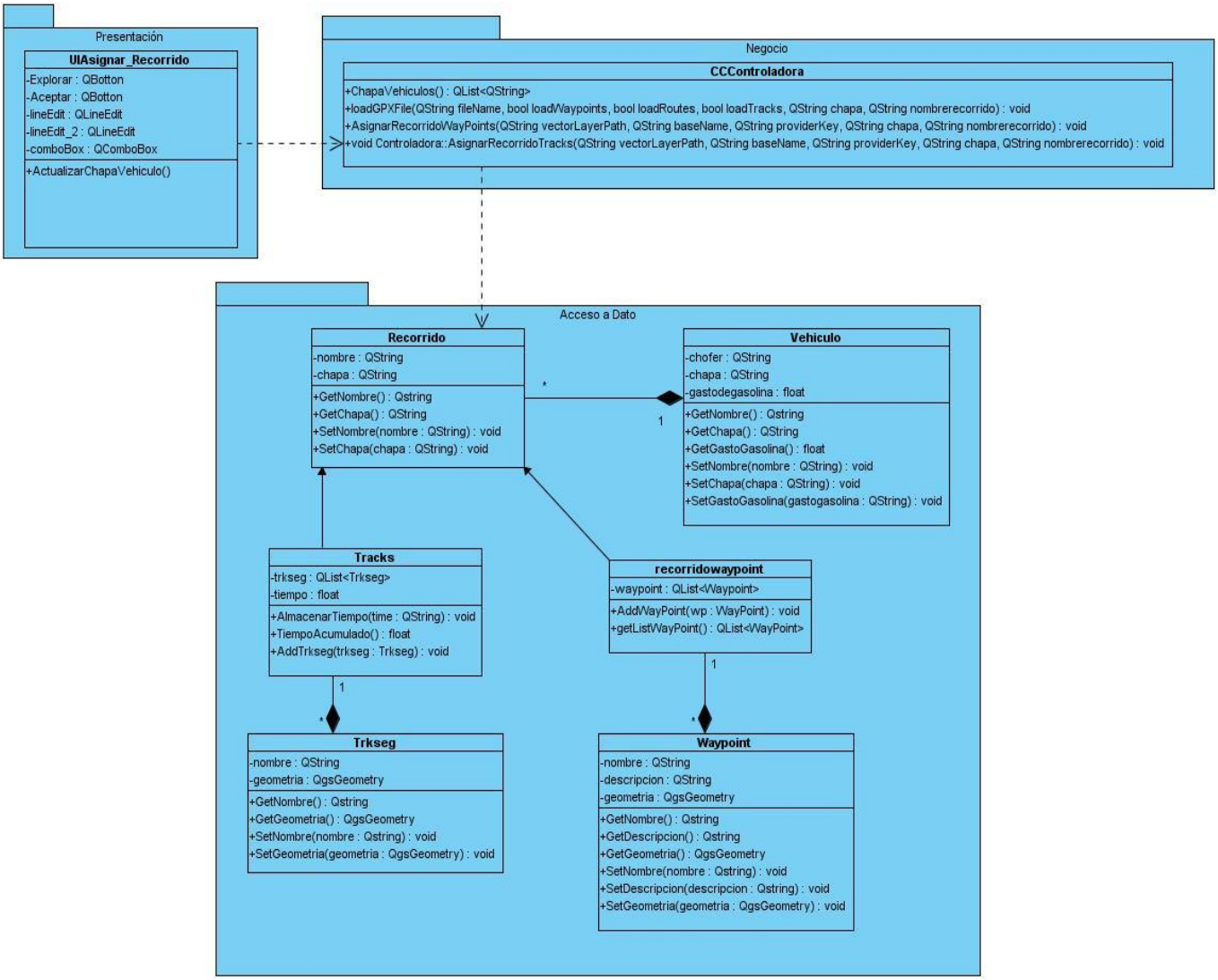


Figura 16: Diagrama de clases del diseño del CU Asignar recorrido.

4.5 Diseño de la base de datos

El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. El mismo se descompone en diseño conceptual, diseño lógico y diseño físico (Andrés, 2001).

4.5.1. Diagrama de clases persistentes

El diagrama de clases persistentes muestra aquellas clases cuyos objetos tienen la capacidad de mantener su valor en el tiempo y el espacio (Ver Figura 17). Por lo general, tienen como base las clases marcadas como entidades debido a que modelan la información y el comportamiento asociado a un proceso, objeto o persona del mundo real.

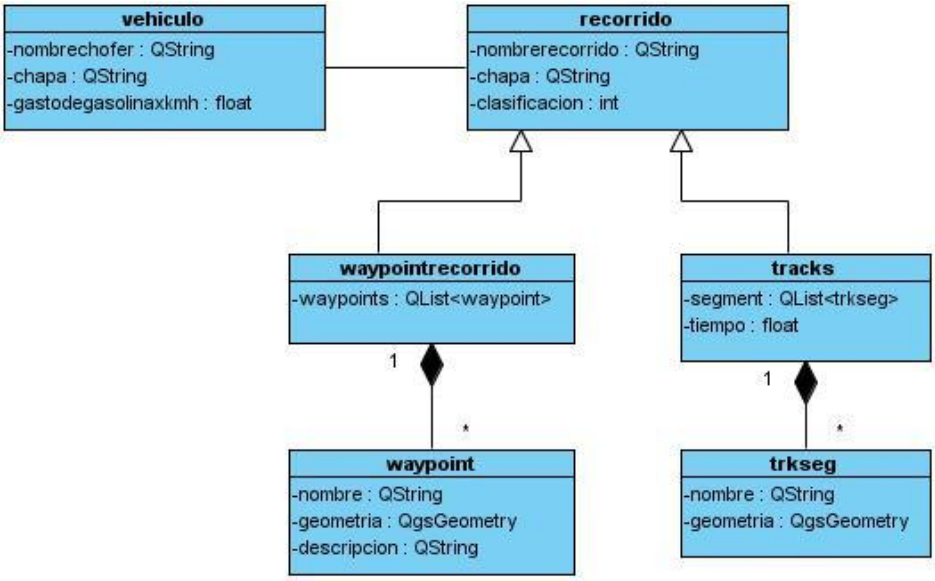


Figura 15: Diagrama de clases persistentes.

4.4.2. Modelo Entidad-Relación

El diagrama Entidad-Relación muestra el modelo de datos utilizado en la aplicación, representando con las entidades los objetos y elementos principales identificados en el problema. Las entidades poseen atributos que son sus características particulares (Ver Figura 18).

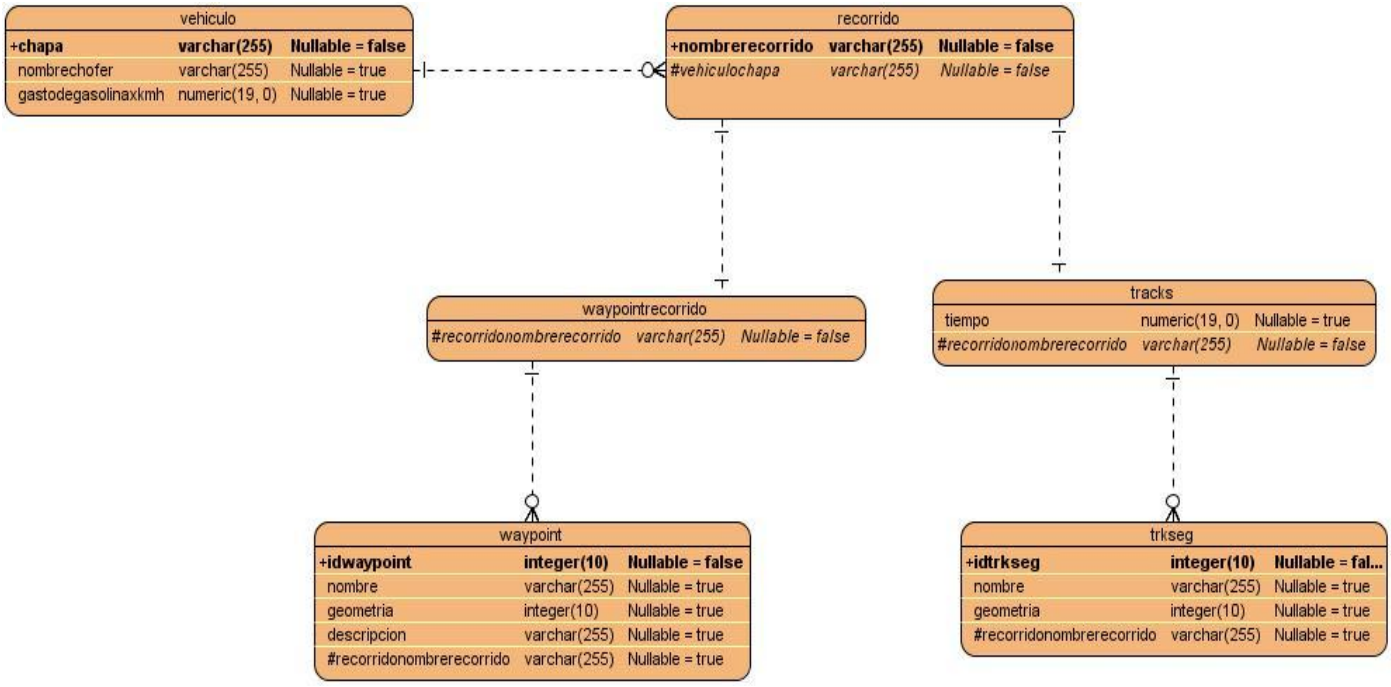


Figura 16: Modelo Entidad-Relación.

4.5 Modelo de despliegue

El Modelo de Despliegue provee un modelo detallado de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de *hardware* y otra información relacionada al despliegue del sistema propuesto.

4.5.1 Diagrama de despliegue

El diagrama de despliegue muestra las relaciones físicas entre los componentes *hardware* y *software* en el sistema propuesto (Ver Figura 19). Se representa como un conjunto de nodos unidos por conexiones de comunicación, donde un nodo puede contener instancias de componentes de *software*, objetos y procesos que representan manifestaciones del código en tiempo de ejecución.

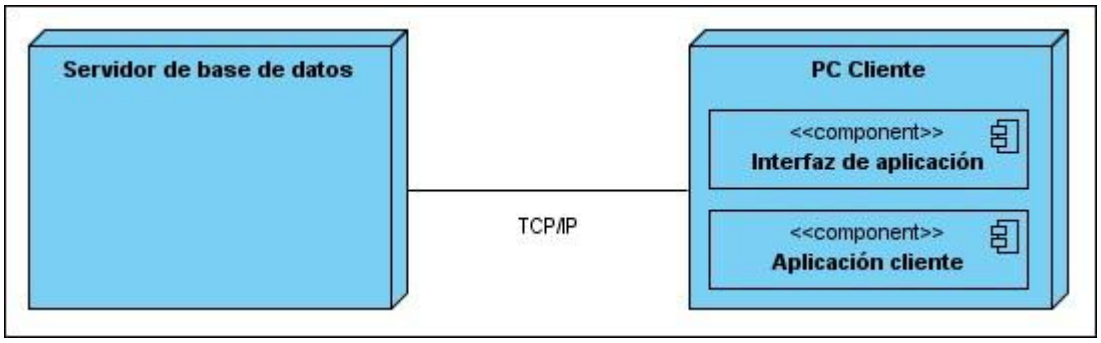


Figura 17: Diagrama de despliegue.

4.6 Modelo de implementación

El modelo de implementación representa la programación de la aplicación en términos de componentes. Describe la organización de los datos, archivos, ejecutables, código fuente y directorios de acuerdo a los mecanismos disponibles para su estructuración en el entorno de implementación. Fundamentalmente, se describen las relaciones que existen entre los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. En este artefacto se describe cómo se implementan los componentes que lo conforman, acoplándolos en paquetes organizados en capas y jerarquías y señalando las dependencias entre los mismos.

4.6.1 Diagrama de componentes

Los diagramas de componentes se utilizan para modelar la vista estática de un sistema. Muestran la organización y las dependencias lógicas entre un conjunto de componentes de *software*, sean componentes de código fuente, librerías, binarios o ejecutables. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema. Se representa como un grafo de componentes *software* unidos por medio de relaciones de dependencia (Ver Figura 20).

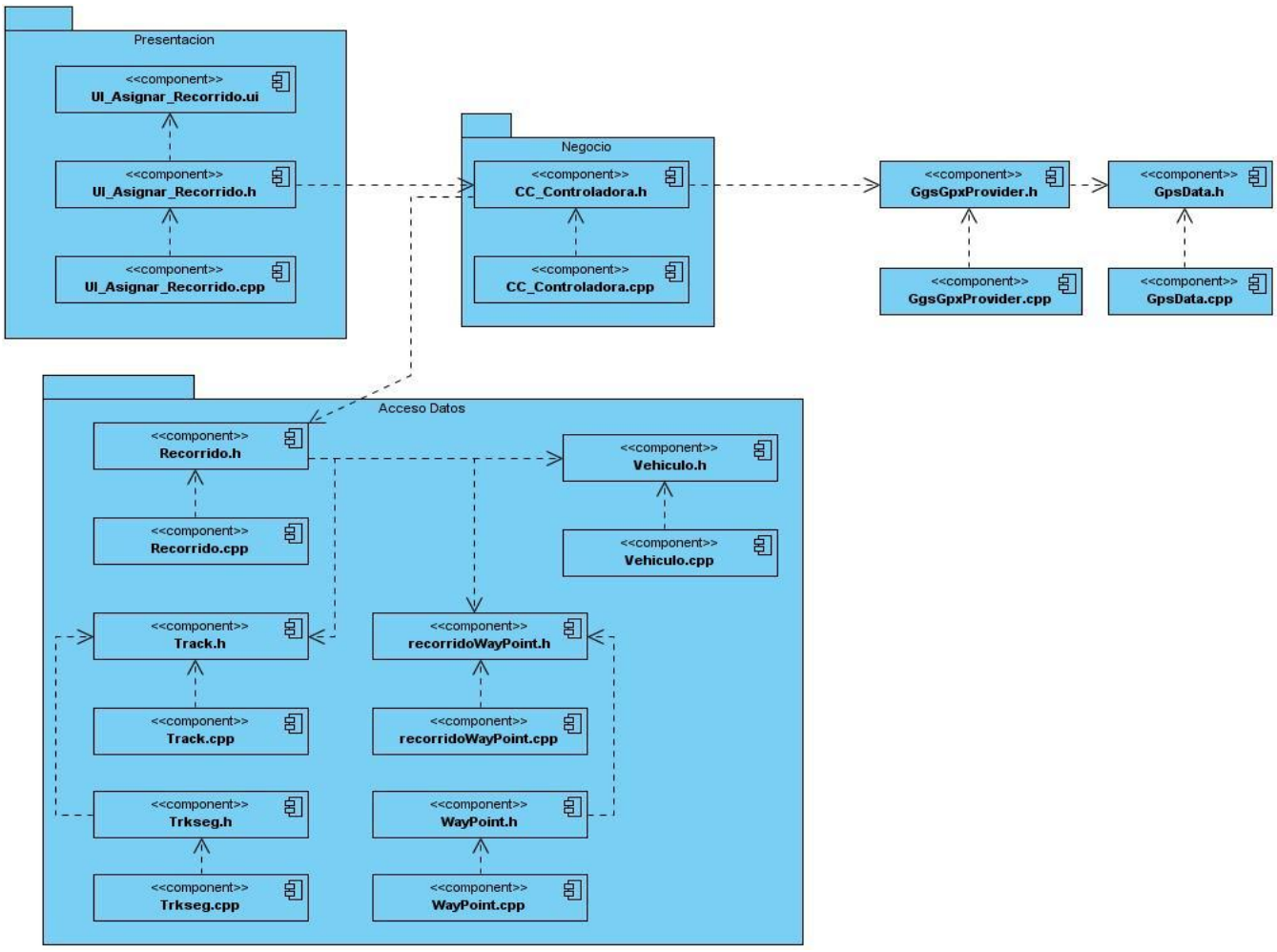


Figura 18: Diagrama de componente del CU Asignar recorrido.

4.7 Prueba

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos. Los resultados son observados, registrados y se desarrolla una evaluación de algún aspecto del sistema o componente.

4.7.1 Diseño de casos de prueba

Un caso de prueba es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para cumplir un objetivo en particular o una función esperada. Los casos de prueba pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. Los casos de pruebas tienen que ser escritos para generar las condiciones de salida deseadas.

4.7.2 Métodos

Para la realización de las pruebas se ha tenido en cuenta el método de caja negra, las cuales se realizan sin importar cómo han sido implementadas las funcionalidades o la estructura interna del programa y se basa en la interacción con la interfaz. Los casos de prueba pretenden:

- Demostrar que las funciones del *software* son operativas.
- Demostrar que las entradas se aceptan de la forma adecuada y que se produce el resultado correcto en la salida.
- Demostrar que la información es manipulada de una forma correcta.

Estas pruebas permiten encontrar funciones incorrectas o no implementadas y errores de interfaz de usuario.

4.7.3. Resultados

Fueron probados seis requisitos funcionales, los que representan el 100% del total de requisitos. Para cada requisito se han tenido en cuenta varios escenarios que implican cada una de las posibles interacciones del usuario o combinaciones de situaciones posibles con el fin de evaluar el comportamiento del sistema ante cada funcionalidad.

El resultado de cada prueba coincide con la especificación de cada requisito funcional lo que demuestra la correcta implementación de los mismos. A continuación se muestra la descripción textual del caso de prueba Asignar recorrido.

Tabla 3: Descripción textual del caso de prueba Asignar recorrido.

Nombre del Requisito	Descripción General	Escenarios de Prueba	Flujo del Escenario
Asignar recorrido	El objetivo de este requisito es asignar un recorrido a un vehículo determinado. El usuario carga un archivo GPS en el sistema y este le asigna el recorrido realizado al vehículo seleccionado.	EP 1.1: Asignar recorrido satisfactoriamente.	➢ El usuario selecciona la opción Asignar recorrido. ➢ El sistema muestra un formulario solicitando la chapa del vehículo al cual desea asignarle el recorrido, la ruta donde se encuentra el fichero GPS y el nombre del recorrido. ➢ El usuario inserta los datos solicitados y oprime el botón “Asignar”. ➢ El sistema verifica la dirección y el estado del

			fichero, lo carga y asigna el recorrido al vehículo.
		EP 1.2: Asignar recorrido insatisfactoriamente.	➢ El usuario selecciona la opción Asignar recorrido. ➢ El sistema muestra un formulario solicitando la chapa del vehículo al cual desea asignarle el recorrido, la ruta donde se encuentra el fichero GPS y el nombre del recorrido. ➢ El usuario inserta los datos solicitados y oprime el botón “Asignar”. ➢ El sistema verifica la dirección y el estado del fichero. ➢ El sistema muestra un mensaje de error informando que la ruta es incorrecta o que el fichero está en mal estado.

Tabla 4: Descripción de las variables del caso de prueba Asignar recorrido.

No	Nombre del campo	Clasificación	Valor nulo	Descripción
1	Chapa del vehículo.	Lista desplegable.	No	Permite introducir la chapa del vehículo.
2	Dirección del fichero.	Campo de texto.	No	Permite introducir la ruta del fichero.
3	Nombre del recorrido	Campo de texto	No	Permite introducir el nombre del recorrido.

Tabla 5: SC "Asignar recorrido".

ID	Escenario	Ruta del fichero	Nombre del recorrido	Chapa del vehículo	Respuesta del sistema	Resultado de la prueba
EP 1.1	Asignar recorrido satisfactoriamente	Home/Cartografías/track.gpx	Los Pinos	QW452	El sistema carga el fichero y asigna el recorrido al vehículo.	Satisfactorio.

EP 1.2	Asignar recorrido insatisfactoriamente.	Home/n/Cartografías/track.gpx	Los Pinos	QW452	El sistema muestra un mensaje de error informando que la ruta es incorrecta o que el fichero está en mal estado.	Satisfactorio.
		Home/Cartografías/track.gpx	Campo vacio	QW452	El sistema muestra un mensaje indicando que hay campos vacios.	Satisfactorio.
		Home/Cartografías/track.gpx	Los Pinos	Campo vacio	El sistema muestra un mensaje indicando que hay campos vacios.	Satisfactorio.

4.8 Conclusiones parciales

Durante el desarrollo del capítulo fueron expuestos algunos temas de gran importancia para la construcción del sistema, entre los que se encuentran los diferentes diagramas de clases correspondientes al diseño y al análisis, que contribuyeron a su adecuada implementación. Además, se abordaron las generalidades para el desarrollo de la aplicación teniendo en cuenta los componentes utilizados, así como las relaciones entre ellos a través del modelo de implementación. También se pudo obtener una vista del sistema en términos de nodos físicos, representada en el diagrama de despliegue.

CONCLUSIONES GENERALES

Después de haber completado el presente trabajo se puede afirmar que el autor pudo poner en práctica los conocimientos adquiridos en las asignaturas recibidas durante la carrera, así como nuevos conocimientos obtenidos en el proceso de estudio e implementación, relacionados con temas como sistemas de información geográfica y sistemas de posicionamiento global.

Con el estudio de los principales conceptos asociados al dominio del problema y las soluciones existentes en la actualidad que realizan control de flotas, se sentaron las bases teóricas para la selección posterior de las herramientas y tecnologías a utilizar.

Se implementó un módulo que permite realizar el monitoreo de objetos móviles de forma remota, la reconstrucción del comportamiento de los mismos en un determinado período de tiempo y la reelaboración de una trayectoria. Esta implementación fue realizada en su totalidad por herramientas libres, lo que posibilitara la eliminación de problemas con la adquisición de licencias y las barreras presupuestales.

Se logró un diseño y una arquitectura de manera sencilla, fácil de usar e intuitiva que proporciona un entorno agradable para los usuarios finales. Todos los requisitos funcionales y no funcionales fueron debidamente implementados.

Además, se cuenta con una documentación técnica que describe claramente todos los artefactos generados, convirtiéndose en una guía para lograr un mejor entendimiento de los procesos automatizados y una base para mejoras futuras.

De esta manera y por todo lo anterior se puede concluir que el objetivo general de la presente investigación fue cumplido, lográndose implementar el módulo de control de flotas para la plataforma GeoQ.

RECOMENDACIONES

Una vez cumplidos los objetivos del presente trabajo y en correspondencia con los resultados obtenidos, los cuales se expusieron en el propio documento, se recomienda como trabajos futuros:

- Integrar el módulo realizado a la plataforma de información geográfica GeoQ de manera que se pongan en práctica las funcionalidades implementadas.
- Extender el módulo realizado a tiempo real para tener un control estricto sobre el trabajo asignado, la eliminación de maniobras innecesarias y la toma de decisiones informadas en caso de imprevistos y contratiempos.

GLOSARIO DE TÉRMINOS

Con el objetivo de hacer más fácil la comprensión del trabajo a continuación se definen algunos términos asociados al dominio del problema:

- **GNU GPL**: Licencia que posibilita la modificación y redistribución del *software*.
- **GEySED:** Centro de Desarrollo de Geoinformática y Señales Digitales.
- **Cartografía digital**: Conjunto de operaciones que permiten a partir de observaciones y mediciones, la representación de una parte o la totalidad de la Tierra.
- **Objeto geográfico**: Abstracción de elementos del mundo real que están asociadas a una posición geográfica y temporal definida.
- **Infraestructura**: Conjunto de medios técnicos, servicios e instalaciones necesarios para el desarrollo de una actividad.

- **Móvil**: Cuerpo que está en movimiento.
- **Ruta**: Camino o itinerario de un viaje.
- ***Software***: Equipamiento lógico o soporte lógico de una computadora digital.
- **Requerimiento**: Petición de una acción que se considera necesaria.
- **Código**: Combinación de letras o números que identifican un producto o persona. Permiten realizar determinadas operaciones o manejar algunos aparatos.
- **Metodología**: Conjunto de métodos que se siguen en una disciplina científica, en un estudio o en una exposición doctrinal.
- **PDF**: Formato de los documentos de *Acrobat Reader* que permite conservar todas las características gráficas durante la transmisión a través de internet.
- **Kilómetro (Km)**: Medida de longitud que equivale a 1000 metros.
- **Gigabyte (GB)**: Unidad de almacenamiento digital de información, equivale a 10^9 bytes.
- **Caso de uso (CU)**: Secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

BIBLIOGRAFÍA CONSULTADA

- **A.Ribeiro, L. García. 2008.** *Sistema de posicionamiento global(GPS):Descripción, análisis de errores, aplicaciones y futuro.* Madrid,España : Instituto de automática industrial, 2008.
- **Ailin Ojuela Duarte, Mauricio Rojas. 2008.** *Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería de Software Educativo.* Medellín : s.n., 2008. 1657-7663.
- **Alieri, Josef. 2009.** Visual Paradigm International. *Visual Paradigm International.* [En línea] 2009. [Citado el: 15 de enero de 2011.] http://www.visual-paradigm.com/product/vpuml/communityedition.jsp..
- **Amaro Calderón, Sarah Dámaris Valverde Rebaza, Jorge Carlos. 2007.** *Metodologías Ágiles.* Trujillo, Perú : s.n., 2007.
- **Bosque, Luis. 2002.** *Sistemas de Informacion Geógrafica.* 2002.
- **—. 2002.** *Sistemas de Informacuión Geográfica.* 2002.
- **Camilo Javier Solis Álvarez, Roberth Gustavo Figueroa Díaz. 2009.** *Metodologías tradicionales vs metodologías ágiles.* s.l. : Universidad Técnica Particular de Loja, 2009.
- **CyMSat. 2005.** Control y Monitoreo Satelital. *Control y Monitoreo Satelital.* [En línea] 2005. [Citado el: 15 de diciembre de 2010.] http://www.cymsat.com.ar/soluciones.htm.
- **ECHEVERRIA. 2007.** *QUÉ ES UN SIG.* 2007.
- **EIRL, AmericaTI. 2008.** Ventajas y Desventajas: Comparación de los Lenguajes C, C++ y Java. *Ventajas y Desventajas: Comparación de los Lenguajes C, C++ y Java.* [En línea] 2008. [Citado el: 18 de enero de 2011.] http://www.americati.com/doc/ventajas_c/ventajas_c.html..
- **FonoAmerica. 2009.** DePeru.com. *DePeru.com.* [En línea] Empresas de Servicio GPS en Perú, 2009. [Citado el: 26 de noviembre de 2010.] http://www.deperu.com/abc/articulo.php?con=858.
- **Jacobson, I., G. Booch, and J. Rumbaugh,. 2000.** *El Proceso Unificado de Desarrollo de Software.* 2000.
- **José David Farré, Guillermo González. 2010.** *APLICACIÓN PARA ANALIZAR Y PROCESAR TRAMAS GPS UTILIZANDO SOFTWARE LIBRE.* Villa Clara : AGENCIA DE SOFTWARE GEOMIX, EMPRESA GEOCUBA LA HABANA., 2010.
- **Jurídicas, Ediciones. 2009.** *Decreto 3422 de 2009.* Colombia : s.n., 2009.
- **Kansas, The University of. 2007.** The Language List. *The Language List.* [En línea] 2007. [Citado el: 21 de enero de 20011.] http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm..

- **Letelier, P. and M.C. Penadés. 2007.** *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* 2007.
- **Lic. José David Farré Rosales, MSc. Lic. Guillermo González Suárez. 2010.** *APLICACIÓN PARA ANALIZAR Y PROCESAR TRAMAS GPS UTILIZANDO SOFTWARE LIBRE.* Villa clara, Cuba : AGENCIA DE SOFTWARE GEOMIX, EMPRESA GEOCUBA, 2010.
- **Manuel, Jose. 2005.** Amanecer. *Amanecer.* [En línea] 2005. [Citado el: 4 de diciembre de 2010.] http://e-amanecer.com/licenciatura/glosario.html.
- **Online, Diccionario Español. 2007.** Diccionario.com. *Diccionario.com.* [En línea] 2007. [Citado el: 15 de diciembre de 2010.] http://www.1diccionario.com/buscar/flota.
- **Orallo, Hernández. 2006.** *El Lenguaje Unificado de Modelado (UML).* 2006.
- **Pedro R. Muro-Medrano, F. Javier Zarazaga Soria. 2006.** *Visualización con MapObjects de la localización de móviles en un sistema de información distribuido orientado a objeto.* Zaragoza, España : Universidad de Zaragoza, 2006.
- **Peña, Juan Luis. 2005.** *Entrada, manejo, analisis, y salida de datos espaciales, teoria general y practica para ESRI ArGIS 9.* 2005.
- **Pressman, R. 1999.** *Software Engineering. A Practitioner's Approach.* USA : s.n., 1999.
- **Río, Víctor G. Sánchez y Homero V. 1995.** *Metodología CASE para el desarrollo de sistemas.* 1995.
- **Sánchez, Marcos. 2007.** *Herramientas CASE para el análisis y diseño orientado a objetos.* 2007.
- **Vega, Noel. 2008.** definicion.de. *definicion.de.* [En línea] 2008. [Citado el: 10 de febrero de 2011.] http://definicion.de/manual-de-usuario/.