Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: Componente para la generación y representación de combinaciones de rutas para el Sistema de Información Geográfica del transporte obrero de la Universidad de las Ciencias Informáticas.

Autor: Yendry López Gómez.

Tutor: Ing. Adrián Gracia Aguila.

La Habana, Junio del 2013“

Año 55 de la Revolución”

No creo que seamos parientes muy cercanos, pero si usted es capaz de temblar de indignación cada vez que se comete una injusticia en el mundo, somos compañeros, que es más importante".

DECLARACIÓN DE AUTORÍA

Se declara a Yendry López Gómez como único autor de este trabajo y se le concede al centro Geoinformática y Señales Digitales (GEySED) de la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmamos la presente declaración de autoría, a los _____ días del mes de _____ del año 2013.

Yendry López Gómez

Ing. Adrián Gracia Aguila

_____

Firma del Autor

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Adrián Gracia Aguila.

Profesión: Ingeniero en Ciencias Informáticas.

Año de graduado: 2009

Correo electrónico: agracia@uci.cu

El compañero Adrián Gracia Aguila es graduado de Ingeniero en Ciencia Informáticas en el año 2009 en la Universidad de las Ciencias Informáticas. Actualmente posee una categoría docente de instructor y se desempeña como líder del proyecto Aplicativos SIG del centro de desarrollo GEYSED perteneciente a dicha universidad. Cuenta con más de cuatro años de experiencia en el desarrollo de *software*, fundamentalmente en el desarrollo de Sistemas de Información Geográfica.

DEDICATORIA

A mi madre por haber luchado junto a mí en los momentos de gran necesidad y nunca haberme dejado solo. Por lo regaños y los castigos que contribuyeron a educarme como mejor persona. Por ser la persona más importante en mi vida y por dejarme ir, a pesar del dolor que esto te causó. Porque siempre serás, aunque no te lo diga tan a menudo como antes, mi "mamita" querida.

A mi padre por todo el apoyo que me ha brindado en mi carrera profesional. Por haber contribuido a que tomara las decisiones correctas desde que llegué a la isla. Gracias por haberme permitido estar en la Lenin y después dejar que decidiera mi destino. Gracias por ser mi mejor amigo

A mi abuela por haber confiado en todo momento en mí, y por siempre darme ánimos.

A Iván por haber asumido mi educación en el momento en que justamente necesitaba un padre. Por haber sido mí ejemplo a seguir durante todo el tiempo que estuve contigo. Por haberme enseñado a ser hombre en la etapa justa en que tenía que aprenderlo.

A Fidel y la revolución por haberme dado la oportunidad de estudiar en esta universidad.

AGRADECIMIENTOS

Quiero agradecer a todas las personas que contribuyeron de una forma u otra a la realización de este trabajo, en especial a Alain León Companioni cuya ayuda fue fundamental, y a mi tutor Adrián Gracia Aguila. Un agradecimiento especial para Rafa, Leandro y Laritza, además de todos mis compañeros de proyecto, apartamento y los grupos con los que he compartido estos cinco años.

RESUMEN

La toma de decisiones para ganar en cuanto al perfeccionamiento empresarial y ahorro de los principales recursos es una primicia actual del estado cubano. La Universidad de las Ciencias Informáticas es una institución con un capital humano de gran envergadura con cerca de quince mil personas entre profesores, estudiantes y trabajadores. A partir de este volumen de personal, el sistema de trasporte obrero de la universidad posee un alto grado de complejidad ya que cuenta con más de cuarenta ómnibus que cubren cerca de setenta itinerarios distintos. Para contribuir a la toma de decisiones respecto al traslado del personal de la UCI, es desarrollado el sistema SIGRutas, herramienta web para la consulta, análisis y representación de las rutas que cubren los ómnibus del sistema de transporte obrero del centro. Existe el inconveniente de que no siempre la selección de una ruta del sistema de transporte obrero por parte de un usuario satisface sus necesidades y este requiere hacer uso del transporte público de La Habana para llegar a su destino final.

El presente trabajo consiste en el análisis, diseño e implementación de un módulo para la obtención de combinaciones de rutas entre el sistema de transporte obrero de la UCI y el sistema de transporte público de La Habana. Con la integración de este módulo al SIGRutas se pretende contribuir de manera eficiente a la toma de decisiones por parte de los usuarios brindando información que satisfaga las necesidades de estos y contribuya al ahorro de recursos.

PALABRAS CLAVES

Grafo, Ruta, Sistema de Información Geográfica

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad, con el constante desarrollo de las Tecnologías de la Informática y las Comunicaciones (TICs), se han realizado avances en la ciencia y la tecnología propiciando que el hombre aumente su perspectiva y desee automatizar las actividades que diariamente efectúa para facilitarse el trabajo. En respuesta a las innovaciones tecnológicas, las disciplinas de Cartografía y Geografía han experimentado grandes cambios individuales y colectivos durante el último medio siglo. La aplicación de la tecnología computarizada a la cartografía durante los años setenta originó la cartografía automatizada, que junto con la adaptación de las matemáticas topológicas a la cartografía condujo al surgimiento de los Sistemas de Información Geográfica (en lo adelante SIG). De 1985 a 1995 se generalizó el desarrollo, la utilización y la aceptación de la tecnología SIG. Durante el período comprendido entre 1995 y 2000, las bases de datos empresariales habilitadas espacialmente y el despliegue de información geográfica en la internet posicionó rápidamente esta nueva tecnología, basada en localización, como parte de la tecnología de la información genérica (Grupo de Autores, 2010).

Los SIG, por la capacidad de gestión espacial que poseen, son indicados para prestar su apoyo en la representación de los sistemas de rutas de trasportación. Conocer la ubicación en el territorio de los distintos tipos de vías aptas para el transporte a través de carreteras, ferrocarriles e itinerarios marítimos, sienta las bases para realizar un buen análisis espacial. La posibilidad de asociar a cada una de estas entidades atributos como el estado de la vía, la capacidad de tráfico, el peligro relativo de cada vía en cuanto a seguridad, además de cualquier información útil para los usuarios del sistema conforman una potente herramienta de gestión y toma de decisiones.

La Universidad de las Ciencias Informáticas (en lo adelante UCI), cuenta con proyectos de desarrollo de *software* pertenecientes a diversos centros de producción, destinados a contribuir a la informatización del país. El centro de desarrollo Geoinformática y Señales Digitales (en lo adelante GEySED) se especializa en la creación de SIG a través de su Línea de Productos de *Software* (LPS) Aplicativos_SIG. Entre los SIG desarrollados por Aplicativos_SIG se encuentra el Sistema de Información Geográfica para el Servicio de Transporte Obrero de la Universidad de Ciencias Informáticas (en lo adelante SIGRutas) el cual fue creado con el objetivo de contribuir a la toma de decisiones respecto a la transportación del personal de la UCI. Éste surge ante la necesidad de automatizar una representación de los recorridos del parque de ómnibus que posee la universidad para la transportación de sus trabajadores. En el mismo se puede visualizar información de las rutas que cubren los ómnibus obreros de la UCI y las paradas que deben

realizar en sus trayectos. Este sistema cuenta con las funcionalidades básicas de un SIG: paneo de mapas, *zoom*, además de otras como el cálculo de distancia entre dos puntos, el cálculo de un área determinada así como la posibilidad de localizar de manera rápida cualquier elemento representado en el mapa.

El SIGRutas contiene información correspondiente a algunas líneas de ómnibus pertenecientes al sistema de transportación público de La Habana, específicamente la línea de ómnibus “P”, por lo que se hace necesario la obtención de las posibles rutas a tomar entre dos puntos de La Habana combinando las que cubren los ómnibus del transporte público de esta ciudad y las del transporte obrero de la UCI, teniendo en cuenta como criterio de optimización la distancia del recorrido, funcionalidades que el SIGRutas no brinda. Además, no posee mecanismos para la obtención de grafos a partir de la cartografía de La Habana lo cual es esencial para la obtención de las combinaciones entre los servicios de transporte antes expuestos. El sistema no brinda información acerca de los ómnibus a tomar en cada ruta así como de las paradas donde deberían realizarse los trasbordos de ómnibus para lograr las combinaciones, lo cual dificulta la toma de decisiones por parte de los usuarios a la hora de tomar un ómnibus para trasladarse entre dos puntos de la ciudad donde sea necesario trasbordar hacia otra ruta para poder llegar al destino deseado.

Por lo anterior planteado se identifica como **problema a resolver:**

¿Cómo obtener y representar combinaciones de rutas entre el sistema de transporte obrero de la UCI y el transporte público de La Habana?

Por lo tanto en la presente investigación se define como **objeto de estudio** los algoritmos para la determinación de caminos mínimos entre dos puntos para un sistema de redes de transporte. El **campo de acción** lo compone el SIGRutas teniendo como base la idea a defender.

Idea a defender: El SIGRutas contará con una funcionalidad que permitirá a los usuarios conocer la información de las combinaciones de rutas entre el sistema de transporte obrero de la UCI y el transporte público de La Habana a partir de criterios de optimización.

Se ha definido como **objetivo general** desarrollar un componente que genere y represente combinaciones de rutas entre dos puntos geográficos de La Habana en el SIGRutas.

Como **objetivos específicos** se han identificado:

- Caracterizar los algoritmos para el trabajo con grafos.
- Implementar un algoritmo para la obtención de caminos mínimos a partir de un grafo ponderado.
- Diseñar e implementar un componente para el cálculo de caminos mínimos sobre los datos que posee el SIGRutas.
- Validar y probar los resultados haciendo uso de pruebas de caja negra sobre las funcionalidades, y de caja blanca sobre la estructura del código.

Para dar cumplimiento al objetivo planteado se han definido las siguientes **tareas de la investigación:**

1. Caracterizar algoritmos para la generación de grafos a partir de la representación cartográfica de un sistema de redes de transporte.
2. Caracterizar los algoritmos de optimización de grafos.
3. Caracterizar los algoritmos para el cálculo de ruta mínima entre dos puntos de un grafo.
4. Implementar el algoritmo de la tarea No 3.
5. Desarrollar un componente que se integre con el Sistema de Información Geográfica para el Trasporte Obrero de la UCI que soporte las funcionalidades de la implementación del algoritmo propuesto.
6. Probar y validar el algoritmo propuesto utilizando técnicas de caja negra y caja blanca.

Como métodos teóricos de la investigación se han seleccionado:

- Análisis Histórico-Lógico: Este método permite conocer cómo han evolucionado los diferentes algoritmos para el cálculo de combinaciones de rutas en los Sistemas de Información Geográfica.
- Analítico Sintético: Con este método es posible realiza una síntesis de los exponentes más significativos para la presente investigación a partir de un análisis exhaustivo de todos los referentes teóricos y la bibliografía relacionada con el tema en cuestión. Esto permite enfocar la investigación solo a lo que es verdaderamente relevante para el desarrollo de la solución.

Como métodos empíricos:

- Análisis Documental: Permite realizar la búsqueda de la información referente al tema de investigación a través de diferentes fuentes bibliográficas.

Posibles resultados: Componente para el SIGRutas que responda a la obtención de combinaciones de rutas entre el transporte obrero de la UCI y el transporte público de La Habana.

El presente documento está estructurado en 4 capítulos.

Capítulo 1. Fundamentación Teórica.

En este capítulo se exponen los conceptos fundamentales para el entendimiento del contenido de la investigación. Se realiza un bosquejo sobre el estado del arte[1] de las funcionalidades para la obtención de rutas y caminos en los Sistemas de Información **Geográfica** para el manejo de información referente a transportación.

Capítulo 2. Herramientas y Tecnologías.

En este capítulo se realiza un estudio sobre las herramientas, lenguajes, tecnologías y metodología de desarrollo de software utilizado para el desarrollo de la aplicación.

Capítulo 3. Características del sistema.

Este capítulo abarca el diseño de la aplicación a implementar identificándose los principales requisitos y casos de uso a automatizar. Se obtienen los diagramas referentes al diseño.

Capítulo 4. Implementación y Pruebas.

Contiene la implementación de los requisitos funcionales e interfaz definida en el diseño así como las pruebas realizadas a los casos de usos del sistema y sus resultados.

[1] Estado del arte: Representa el avance y desarrollo que se tiene en el tema en cuestión.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se tratarán los temas correspondientes a la fundamentación teórica de la investigación. Se expondrán los principales conceptos asociados al tema en cuestión, además se realizará un análisis del estado del arte de los principales algoritmos para la obtención de combinaciones de rutas en un SIG.

1.2. Argumentación de la problemática existente

El SIGRutas cuenta con un grupo de funcionalidades que permiten al usuario interactuar de manera dinámica con la información que brinda el mismo. Este muestra las rutas que cubren los ómnibus de la UCI a través de la ciudad de La Habana, además cuenta con información de las rutas de los ómnibus del sistema de transporte público de La Habana, específicamente los ómnibus articulados o conocidos como “P”. Esta información es necesaria para la gestión de la trasportación obrera y para la planificación de los itinerarios por parte de los usuarios para el traslado hacia esta ciudad. En el sistema se pueden localizar las distintas paradas de los ómnibus así como las paradas más cercanas a un punto del mapa determinado por el usuario.

El SIGRutas no cuenta con funcionalidades que permitan a los usuarios combinar las rutas de los ómnibus de la UCI con los del transporte público de La Habana para obtener una posible ruta para el traslado de los usuarios teniendo como punto de inicio la universidad y como destino un punto de la ciudad[2], siendo esto actualmente una situación no automatizada y que los usuarios deben hacer a partir de su interpretación lógica de lo mostrado en el sistema. La ruta extraída de las combinaciones debe estar sujeta a criterios de optimización seleccionados por el usuario, tales como: tiempo de duración del viaje, flujo de tráfico, estado de las vías o la de menor distancia. El sistema no posee mecanismos que posibiliten el trabajo con grafos, necesarios para el desarrollo de un componente que permita la obtención de combinaciones de caminos sobre un sistema de redes de transporte, asimismo no cuenta con un algoritmo capaz de obtener, a partir de la información sobre las rutas de los ómnibus, un grafo donde los nodos sean las paradas de todas las rutas y las aristas las conexiones entre estos puntos. Se ha tenido en consideración durante el estudio de la problemática existente posibilitar al usuario una ruta óptima entre dos puntos llegándose a la conclusión que, a partir de la información que se tiene, solo es posible obtener como criterio de optimización la ruta más corta entre estos puntos. El algoritmo a implementar debe

[2] Entiéndase que estos puntos de destinos deben ser puntos significativos o de interés de la ciudad o puntos de paradas de ómnibus.

cumplir con los requisitos de optimización en cuanto a tiempo de ejecución para evitar sobrecargas desagradables del sistema, así como ser compatible con el sistema en general para su integración con el mismo.

1.3. Conceptos asociados al dominio del problema

1.3.1. SIG

A continuación se exponen conceptos del término definidos por diferentes autores:

- *"Sistema de Información Geográfica es el conjunto de mapas de la misma porción del territorio, donde un lugar concreto tiene la misma localización en todos los mapas incluidos en el sistema de información, resultando posible realizar análisis de sus características espaciales y temáticas para obtener un mejor conocimiento de esa zona"* (Bosque, 1992).
- *"Los sistemas de información geográfica son los conjuntos de instrumentos y métodos especialmente dispuestos para capturar, almacenar, transformar y presentar información geográfica o territorial referenciada al mundo real***"** (Calvo, 1993)**.**
- *"Un sistema para capturar, almacenar, comprobar, integrar, manipular, analizar y visualizar datos que están espacialmente referenciados a la tierra"* (Chorley, 1987).

  Después de analizar estos conceptos se ha llegado a la conclusión de que un SIG es una integración de herramientas y tecnologías que permite manipular y analizar información geográfica.

Geometría Topológica

Es el estudio de las propiedades geométricas invariables dentro de espacios cartográficos. Esta técnica es utilizada en los SIG para describir áreas de espacios geográficos de distintas maneras (multi-lineales, multi-poligonales) (Esri, 2010).

1.3.2. Sistema de transporte

Un sistema de transporte es un conjunto de instalaciones fijas (redes y terminales), entidades de flujo (vehículos) y un sistema de control que permiten movilizar eficientemente personas y bienes, para satisfacer necesidades humanas de movilidad (Martinez, 2002).

Ruta de Transporte

Una ruta de transporte es un conjunto de vialidades por donde circulan unidades de transporte en servicio entre dos punto terminales (Molinero, y otros, 2005).

Línea de Transporte

Las líneas de transporte están constituidas por las vialidades por donde operan una o más rutas de transporte (Molinero, y otros, 2005).

Medio de Transporte

Vehículo de movilidad que permite el traslado de personal y logística entre dos puntos distintos (Molinero, y otros, 2005).

1.3.3. Teoría de grafo

La Teoría de Grafos[3] permite asociar a redes de transporte o de circulación una estructura sencilla de nodos y arcos conectados, ya que sus elementos pueden vincularse fácilmente a objetos geográficos de la vida real. De esta forma los *nodos* pueden representar ciudades, paradas o estaciones, cruces de carreteras, aeropuertos, puertos; de manera similar los *arcos* que conectan los nodos son asimilables a carreteras, líneas de ferrocarril, tendido de cables, trama de calles, canales, cauces fluviales, rutas aéreas o marítimas, etc. a través de los cuales se realizan movimientos de personas, mercaderías, información, materia, etc (Cardozo, et al., 2009). A continuación se exponen algunos conceptos relacionados con la teoría de grafos esenciales para un mejor entendimiento de la solución.

Definición de grafo

Un grafo es un par $G = V, E$, donde V representa un conjunto de vértices y E representa un conjunto de aristas que satisface que $E \subseteq V^2$; por lo tanto, los elementos de E son subconjuntos de pares de elementos del conjunto V. Para evitar ambigüedades en la notación siempre se asume que $V \cap E = \emptyset$. Dos vértices $v1, v2$ son adyacentes si $v1, v2$ es una arista de G (Diestel, 2000).

Definición de camino

Se denomina camino al grafo no vacío $P = (V, E)$ de la forma:

$$V = x_0, x_1, \dots, x_k \quad E = x_0x_1, x_1x_2, \dots, x_{k-1}x_k$$

[3] Un problema no resuelto aún es la falta de homogeneidad en la terminología utilizada para designar a los elementos del grafo: por ejemplo, los elementos puntuales pueden denominarse vértice, nodo, nudo, etc., mientras que los lineales pueden llamarse arista, arco, eje, segmento, etc. Ante la incertidumbre y debido a que son los términos más usados, se optó por las primeras denominaciones respectivamente (arista y vértice).

Donde todos los x_i son distintos. Los vértices x_0 y x_k se encuentran conectados por P y son llamados sus extremos. Los vértices $x_1, \dots, x_{k-1}$ son los vértices interiores de P. La longitud de un camino es el número de aristas[4] del mismo, y el camino de longitud k es denotado P^k (Diestel, 2000).

Algoritmos para la exploración de grafos

Existe una gran cantidad de problemas que se pueden formular en términos de grafos, pero con frecuencia estos problemas se traducen en buscar un nodo, un camino o un patrón específico en el grafo asociado. Entre los algoritmos utilizados para dar solución a estos problemas se encuentran los algoritmos de búsqueda o exploración. Estos se usan como algoritmos de "fuerza bruta", donde se realizará una búsqueda exhaustiva por el espacio de posibles soluciones hasta encontrar una que satisfaga los criterios exigidos o determinar que no existe solución. Se emplean en problemas cuyas soluciones se construyen por etapas, es decir, se expresan en forma de n-tuplas$(x_1, x_2, \dots, x_n)$, donde cada $x_i \in S_i$ representa la decisión tomada en la etapa i-ésima entre un conjunto finito de alternativas. Se establecen dos categorías de restricciones (Universidad de Jaén, 2006):

- Restricciones explícitas: Indican los conjuntos S_i a los que pertenecen cada una de las componentes de una tupla solución (Universidad de Jaén, 2006).
- Restricciones implícitas: Son las relaciones que se han de establecer entre las componentes de la tupla solución para satisfacer la función criterio (describen la forma en la que se relacionan las x_i) (Universidad de Jaén, 2006).

El espacio de soluciones estará formado por el conjunto de tuplas que satisfacen las restricciones explícitas y se puede estructurar como un árbol de exploración: en cada nivel se toma la decisión de la etapa correspondiente. Luego de construido el árbol de exploración, el algoritmo para resolver el problema consistirá en realizar un recorrido por el árbol hasta encontrar las soluciones (Universidad de Jaén, 2006).

Técnica *Backtracking* (Vuelta Atrás)

Es un algoritmo de recorrido en profundidad para resolver problemas de búsqueda exhaustiva. Contiene una función de factibilidad que determina en un momento dado si existirá o no una solución factible. Si se decide que a partir del nodo donde se encuentra en ese momento no se llegará a ninguna solución

[4] En un grafo ponderado (las aristas poseen un costo unitario u) la longitud de un camino es la sumatoria de la ponderación de cada arista que pertenece al camino.

factible, se deshacen todas las combinaciones que contengan ese nodo y se regresa al estado anterior. Esta técnica es demasiado lenta y realiza un análisis de un número excesivo de nodos (Dorigo, y otros, 2004).

Técnica de Ramificación y Poda (*Branch & Bound*)

El objetivo de esta técnica es evitar analizar la mayor cantidad de nodos posible para mejorar el rendimiento del algoritmo de solución. Usando esta técnica se calcula en cada nodo hijo el valor de una cota (función cota) del posible valor de aquellas soluciones que se pueden encontrar explorando esa rama del árbol. A partir de aquí se utiliza la técnica Vuelta Atrás obteniendo en cada vuelta una cota que podará aquellos nodos cuya cota sea menor que la devuelta (Skiena, 2008).

Algoritmos para recorridos de grafos

BFS: *Breadth-First Search* (Recorrido a lo Ancho). A partir de un vértice, visitar sus vecinos, luego los vecinos de sus vecinos, y así sucesivamente, si al finalizar quedan vértices sin visitar, se repite a partir de un vértice no visitado (Skiena, 2008).

DFS**:** *Depth First Search* (Recorrido en Profundidad). A partir de un vértice, visitar alguno de sus vecinos, luego un vecino de éste, y así sucesivamente hasta llegar a un vértice que ya no tiene vecinos sin visitar. Continuar por algún vecino del último vértice visitado que aún tiene vecinos sin visitar. Si al finalizar quedan vértices sin visitar, se repite a partir de un vértice no visitado (Skiena, 2008).

Problema del camino mínimo

El problema del camino mínimo consiste en encontrar la menor distancia entre dos pares de vértices, entre un vértice y todos los demás, o entre todos los pares de vértices de un grafo. Para dar solución a este problema se expondrán tres algoritmos fundamentales (Dorigo, y otros, 2004).

- Bellman & Ford: Permite calcular el camino mínimo entre dos vértices de un grafo (Dorigo, y otros, 2004).
- Floyd: Permite calcular el camino mínimo entre todos los pares de vértices de un grafo (Dorigo, y otros, 2004).
- Dijkstra & Moore: Permite calcular el camino mínimo entre un vértice S y todos los demás vértices alcanzables desde S. Tiene como salida el árbol de expansión mínima del grafo (Dorigo, y otros, 2004).

Se selecciona este último algoritmo para solucionar el problema planteado ya que es el de menor complejidad temporal y el que mejor se acopla para dar solución al problema.

1.4. Análisis de las soluciones existentes

1.4.1. Google Transit

Google Transit **(Figura 1)** es una aplicación derivada de *Google Maps* que permite obtener instrucciones paso a paso sobre cómo llegar a un lugar, buscar paradas de transporte público en una zona determinada y obtener información y horarios de la estación. Ofrece servicio en casi quinientas ciudades en todo el mundo y funciona en dispositivos móviles.

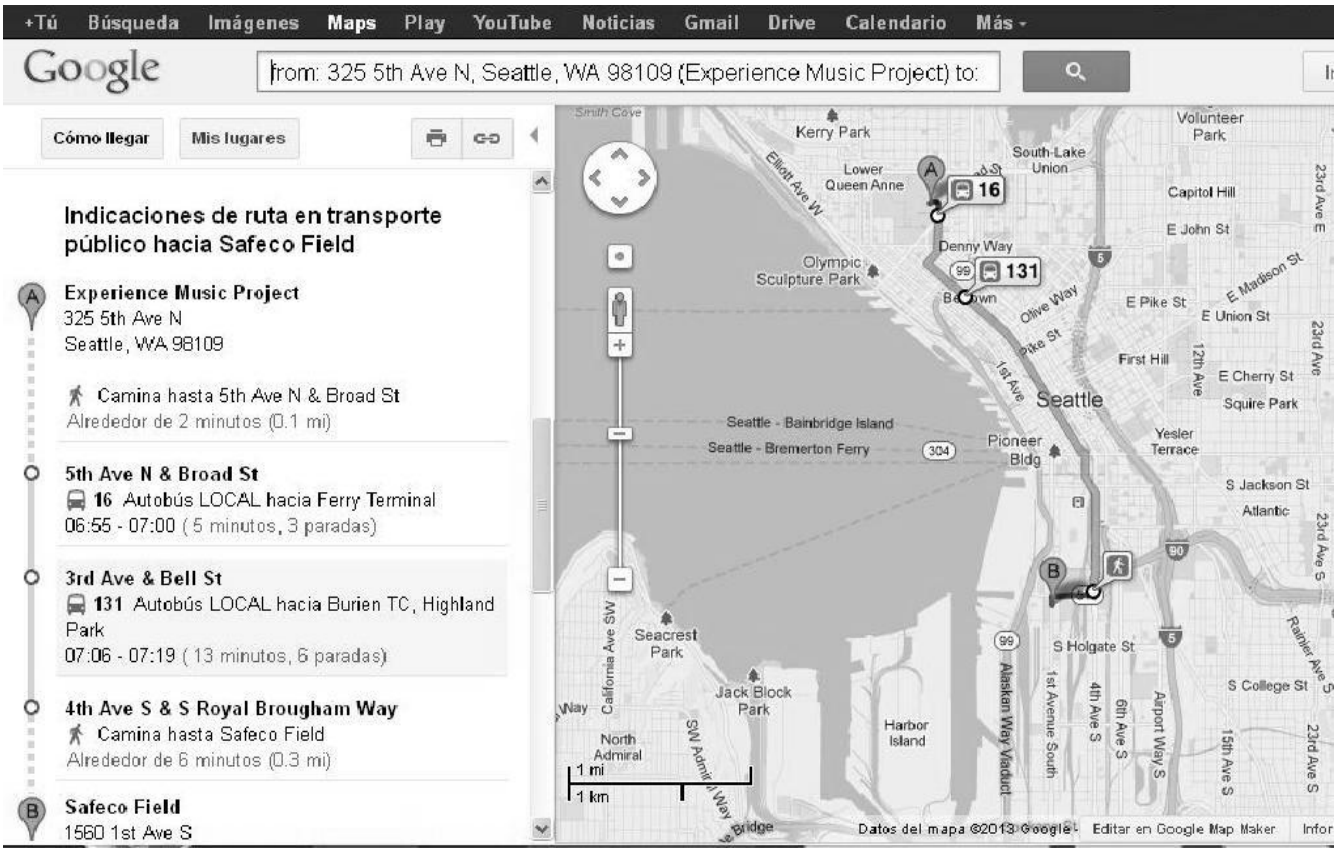

Figura 1: *Google Transit.* (Google, 2013)

Se utiliza a partir de una *url* específica para la ciudad o desde la página de inicio de *Google Maps*. Al introducir un origen y un destino en la opción “Cómo llegar” se muestran tres iconos que permite activar el cálculo del recorrido para coche, transporte público o desplazamiento a pie. El usuario puede introducir en

las casillas de origen y destino direcciones, establecimientos públicos o nombres de negocios; la integración con el buscador genera una lista de resultados (indicados sobre el mapa) que permite señalar el lugar de destino. Para calcular las rutas, *Google Transit* utiliza la información sobre horarios de paso de metro y una estimación de tiempos de recorrido a pie y en autobús (Doovive, 2011). La compañía *Google* no da acceso a los algoritmos que utiliza este servicio para el procesamiento de la información, además de que este servicio solo puede usarse para los sistemas de transporte contenidos en las bases de datos de *Google* y no se puede obtener de manera genérica para cualquier sistema. Por lo tanto el uso de *Transit* no es factible para dar solución a la problemática planteada en este documento.

1.4.2. Tras-Mi-SIG

Tras-Mi-SIG **(Figura 2)** es una aplicación de *software* libre y código abierto de escritorio creada en lenguaje *C# .Net Framework* Versión 1.0[5] para la obtención de rutas de la empresa de transporte Transmilenio, la cual opera en la ciudad de Bogotá en Colombia. Esta herramienta genera un reporte que contiene varios criterios para decidir la ruta óptima entre dos estaciones del sistema de transporte atendiendo a parámetros de entrada como son el día, la hora y el sentido del viaje. La persistencia de datos se maneja a partir de una base de datos en Postgres/Postgis. (Carrillo Romero, 2008)

[5] Esta versión fue liberada por la compañía Microsoft como software libre y de código abierto.

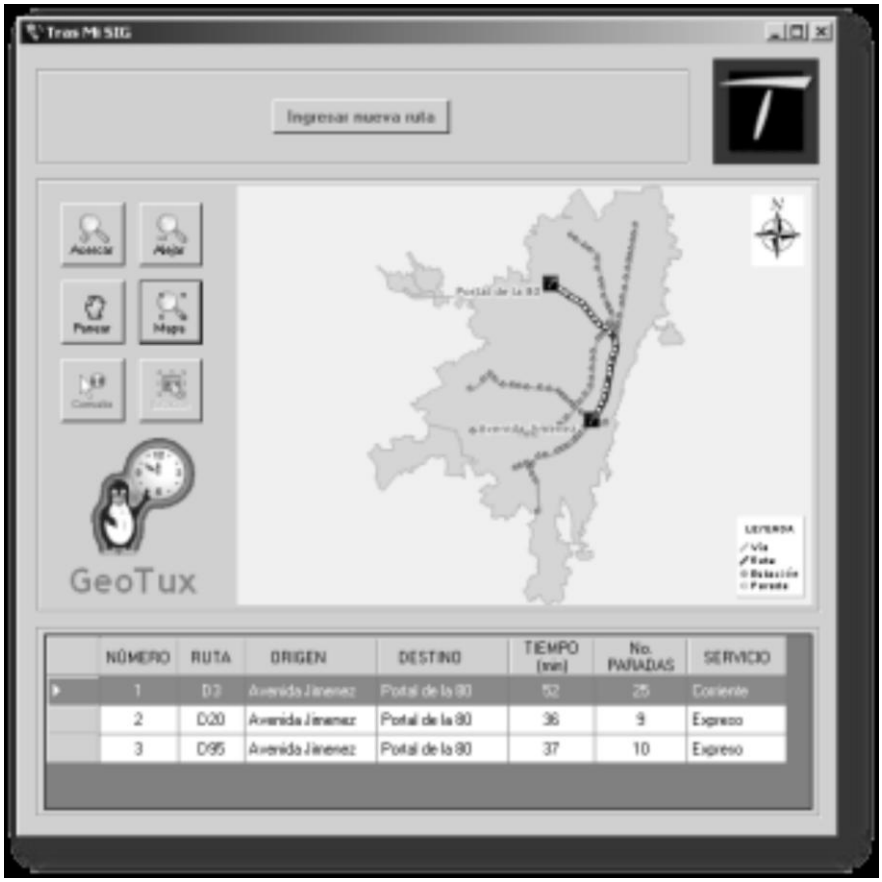

Figura 2: Tras-Mi-SIG. (Carrillo Romero, 2008)

El sistema no permite obtener combinaciones de rutas entre distintas líneas de los ómnibus de la empresa Transmilenio lo cual podría disminuir el tiempo de viaje manteniendo el mismo costo (Carrillo Romero, 2008). Esta aplicación posee similitudes con el módulo a desarrollar que proveerá de una solución a la problemática planteada en este documento. El código fuente de Tras-Mi-SIG puede ser obtenido y contribuir a la implementación de la solución del problema planteado en cuanto a la lógica de los procesos internos y algoritmos utilizados en este sistema que puedan ser aplicados al que se desea desarrollar.

1.4.3. Rutas

Rutas **(Figura 3)** es una herramienta que permite la resolución de problemas reales para flotas de vehículos, el cálculo de rutas y su gestión, desarrollado por la Universidad Politécnica de Valencia. El software Rutas, está basado en la experiencia profesional de empresas e ingenieros, y el conocimiento científico adquirido por miembros de un grupo de I+D+I. Este software combina e integra tres elementos:

el sistema de información geográfica SIG, la información del sistema logístico (VRP-XML) y toda la información del sistema empresarial necesaria para crear y resolver los modelos matemáticos mediante técnicas de optimización combinatoria. Además, integra todo un conjunto de funciones de interés para la gestión logística de los procesos de aprovisionamiento y distribución de mercancías (Rodríguez Villalobos, 2006).

Entre sus principales funciones se encuentran:

- Cálculo y gestión de distancias, tiempos y costes de transporte.
- Cálculo y optimización de rutas de transporte.
- Gestión de flotas de vehículos.
- Gestión de órdenes de servicio.
- Definición de zonas de distribución.
- Análisis de flujos logísticos.
- Diseño y análisis de una red de transporte.
- Planificación de rutas de reparto y aprovisionamiento.
- Definición y análisis de ventajas horarias de entrega o recogida.
- Generación de mapas, mejora de la documentación logística.
- Seguimiento de vehículos (*GPS tracking*).
- Reducción de costes y tiempos de transporte.
- Compartir e integrar la información sobre rutas con clientes y proveedores.
- Exportar información sobre localizaciones e itinerarios para otro *software* cartográfico (*Google Earth, OziExplorer, GPS Visualizer, CompeGPS, Google Maps, GPS TrackMaker, TomTom, etc.*).

Figura 3: Software Rutas. (Rodríguez Villalobos, 2006)

Entre sus principales ventajas se encuentra que la cobertura cartográfica del sistema es casi global al utilizar los servicios de *Google Maps* y *Ms BingMaps*. Además, Rutas permita la integración y comunicación con las más recientes herramientas web en Sistemas de Información Geográfica. Tiene una estructura modular, abierta y extensible que le permite integrarse con otros sistemas de información e incorporar nuevas rutinas de cálculo y análisis de problemas de rutas. La estructura de datos XML es una de las aportaciones más novedosas de este proyecto (Rodríguez Villalobos, 2006).

Este software está desarrollado para funcionar sobre sistema operativo Windows (XP, Vista, *Seven*) y su comercialización se realiza bajo una licencia privativa. Además de que no se ofrecen datos especifico de las tecnologías y herramientas utilizadas para su desarrollo, así como tampoco del código fuente y algoritmos utilizados. Por lo anterior planteado esta herramienta no se considerará para dar solución al problema planteado.

1.5. Conclusiones del capítulo

La creciente necesidad de contar con sistemas automatizados que faciliten las acciones diarias de las personas ha provocado una explosión tecnológica durante las últimas décadas. La tendencia de mejorar los sistemas y hacerlos más fiables y efectivos es cada día mayor en todas las industrias del software. Por este motivo se ha realizado una investigación profunda sobre los principales elementos que pueden aportar a la solución de la problemática planteada.

A raíz del estudio de los referentes teóricos y prácticos realizados en este capítulo relacionados con la teoría de grafos, los sistemas de información geográfica y la correlación existente entre estos para la representación y análisis de la información relacionada con sistemas de transporte, los algoritmos matemáticos para la optimización de grafos así como de las necesidades del sistema SIGRutas se puede concluir que se cuentan con las bases necesarias para dar solución al problema planteado.

CAPÍTULO 2: HERRAMIENTAS Y TECNOLOGÍAS

2.1. Introducción

A continuación se describen las herramientas, metodología a utilizar para el proceso de desarrollo del *software*, así como de las tecnologías utilizadas para el desarrollo del sistema propuesto.

2.2. Lenguajes de programación

2.2.1. UML (V 2.0)

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML) es el lenguaje más conocido para el modelado de sistemas de *software*. Este lenguaje gráfico puede ser utilizado para la visualización, especificación, construcción y documentación de los artefactos de un sistema a partir de diversos tipos de diagramas los cuales muestran las características de las entidades representadas en los mismos. Además, UML es apropiado para modelar cualquier tipo de sistema, ya sean sistemas de información de empresas o aplicaciones distribuidas basadas en la *Web*, ya que este ofrece todas las vistas necesarias para desarrollar y después desplegar estos sistemas (Booch, y otros, 2006).

Este lenguaje ofrece un estándar para detallar el modelo, incluyendo características conceptuales como procesos de negocio y las funciones y aspectos concretos como expresiones de lenguaje de programación, esquemas de bases de datos y componentes reutilizables. Se puede emplear en el desarrollo de un *software* como soporte a una metodología de desarrollo como RUP, pero no se especifica cuál metodología o proceso se debe usar (Mora, 2003).

UML ha sido seleccionado para la elaboración de todos los diagramas que describen los artefactos ingenieriles generados por la metodología de desarrollo de software seleccionada para guiar el proceso de desarrollo del módulo.

2.2.2. PHP (V 5.0)

"*PHP es un lenguaje de programación de código abierto*[6] *interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor*" (Achour, y otros, 2005). En los últimos años, PHP se ha convertido en la plataforma *web* más extendida en el mundo, funcionando en más de un tercio de los servidores *web* a través del globo. En su versión 5.0 se incorporan elementos ausentes en sus versiones anteriores

[6] Código abierto (*Open Source*) es el término con el que se conoce al *software* distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones morales y/o filosóficas las cuales destacan en

realizando una gran mejora en la programación orientada a objetos (POO[7]) la cual estaba soportada por el lenguaje a partir de su versión 3.0 pero en la que existían varios agujeros que se han intentado corregir en esta versión (Gutmans, y otros, 2004).

Por las características antes expuestas se selecciona este lenguaje de programación para la implementación de los componentes del lado del servidor del módulo a desarrollar.

2.2.3. JavaScript (V 1.5)

JavaScript es un lenguaje interpretado y muy utilizado en el ámbito de la programación para la creación de páginas *web* dinámicas, contribuyendo a mejoras significativas en las interfaces de usuario. Actualmente los navegadores *web* modernos incorporan la interpretación de código *JavaScript* embebido en el código de las páginas *web* independientemente del sistema operativo en que se utilice.

Debido a su sencillez, facilidad de aprendizaje y factibilidad, se selecciona este lenguaje para la programación de los componentes del lado del cliente del módulo a desarrollar.

2.3. Herramientas

2.3.1. Sistema Gestor de Base de Datos PostgreSQL (V 8.4)

Un Sistema Gestor de Base de Datos (SGBD) es una colección de programas que permiten al usuario crear y administrar una base de datos[8]. Es el programa o conjunto de programas que gestiona y mantiene consistente el conjunto de datos almacenados. Un SGBD relacional es aquel donde los datos almacenados cumplen una serie de normas y reglas y son percibidos por los usuarios como un conjunto de tablas interrelacionadas (Garzón Pérez, 2010).

PostgreSQL es un sistema de base de datos objeto-relacional (ORDBMS) de código abierto, descendiente de POSTGRES, versión 4.2, desarrollado por el Departamento de Informática de la Universidad de California en Berkeley. Es compatible con una gran parte del estándar SQL y ofrece muchas características modernas. Debido a la libertad de la licencia, puede ser usado, modificado y distribuido por todo el mundo libre de carga para cualquier propósito, ya sea privado, comercial o académico (Development Group, 2005). Cuenta además con la extensión PostGIS que le añade soporte para el

el llamado s*oftware* libre.

[7] Programación Orientada a Objetos, es una forma de programar que trata de modelar los procesos de programación de una manera cercana a la realidad: tratando a cada componente de un programa como un objeto con sus características y funcionalidades.

[8] Una base de datos es un fondo común de información interrelacionada para ser accedida mediante consultas.

trabajo con datos espaciales la cual posee varias funciones implementadas para el trabajo con datos geográficos.

Por lo antes expuesto, además de que la base de datos del SIGRutas está desarrollada en este sistema, se optó por usar este SGDB para dar solución al problema propuesto.

2.3.2. ExtJS (V 3.0)

ExtJS es una biblioteca o conjunto de librerías de *JavaScript* para el desarrollo de aplicaciones *web* interactivas utilizando tecnología *AJAX*, *DHTML* y *DOM*. Permite realizar interfaces de usuarios muy parecidas a las aplicaciones de escritorio, procurando así interfaces amigables y fáciles de usar. Este sistema permite a las aplicaciones web funcionar de manera asincrónica utilizando eventos y manejadores para las peticiones de las interfaces a los controladores lo cual elimina la programación tradicional para *internet*, también llamada *Web 1.0*, la cual ejecutaba el código línea a línea (Frederick, y otros, 2008).

Ajax, acrónimo de *Asynchronous JavaScript and XML* (*JavaScript* asíncrono y *XML*), es una técnica de desarrollo *web* para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente mientras se mantiene la comunicación asíncrona con el servidor. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que aumenta la interactividad, velocidad y usabilidad en las aplicaciones (Eguíluz Pérez, 2008).

El **HTML Dinámico** (*DHTML*), aporta interactividad a las páginas *web*, este tiene la ventaja de poder crear efectos que requieren poco ancho de banda a la hora de ejecutarse y son estos efectos los que aumentan la funcionalidad de la página, los cuales serían imposibles de realizar con HTML simple. Aunque muchas de las características del DHTML se podrían realizar con otras herramientas como *Java* o *Flash*, el DHTML ofrece la ventaja de que no requiere ningún tipo de *plugin* para poder utilizarlo (Powe, 2001).

El **DOM** (*Document Object Model*), es una plataforma que permite a los programas y *scripts* acceder y actualizar dinámicamente el contenido, la estructura y el estilo de los documentos. Esta estructura de objetos es generada por el navegador cuando se carga un documento (Garcia, 2009).

Se selecciona ExtJS, versión 4.0, para el desarrollo de la interfaz de usuario de la aplicación generada para la solución a la problemática propuesta, ya que la interfaz del SIGRutas está desarrollada con ExtJS pretendiendo con esto mantener la compatibilidad con el mismo además de que ExtJS brinda las siguientes ventajas:

- Provee *widgets* para el trabajo de los programadores.
- Facilita la interacción con el navegador con un Administrador de Eventos que responde a los *clic* del *mouse*, la opresión de teclas por el usuario y monitorizando eventos como cambio de tamaño de las ventanas y de la letra.
- Permite la comunicación con el servidor sin refrescar la página (AJAX).

2.3.3. MapServer (V 5.6)

Un servidor de mapas es una herramienta que permite la interacción con la información espacial almacenada en servidores de datos espaciales accesibles a través de la *web*. *MapServer* es una plataforma de código abierto para la publicación de datos espaciales y aplicaciones de cartografía interactiva para la *web* desarrollado en la década de 1990 en la Universidad de Minnesota. *MapServe*r se encuentra liberado bajo una licencia tipo MIT[9].

Entre sus principales características se encuentran (MapServer):

- Soporta los lenguajes de *scripting* y entornos de desarrollos más populares: *PHP MapScript, Phyton MapScript, Perl MapScript, Ruby MapScript, MapScript Java y .NET MapScript*;
- Funciona en los principales sistemas operativos: Windows, Linux, Mac OS X.
- Soporta múltiples normas *OGC, WMS (cliente / servidor), no transaccional WFS (cliente / servidor), WMC, WCS, Filter Encoding, SLD, GML*.
- Permite el manejo de datos en formato raster: *TIFF / GeoTIFF, EPPL7,* y muchos otros a través de *GDAL*.
- Datos en formato vectorial: *ESRI shapfiles, PostGIS, ArcSDE de ESRI, Spatial Oracle, MySQL* y muchos otros a través de *OGR*.

  Por lo anterior planteado se elije utilizar *MapServer* como servidor de mapas para la representación espacial de los datos generados por el módulo a desarrollar.

2.3.4. Visual Paradim (V 5.0)

Visual Paradigm para UML (VP-UML) es una herramienta CASE de diseño desarrollado para contribuir a la construcción de *software*. VP-UML soporta los principales estándares de modelado como Lenguaje de Modelado Unificado (UML) 2.4, *SysML, ERD, DFD, BPMN 2.0, ArchiMate 2.0*, etc. Esta herramienta

[9] Este tipo de licencia no posee *copyright*, lo que permite la modificación y distribución del software a terceros.

facilita el trabajo a los equipos de desarrollo de software en la captura de requisitos, planificación de software (análisis de casos de uso), el modelado de clases, el modelado de datos, entre otros. *Visual Paradigm* es una potente multiplataforma, y sin embargo es fácil de usar para el modelado UML, proporcionando a los desarrolladores de software una plataforma de desarrollo para construir aplicaciones de alta calidad, más rápido, mejor y barato. Se facilita la interoperabilidad con otras herramientas CASE y la mayoría de los IDE de desarrollo (Visual Paradigm).

A partir de lo antes expuesto se selecciona *Visual Paradigm* como herramienta para el modelado ya que el equipo de desarrollo ha trabajado con anterioridad con esta herramienta teniendo conocimiento sobre la misma. Además, VP-UML permite diseñar los artefactos generados por la metodología de desarrollo de software seleccionada.

2.3.5. Netbeans (V 7.0)

NetBeans IDE es un entorno de desarrollo integrado libre que soporta lenguajes como C/C++ y PHP. Este proporciona editores y herramientas integrales para *frameworks* y tecnologías como *XML, HTML, PHP, Groovy, Javadoc, JavaScript*, etc. siendo compatible con todos los sistemas operativos que soportan Java, desde Windows a Linux Entre sus características fundamentales sobresalen la capacidad de detección de errores a partir de un depurador y un selector de vistas que permite al usuario navegar por las líneas de códigos además de reportar posibles pérdidas de información (Oracle Corporation). Se selecciona este IDE ya que es robusto, libre e intuitivo además por su capacidad para integrar con otras aplicaciones lo que posibilita un posible avance en el desarrollo de aplicaciones.

2.3.6. GENESIG (V 1.5)

GENESIG es una plataforma tecnológica libre y de código abierto orientada a la web que permite satisfacer demandas básicas de información espacial, en cuanto a su almacenamiento, análisis, y representación. Esta plataforma soporta una gran cantidad de funcionalidades relacionadas con la gestión de datos espaciales. Es desarrollada por la Universidad de las Ciencias Informáticas y permite además realizar personalizaciones de Sistemas de Información Geográfica (Gracia Aguila, 2011). Esta herramienta se utilizó para la integración del módulo con el SIGRutas.

2.4. Metodología de desarrollo de software

Las Metodologías de Desarrollo de Software son una serie de procedimientos, técnicas, herramientas y soporte documental aplicados al desarrollo de un producto software. Dichas metodologías pretenden guiar

a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software (Carrillo Pérez, y otros, 2008).

2.4.1. Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo (**RUP**, por sus siglas en inglés *Rational Unified Process*) constituyó el resultado de tres décadas de desarrollo y trabajo práctico. Es un proceso de desarrollo de software que ofrece un conjunto de actividades o flujos de trabajo para transformar los requerimientos de un usuario en un sistema informático separándolas en fases **(Figura 4)**.

Las mejores prácticas que RUP propone son (Booch, y otros, 2000):

- Desarrollo iterativo e incremental del software: Durante el ciclo de vida del desarrollo de un software, es mucho más fácil y práctico dividirlo en problemas más pequeños, donde cada uno de estos es llamado iteración, que resulta en un incremento. Las iteraciones referencian los distintos pasos seguidos en los flujos de trabajo y los incrementos al crecimiento del producto final. Esto trae como consecuencia la realización de un proceso de desarrollo de larga duración por lo que es muy útil en proyectos de alta complejidad.
- Proceso centrado en la arquitectura: El objetivo de la arquitectura de software es el de dotar a los desarrolladores de una imagen completa del producto, desde diferentes puntos de vista, antes de comenzar la construcción. La arquitectura de software tiene en cuenta los aspectos estáticos y dinámicos más significativos, las necesidades de la empresa y clientes reflejadas en casos de uso y muchos otros factores como la plataforma en que funciona (*Software*, *Hardware*, etc.…), componentes reutilizables, requisitos no funcionales, etc.
- Proceso dirigido por casos de uso: Todo sistema de *software* se hace para brindar un servicio a un grupo de usuarios determinados. El término usuario representa alguien o algo que interactúa con el sistema que se desarrolla. Un caso de uso es un fragmento de funcionalidad que proporciona al usuario un resultado importante. La unión de todos los casos de uso, conforman el modelo de casos de uso, que describe todas las funcionalidades del sistema, guiando de esta manera el diseño, implementación y prueba.

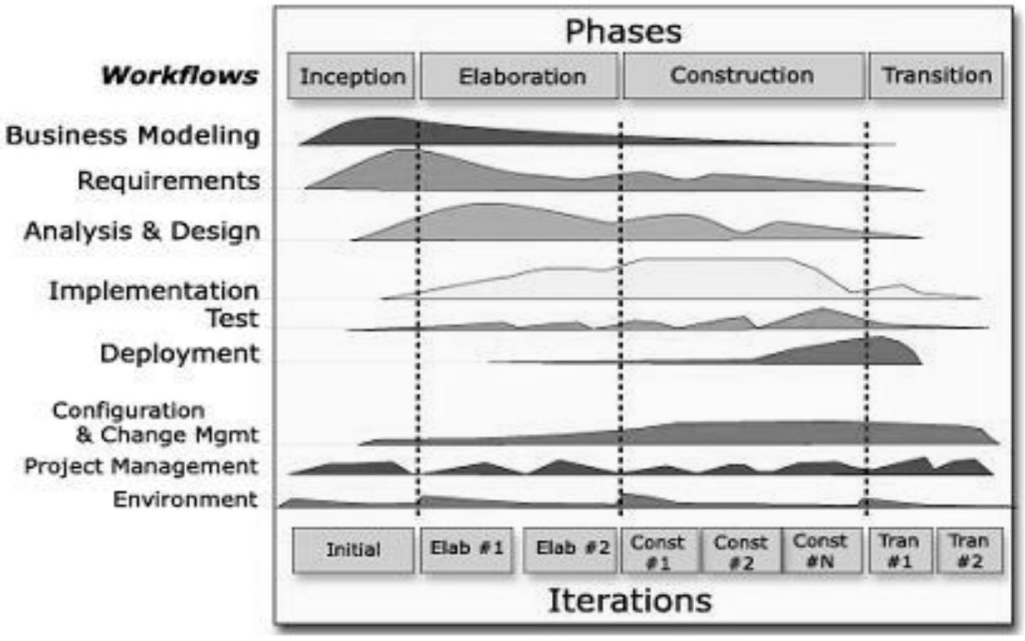

Figura 4: Flujo de Trabajo de la Metodología RUP (Rational Software, 2012).

RUP es una de las metodologías estándar más eficaz para la implementación y documentación de un sistema, pero plantea la creación de una cantidad de documentación excesiva, mucho más de la necesaria para tener correctamente documentado el módulo a desarrollar, además está diseñada para la realización de proyectos de larga duración, por lo que RUP no es la metodología que mejor se acopla al proceso de desarrollo.

2.4.2. Metodología OpenUP

OpenUP es un marco de trabajo **(Figura 5)** para procesos de desarrollo de *software*, liberado por *Eclipse Process Framework* (EPF) y construido sobre una donación realizada por IBM del *Basic Unified Process*. Entre sus principales característica se encuentra (Balduino, 2007):

- Preservación de la escencia del Unified Process (iterativo e incremental, desarrollo dirigido por casos de uso y centrado en la arquitectura).
- Solo se incluye el contenido fundamental (genera menos de veinte artefactos en comparación con RUP) y necesario sin dejar de ser un desarrollo completo y flexible ya que puede ser usado para añadir o ajustar el contenido del proceso.
- Es una metodología para guiar el desarrollo de aplicaciones pequeñas.

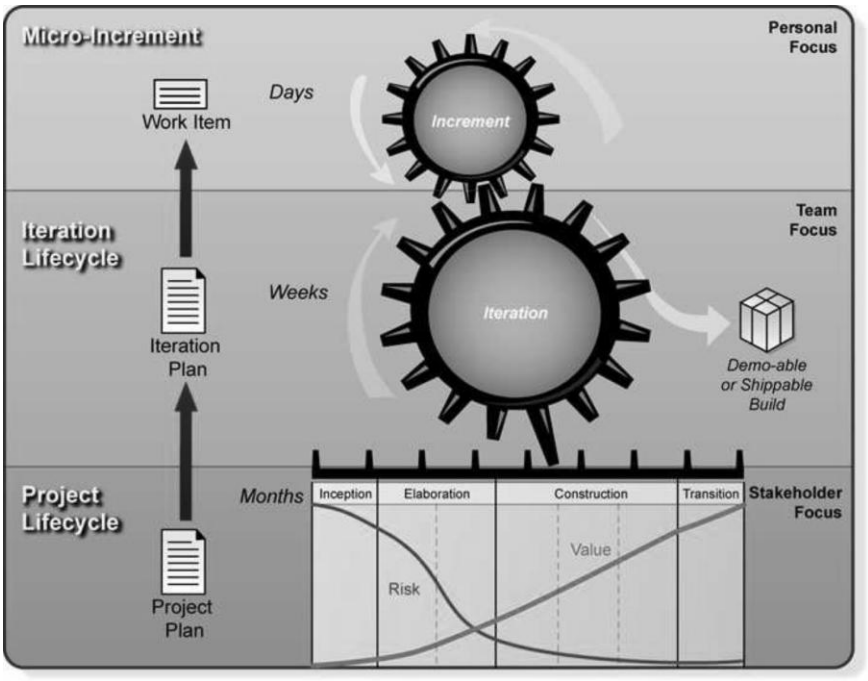

Figura 5: Organización del trabajo y el enfoque contenido en OpenUP (Balduino, 2007).

A partir de las características antes expuestas de la metodología OpenUP, y teniendo en cuenta la envergadura del módulo a desarrollar, se decide emplear esta ya que se considera que sus artefactos y pautas son suficientes para guiar el desarrollo del módulo propuesto además de que permitir añadir tantos artefactos propios como se consideren necesarios, o ignorar los predefinidos y utilizar cualquier tipo de artefactos definidos por el equipo de desarrollo. Con la documentación generada por esta metodología es posible documentar de manera correcta y entendible el desarrollo del módulo además de que esta metodología se acopla correctamente a los conceptos definidos por RUP siendo esto importante ya que el SIGRutas se desarrolló a partir de RUP.

2.5. Conclusiones del capítulo

Una correcta selección de metodología de software en conjunto con las herramientas y tecnologías adecuadas a un problema puede facilitar en gran medida el trabajo de encontrar una solución factible a dicho problema. La selección de la metodología OpenUP está acorde a la necesidad del equipo de desarrollo posibilitando una gestión ágil del proceso de desarrollo y la generación de la documentación necesaria para el entendimiento de la solución propuesta. Se ha procurado la selección de herramientas y

tecnologías libres continuando con las políticas de la universidad de una migración hacia este tipo de sistemas para obtener una liberación tecnológica.

Se puede concluir que con los datos aportados por la investigación teórica, y los principales elementos de las herramientas a utilizar, se cuenta con suficientes elementos para solucionar la problemática planteada en este documento.

CAPÍTULO 3: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

3.1. Introducción

En el siguiente capítulo se describen los artefactos suficientes para documentar el desarrollo de la aplicación, generados por la metodología de software seleccionada OpenUP, aprovechando su característica de ser configurable al contexto. Se confecciona el modelo de dominio y el glosario de términos asociado a este, así como el diagrama de casos de uso, además se selecciona y clasifican los requisitos que debe cumplir la aplicación.

3.2. Modelo de dominio

El Modelo de Dominio es una expresión del lenguaje definido por los expertos en el dominio del negocio, llamado Lenguaje Ubicuo, que está centrado en como dichos expertos articulan el trabajo, siendo este modelo un marco dentro del cual se debe diseñar la aplicación. Es importante una concepción correcta y un entendimiento de las principales características de la aplicación para evitar errores en la concepción de los requisitos funcionales, siendo esta la principal función del Modelo de Dominio (de la Torre LLorente, y otros, 2010). El modelo de dominio del módulo desarrollado cuenta con doce clases **(Figura 6)** interrelacionadas entre sí para brindar una mejor comprensión de la problemática planteada cuya descripción se realiza mediante un diagrama de clases UML.

3.2.1. Diagrama de clases del dominio

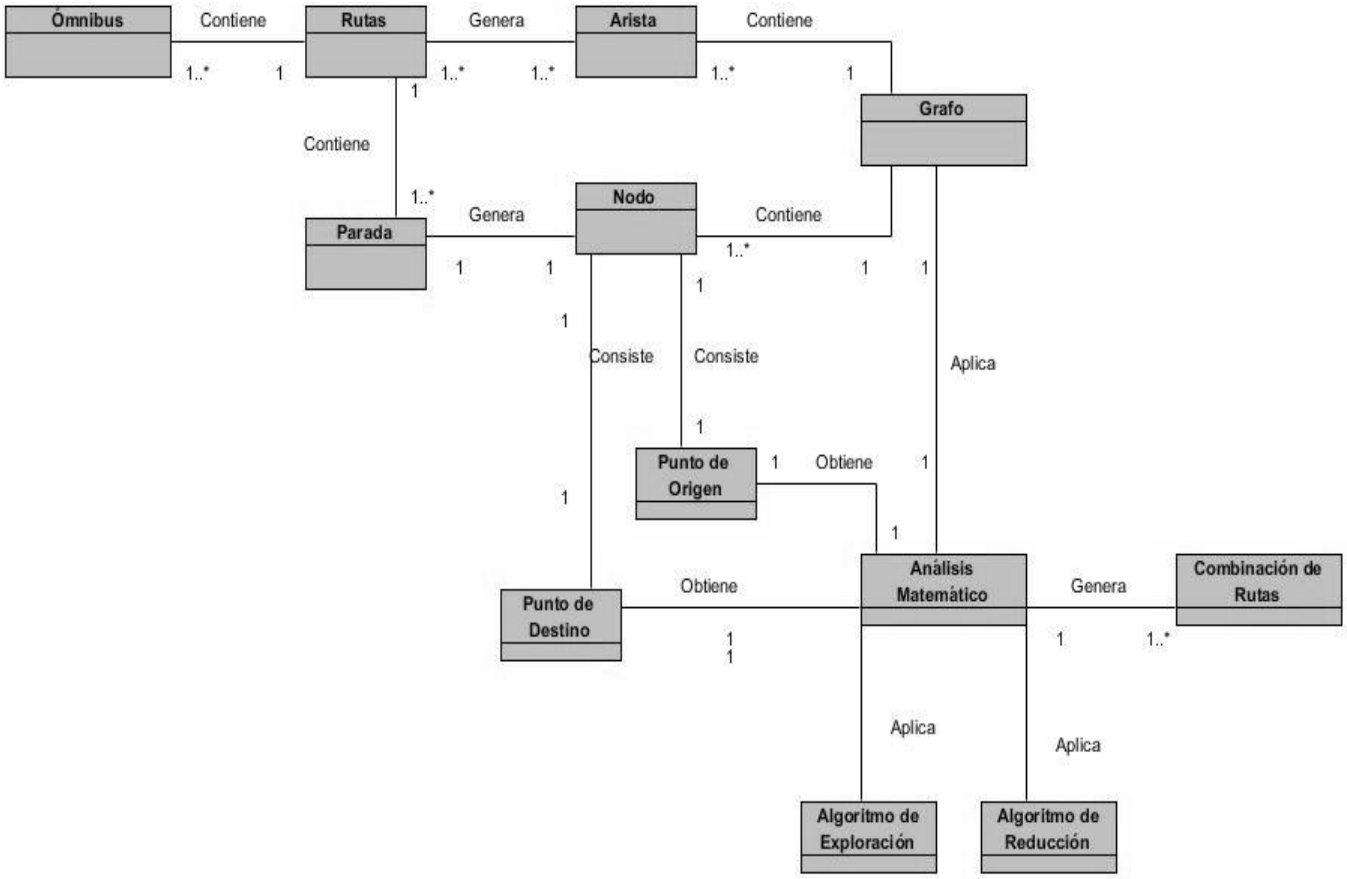

Figura 6: Modelo de Dominio.

3.2.2. Descripción del modelo de dominio

Un ómnibus tiene asociado varias rutas a través de La Habana, estas rutas poseen paradas ubicadas en las calles de la ciudad por las cuales pasa dicha ruta. A partir de las rutas y las paradas se genera un grafo donde las paradas representan nodos y las rutas o calles que los unen las aristas. Se selecciona un punto de partida y un punto de origen de este grafo, los cuales consisten en dos nodos del grafo construido, se aplica un algoritmo de exploración a través del grafo obteniéndose el camino mínimo entre dichos puntos.

3.2.3. Glosario de términos del dominio

Ómnibus: Vehículo de gran capacidad para el transporte público.

Ruta: Combinaciones de vías o calles para representar el espacio a recorrer dentro de una localidad.

Parada: Lugar donde debe detenerse de manera obligatoria un ómnibus para el trasbordo de pasajeros.
Calle: Espacio urbano lineal que permite la circulación de personas y vehículos.
Grafo: Representación matemática de redes de transporte.
Nodo: Vértices de un grafo que representa paradas e intercepciones de calles en la red de transporte.
Arista: Conexiones entre nodos que representa calles o una porción de una ruta en la red de transporte.
Análisis Matemático: Realiza algoritmos de intersección, reducción y exploración sobre grafos, generando combinaciones de caminos entre dos puntos.
Punto de Origen: Nodo del grafo que representa el punto de partida para un recorrido en la red de transporte.
Punto de Destino: Nodo del grafo que representa el punto de llegada para un recorrido en la red de trasporte.
Combinación de Rutas: Representa las combinaciones de rutas entre dos puntos de la red de transporte.

3.3. Levantamiento de requisitos

El objetivo del levantamiento de requisito es que el equipo de desarrollo comprenda lo que quiere el cliente, analizando sus necesidades y negociando una solución razonable que satisfaga ambas partes, especificando la solución sin ambigüedad para evitar inconformidades por parte del cliente, validando la especificación y gestionando estos requisitos para que se transformen en un sistema operacional (Pressman, 2005).

3.3.1. Requisitos funcionales

Estos requisitos representan las funcionalidades que debe realizar el sistema para dar solución a distintas problemáticas en determinadas situaciones. Deben ser descritos correctamente ya que son la piedra angular de la implementación de la aplicación.

A continuación se definen los requisitos funcionales que debe cumplir la aplicación propuesta para dar solución a la problemática planteada:

- **RF 1 Seleccionar punto de origen y destino**: El sistema debe ser capaz de mostrar una interfaz donde el usuario pueda seleccionar un punto de origen y un punto de destino en el mapa.
- **RF 1.1 Listar direcciones de puntos relevantes:** El sistema debe listar direcciones de puntos geográficos relevantes de la ciudad de La Habana.

- **RF 1.2 Capturar punto de origen:** El sistema debe ser capaz de capturar como punto de partida todas las rutas que tienen como salida la universidad.
- **RF 1.3 Capturar punto de destino:** El sistema debe ser capaz de capturar un punto seleccionado por el usuario identificarlo como punto de destino.
- **RF 2 Obtener datos de rutas**: El sistema debe obtener información asociada a las rutas de los ómnibus.
- **RF 3 Obtener datos de calles**: El sistema debe obtener información asociada a las calles de la ciudad de La Habana.
- **RF 4 Generar grafo de rutas:** El sistema debe ser capaz de generar un grafo que represente las rutas que debe recorrer un usuario para trasladarse entre dos puntos ya sea a través de un ómnibus o caminando.
- **RF 5 Reducir grafo de rutas:** El sistema debe reducir el grafo creado para eliminar información redundante.
- **RF 6 Aplicar análisis matemático sobre el grafo de rutas:** El sistema debe ejecutar una exploración sobre el grafo generado para obtener combinaciones de caminos entre dos puntos dentro de este espacio de búsqueda.
- **RF 6.1 Aplicar algoritmo de exploración sobre el grafo:** El sistema debe ser capaz de aplicar un algoritmo de exploración de grafo en profundidad.
- **RF 6.2 Aplicar técnica Bactracking**: El sistema debe ser capaz de realizar la técnica Backtracking para explorar el grafo.
- **RF 6.3 Aplicar algoritmo de Dijkstra**: El sistema debe calcular caminos mínimos en el grafo a partir del algoritmo de Dijkstra.
- **RF 7 Mostrar datos de la ruta calculada:** El sistema debe mostrar una interfaz con la descripción de las combinaciones de rutas tramo a tramo entre dos puntos del mapa.
- **RF 8 Representar la ruta calculada:** El sistema debe mostrar una representación geográfica sobre el mapa de La Habana de la ruta calculada.

3.3.2. Requisitos no funcionales

Los requisitos no funcionales son cualidades y propiedades que debe cumplir un sistema para lograr que sea funcional clasificados en grupos de acuerdo a la función que realizan en la aplicación. Estos garantizan la seguridad, fiabilidad, acoplamiento y rapidez con la que el sistema funciona. Suelen ser

fundamentales en lograr resultados positivos al término de la aplicación. Generalmente se vinculan a requisitos funcionales ya que es necesario primero conocer lo que el sistema debe hacer para luego definir cómo debe comportarse, cuán grande será o a qué velocidad debe operar (Pressman, 2005).

Usabilidad: El sistema podrá ser usado por persona con conocimientos básicos en el manejo de computadoras contando con interfaces intuitivas para un mejor desenvolvimiento de los usuarios. Podrá ser utilizado, una vez desplegado, las veinticuatros horas del día siete días a la semana contándose con servidores de respaldo para evitar una posible caída del sistema.

Fiabilidad: La información manejada por el sistema estará protegida para evitar posibles modificaciones de la misma por los usuarios a partir de un módulo de autenticación. La información brindada a los usuarios será la correcta para cada petición siendo esto validado en las pruebas realizadas al sistema.

Eficiencia: El tiempo de respuesta estará dado por la cantidad de información a procesar. La complejidad temporal de los algoritmos implementados está determinada por la cantidad de nodos contenidos en el grafo. Se decidió la utilización del algoritmo de Dijkstra para el cálculo de camino mínimo ya que es el más eficiente de los algoritmos analizados.

Restricciones de Diseño: El sistema contará con un diseño sencillo, con pocas entradas, donde no sea necesario mucho entrenamiento para utilizarlo. Se deben emplear los estándares establecidos en cuanto a diseño de interfaces y codificación definidos y especificados en el expediente de proyecto del SIGRutas. Se debe lograr un producto altamente configurable y extensible, teniendo en cuenta que se desarrollará sobre la Plataforma GeneSIG y que constituye una plataforma de desarrollo para ser personalizada como aplicaciones a la medida, pudiéndose incorporar a éste nuevas funcionalidades.

Los requisitos no funcionales del módulo a desarrollar están relacionados directamente con los requisitos no funcionales del sistema SIGRutas al cual será integrado el mismo, estando estos definidos y descritos completamente en el expediente de proyecto del SIGRutas (**Expediente de proyecto SIGRutas/Ingeniería/Requisitos/**).

3.4. Modelo del sistema

El modelo del sistema representa una visión, ya sea global o específica, que define los procesos que darán respuesta a las necesidades del sistema. Además debe representar el comportamiento de estos procesos y las bases que dan lugar a estos comportamientos. En este modelo se representan todas las entradas y salidas que permitirán al ingeniero entender el funcionamiento del sistema (Pressman, 2005).

3.4.1. Actores del sistema

Los actores del sistema son los trabajadores del negocio que realizan las tareas a automatizar. Además, se pueden considerar como actores del sistema a los actores del negocio siempre y cuando interactúen con el sistema.

Tabla 1: Actores del Sistema

Actor del Sistema	Descripción
Usuario	Interactúa directamente con el sistema. Es el encargado de suministrar los datos al sistema para obtener un resultado para su propio beneficio.

3.4.2. Casos de uso del sistema

Los casos de usos del sistema son un fragmento de funcionalidad que el sistema debe realizar para ofrecer un resultado de valor al usuario. Estos configuran una parte importante del análisis de requisitos, que pueden usarse de forma independiente de la implementación tecnológica que se seleccione (Pressman, 2005).

Los casos de uso extraídos del análisis para el sistema a desarrollar son (Figura 7):

- Generar Ruta.
- Visualizar Ruta.

3.4.3. Diagrama de casos de uso

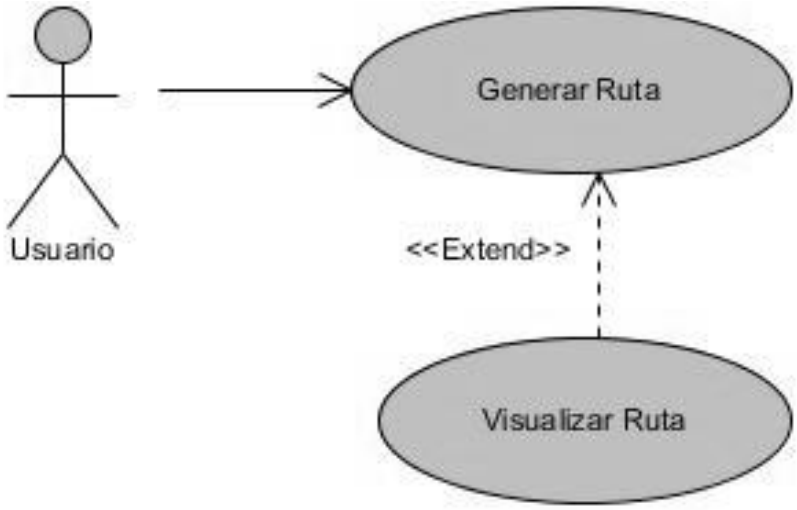

Figura 7: Diagrama de Casos de Usos.

3.4.4. Descripción textual de casos de uso

Tabla 2: Caso de Uso "Generar Ruta".

Caso de Uso	Generar Ruta.
Actores	Usuario (Inicia)
Resumen	El caso de uso inicia cuando el usuario selecciona la opción de "Encontrar Combinaciones de Rutas". El sistema muestra una interfaz para la selección de dos puntos entre los que se encontrará la ruta.
Precondiciones	
Referencias	RF 1, RF 1.1, RF 1.2, RF 1.3, RF 2, RF 3, RF 4, RF 5, RF 6, RF 6.1, RF 6.2, RF 6.3, RF 7
Prioridad	Crítica
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
Selecciona la opción "Obtener combinaciones de rutas" en la interfaz principal.	Obtiene información de los puntos de interés de la ciudad de La Habana. Muestra una ventana con el listado de puntos para que el usuario selecciones los que desea. Punto de Origen (ver sección "Seleccionar Punto de Origen") Punto destino (ver sección "Seleccionar punto de Destino")
Selecciona la opción "Enviar".	El sistema envía los puntos seleccionados por el usuario. El sistema genera un grafo a partir de un conjunto de aristas contenidos en la base de datos (ver sección "Generar Grafo", **Anexo #1/ Tabla 13**). El sistema calcula el camino entre los dos puntos seleccionados sobre el grafo generado (ver sección "Calcular Ruta", **Anexo #1/ Tabla 12**). . El sistema muestra información de cada tramo de la ruta calculada especificando: "Desde": punto de origen o trasbordo, "Hasta": punto de destino o trasbordo, "Modo": ruta utilizada en ese tramo, ya sea un ómnibus o caminando.
Sección "Seleccionar Punto de Origen"	
Acción del Actor	**Respuesta del Sistema**
	El sistema despliega en un campo el listado de los puntos definidos como origen.

Selecciona un Punto.	Valida que el campo no esté vacío. Identifica la selección como punto de origen y la guarda.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
No selecciona ningún punto.	Muestra mensaje de error "Seleccionar Punto de Origen".
Sección "Seleccionar Punto de Destino"	
Acción del Actor	**Respuesta del Sistema**
	El sistema despliega en un campo el listado de los puntos definidos como destino.
Selecciona un Punto.	Valida que el campo no esté vacío. Identifica la selección como punto de destino y la guarda.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
No selecciona ningún punto.	Muestra mensaje de error "Seleccionar Punto de Destino".

La descripción textual del caso de uso "Visualizar Ruta" se encuentra recogida en el **Anexo #1**.

3.5. Descripción de la arquitectura

El diseño de la arquitectura del sistema es un proceso por el cual se define la solución a los requisitos técnicos y funcionales del mismo. Este proceso se encarga de definir qué componentes forman el sistema, como se relacionan y como realizan una funcionalidad del sistema. Durante la selección de los patrones arquitectónicos que conformaran la arquitectura del sistema se debe tener en cuenta el entorno donde será desplegado el mismo, así como impactaría un cambio en la arquitectura una vez que sea desplegado (de la Torre LLorente, y otros, 2010). La arquitectura seleccionada para el desarrollo del módulo está relacionada estrechamente con la arquitectura definida en el SIGRutas, por lo que se tuvo que tener en cuenta esta situación para la selección de los estilos y patrones arquitectónicos para poder integrar el componente al SIGRutas.

3.5.1. Estilos arquitectónicos

Cliente-Servidor

Este estilo arquitectónico define una relación entre dos aplicaciones en las cuales una de ellas (cliente) envía peticiones a la otra (servidor). Este es un estilo para sistemas distribuidos el cual puede usar un amplio rango de protocolos y formatos de datos para establecer la comunicación entre el cliente y el servidor. La interacción del usuario con el sistema se realiza a través del cliente, generalmente utilizando interfaces, para generar las peticiones, las cuales son enviadas al servidor en espera de una respuesta para su procesamiento. Este estilo ofrece una mayor seguridad en cuanto a la protección de la información ya que esta se encuentra centralizada del lado del servidor, facilitando esta característica el acceso y actualización eficaz de la información. Este estilo facilita además el mantenimiento del sistema ya que los roles y funcionalidades se distribuyen entre los distintos servidores por lo que un error en los mismos no afectaría al cliente (de la Torre LLorente, y otros, 2010).

Este estilo se utiliza en el SIGRutas en general ya que es el más común para el desarrollo de aplicaciones web y su uso está limitado a un área LAN controlada y la plataforma GENSIG, donde fue desarrollado, utiliza este estilo. Además, es necesario el acceso de varios clientes a la información que brinda el SIGRutas lo que reduce el costo de la aplicación ya que en un solo dispositivo servidor se pueden procesar todas las peticiones de los dispositivos cliente.

3.5.2. Patrones arquitectónicos

Patrón arquitectónico orientado a objeto

Este patrón describe al sistema como un conjunto de objetos que interactúan entre sí haciendo énfasis en la reutilización a partir de la modularidad, la encapsulación, la herencia y el polimorfismo. Estos objetos son independientes y poco acoplados, comunicándose entre ellos a través de interfaces permitiendo enviar y recibir mensajes (Pressman, 2005).

Patrón de arquitectura basado en componentes

Este patrón describe el diseño del sistema como un conjunto de componentes que poseen interfaces bien definidas y que interactúan entre sí para realizar las funcionalidades del sistema. Plantea que los componentes debes estar diseñados de la manera más independiente posible evitando que puedan afectar a otros componentes durante su despliegue. Entre sus principales características se encuentran al reusabilidad ya que los componentes son independientes del contexto pudiéndose emplear en otros sistemas (de la Torre LLorente, y otros, 2010).

El módulo será desarrollado a partir de estos patrones ya que el SIGRutas, al cual debe ser integrado, utiliza la plataforma GENESIG la cual posee este tipo de arquitectura. La estructura de esta plataforma está compuesta por *plugins* siendo cada uno de ellos un componente independiente del sistema. El módulo a desarrollar debe ser integrado como un *plugin* al sistema SIGRutas para la ejecución de una funcionalidad específica.

3.6. Modelo del diseño

El flujo de trabajo "análisis" explica detalladamente los requisitos a partir de un refinamiento de los casos de uso del sistema, funcionando esta práctica como una primera aproximación al modelo del diseño. No obstante, durante el diseño tiende a cambiar la estructura extraída del análisis ya que en este se deben tener en cuenta herramientas y tecnologías que serán usadas en la implementación. Ya que la metodología usada para el desarrollo del módulo es un proceso configurable que permite una adaptación al contexto de la problemática donde va a ser aplicada, se decide prescindir de la realización del modelo del análisis atendiendo a las siguientes características del sistema desarrollado:

- La descripción del dominio de la problemática es amplia y simple.
- Se definieron requisitos funcionales simples y entendibles contándose con una gran comprensión de los mismos.
- No se posee un número elevado de casos de usos y la complejidad de los mismos no es difícil al entendimiento del equipo de desarrollo.
- Se tiene conocimiento del lenguaje de programación sobre el que se realizará la aplicación.
- Se reducirían los gastos en cuanto a tiempo y recursos.

Decidiéndose entonces la eliminación del flujo "análisis" se da paso a la realización del diseño cuyo objetivo es traducir los requisitos en una representación del software. En este se establecen las estructuras de datos, la arquitectura general del software y representaciones generales de interfaz y algoritmos a partir de patrones ya definidos constituyendo una abstracción de la implementación. En el modelo del diseño los casos de usos son representados por clases del diseño y sus objetos (Pressman, 2005).

3.6.1. Patrones de diseño

Los patrones de diseño, utilizando métodos y estructuras probados en diferentes sistemas, guían mediante expresiones y elementos el proceso de desarrollo de cualquier aplicación informática. Estos se

utilizan como un conjunto estructural de relaciones e interrelaciones entre clases, componentes y módulos para obtener una abstracción de la implementación de la solución a desarrollar (Larman, 1999).

Patrones GRASP

Los patrones GRASP describen principios fundamentales para la asignación de responsabilidades a objetos, expresados en forma de patrones (Larman, 1999). Los patrones GRASP aplicados al diseño de la solución propuesta son:

- **Experto:** este patrón plantea que la responsabilidad de realizar una operación debe recaer sobre la clase que cuenta con la información necesaria para realizar dicha operación (Larman, 1999). En el módulo desarrollado este patrón se manifiesta en las clases ServerCombinaciones y DijkstraGrafo donde las respuestas que se deben obtener del sistema se realizan en dichas clases a partir de la información que manejan las mismas.
- **Creador:** este patrón define que clase es la encargada de la instanciación de otras a partir de la usabilidad en dicha clase de las funcionalidades brindadas por las instancias (Larman, 1999). En la solución propuesta este patrón se pone de manifiesto en la clase ClientCombinaciones donde se realizan instancias a otras clases para hacer uso de sus funcionalidades y poder obtener la respuesta deseada.
- **Bajo Acoplamiento:** su función es la de reducir las dependencias entre clases a partir de las relaciones que existen entre ellas. Existe bajo acoplamiento cuando las clases del sistema no dependen para su funcionamiento de un número demasiado grande de instancias de otras clases (Larman, 1999). En la aplicación desarrollada se pone de manifiesto esto ya que no existe una gran cantidad de clases y la clase ServerCombinaciones solo depende de la clase GrafoDijkstra no dependiendo esta de ninguna otra clase.
- **Alta Cohesión:** permite disminuir la complejidad de las clases realizando el diseño de manera que las clases solo contengan las responsabilidades mínimas necesarias para su funcionamiento. Este patrón permitirá tener clases fáciles de mantener, entender y reutilizar.

Patrones GoF

- **Command:** encapsula una petición en un objeto, pudiendo realizar operaciones como crear una cola de objetos, gestionarla y poder deshacer las operaciones. Entre las ventajas que posee se

encuentran la simplificación de llamadas, estructuración del sistema y que permite realizar una cola de operaciones. (Larman, 1999)

Se utiliza en el proceso de petición mediante la Interfaz Gráfica de Usuario (GUI) al sistema de una información cualquiera por un cliente. Esto se evidencia en la clase AjaxHelper que es la encargada de manejar las peticiones mediante la interfaz.

- **Singleton:** la aplicación de este patrón asegura la existencia de una única instancia de un objeto y la creación de un mecanismo de acceso global del mismo (Gamma, y otros, 1994). En la aplicación desarrollada este patrón se pone de manifiesto en la utilización de una única instancia a una clase para el acceso a la información contenida en la base de datos

3.6.2. Diagrama de clases del diseño

En los diagramas de clases del diseño se representan las clases del diseño, sus relaciones, atributos y métodos. En este epígrafe se muestra el diagrama de clases del diseño del caso de uso “Generar Ruta”, el resto de los diagramas se encuentran en el **Anexo 2.**

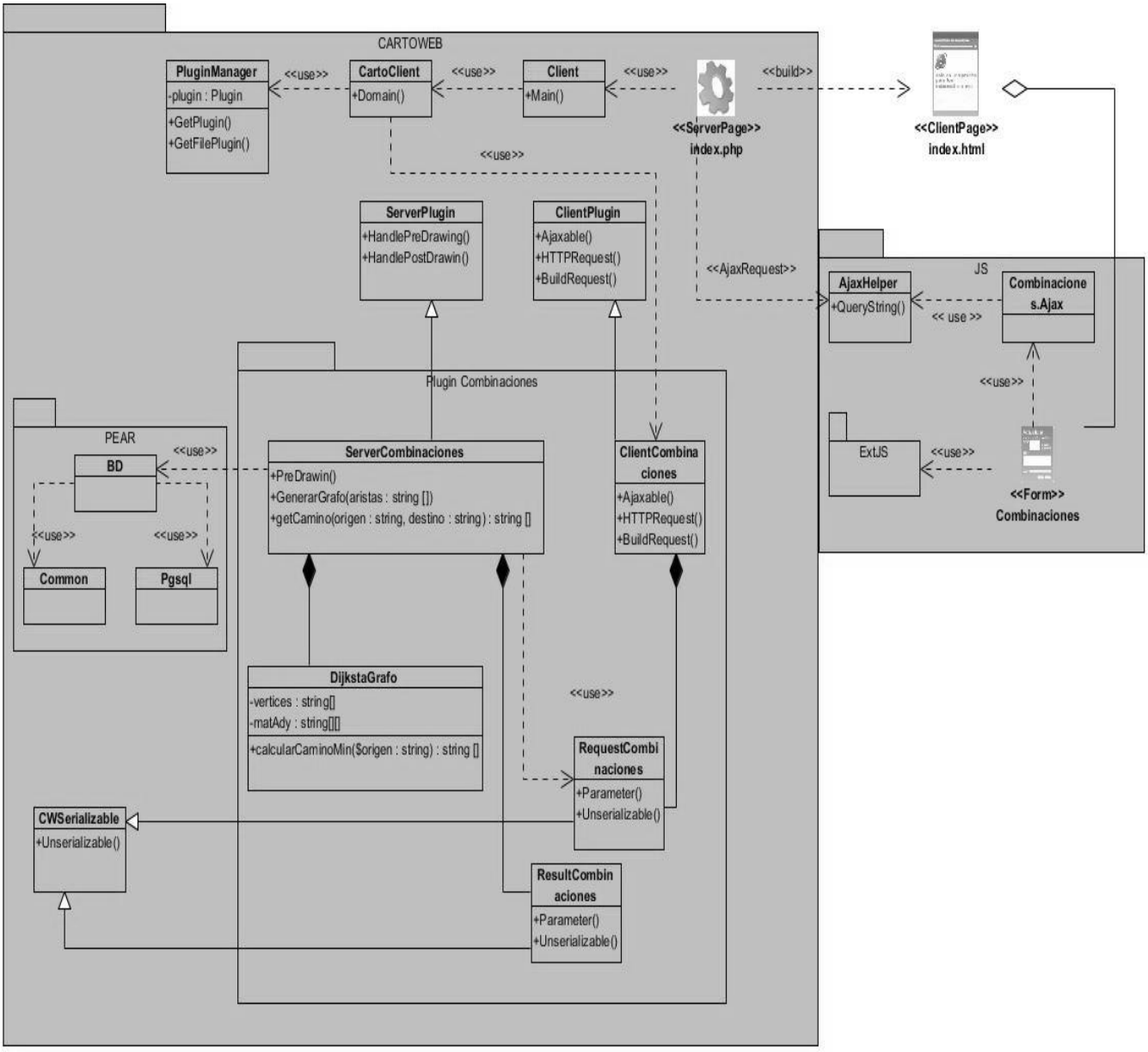

Figura 8: Diagrama de clases del diseño CU "Generar Ruta".

3.6.3. Descripción de las clases del diseño

Los diagramas de clases del diseño de los casos de uso “Generar Ruta” y “Visualizar Ruta” poseen clases en común, por lo que en este epígrafe se explicarán las clases asociadas solo al caso de uso “Generar Rutas” **(Tabla 3 y 4)**. La descripción del resto de las clases se encuentra en el **Anexo 3.**

Tabla 3: Descripción de la clase "ServerCombinaciones"

Nombre: ServerCombinaciones.	
Tipo de clase: Controladora	
Responsabilidades	
GetOrigenes(): string[]	Obtiene los datos de los puntos de orígenes.
GetDestinos(): string[]	Obtiene los datos de los puntos de destino.
GenerarGrafo(): string[]	Se encarga de generar un grafo a partir de un conjunto de aristas.
getCamino(): string[]	Se encarga de obtener la información de los tramos que conforman el camino mínimo entre los puntos de origen y destino.

Tabla 4: Descripción de la clase "DijkstraGrafo".

Nombre: DijkstraGrafo.	
Tipo de clase: Controladora	
Atributos	Tipo
-vertices	string[]
-matAdy	double[][]
Responsabilidades	
GetCaminoMin(origen:string): string[]	Aplica el algoritmo de Dijkstra al grafo para obtener el camino mínimo desde el punto dado como origen.

3.7. Conclusiones del capítulo

Los artefactos generados por la metodología de software brindan una mejor comprensión del sistema dotando al equipo de desarrollo de una guía necesaria para el desarrollo del proceso de implementación. El modelo de dominio provee una mejor comprensión de las entidades de mayor relevancia dentro del contexto de desarrollo del sistema. La definición de los requisitos que debe cumplir el sistema ilustra de manera general el funcionamiento interno del sistema expresado en casos de uso. Los diagramas de clases del diseño constituyen herramientas muy útiles cuando se comienza la fase de implementación del software, ya que constituyen una guía necesaria para la organización del trabajo de los programadores del equipo de desarrollo.

Con los elementos descritos en este capítulo se puede comenzar a desarrollar el proceso de implementación y construcción del módulo, teniendo en cuenta que se cumplan todas las funcionalidades que el sistema debe manejar.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

4.1. Introducción

Después de haber realizado todo el proceso de diseño del módulo se pasa a la fase de implementación cuyo propósito principal es desarrollar la arquitectura y el sistema como un todo. La implementación se realiza de manera incremental lo cual da la posibilidad de ir manejando paso a paso la realización del sistema. Además, esta fase constituye el principio de la etapa de pruebas ya que a cada entregable se le realizan pruebas de integración y sistema (Jacobson, y otros, 2004). En el siguiente capítulo se expresa el modelo de implementación y de pruebas correspondiente al módulo desarrollado. Se definen los artefactos modelo de despliegue y diagrama de componentes, además se realizan los casos de prueba para comprobar la funcionalidad de la aplicación.

4.2. Modelo de implementación

El modelo de implementación describe como los elementos del diseño se implementan en término de componentes tales como ficheros de código fuente, ejecutables entre otros. Además este modelo describe también la organización de estos componentes a partir de los mecanismos de estructuración que brinda el entorno de implementación y el lenguaje de programación utilizado así como la relación existente entre dichos componentes. El modelo de implementación se representa a partir de un sistema de implementación definido por el subsistema de nivel superior del modelo pudiéndose utilizar otros subsistemas para hacerlo más manejable (Pressman, 2000).

4.2.1. Modelo de despliegue

Los diagramas de despliegue **(Figura 9)** muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos.

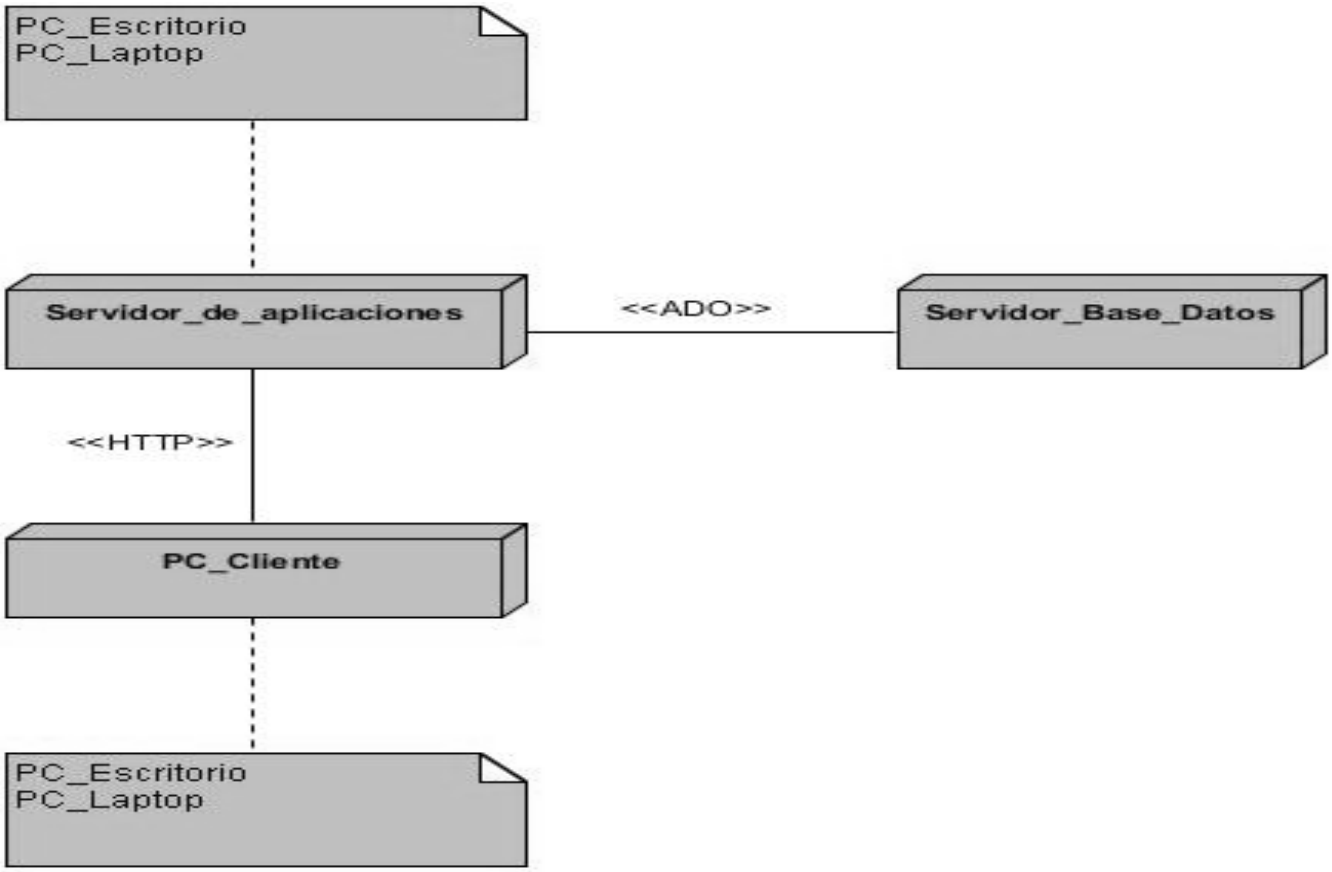

Figura 9: Diagrama de despliegue.

4.2.2. Diagrama de componentes

Un componente representa el empaquetamiento físico de los elementos de un modelo, de manera análoga a las clases en el modelo de diseño. El diagrama de componentes para el caso de uso “Visualizar Ruta” se encuentra recogido en el **Anexo 4**.

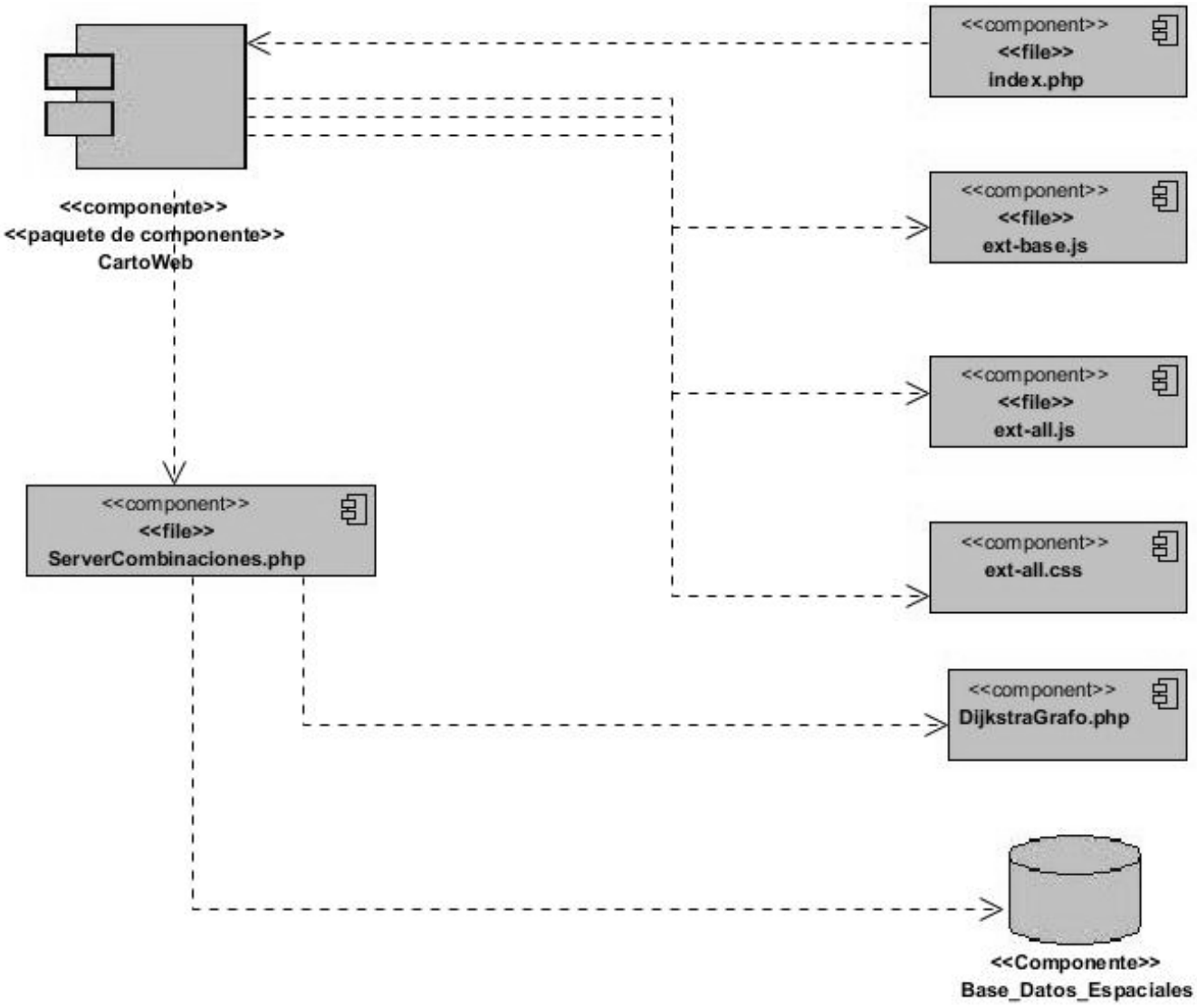

Figura 10: Diagrama de Componentes CU "Generar Ruta".

4.3. Modelo de pruebas

El modelo de prueba constituye una descripción de cómo se prueban los componentes del modelo de implementación. En este flujo de trabajo se valida el resultado de la implementación probando cada construcción así como las versiones finales del software (Jacobson, y otros, 2004).

4.3.1. Casos de pruebas funcionales

Las pruebas funcionales o de caja negra constituyen métodos de diseño de casos de pruebas que se llevan a cabo principalmente sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa. Este tipo de prueba pretende demostrar que las funcionalidades del software son operativas y que se produce una salida correcta a partir de los datos de entrada insertados en el sistema (Pressman, 2005). Teniendo en cuenta estas características se decidió realizar este tipo de prueba al caso de uso “Seleccionar Puntos” ya que este opera íntegramente sobre la interfaz del sistema **(Tabla 4)**.

Tabla 5: Caso de prueba caso de uso "Generar Ruta". Secciones.

Secciones a probar en el caso de uso		
Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**
SC 1: Mostrar Datos de Ruta.	EC 1.1: Selección de la opción “Obtener Combinaciones de Rutas”.	El sistema muestra una interfaz con los campos a seleccionar por el usuario: Origen. Destino.
	EC 1.2: Desplegar Origen.	El sistema despliega una lista con los puntos de origen.
	EC 1.3: Desplegar Destino.	El sistema despliega una lista con los puntos de destino.
	EC 1.4: Contraer Origen.	El sistema contrae la lista de los puntos de origen.
	EC 1.5: Contraer Destino.	El sistema contrae la lista de los puntos de destino.
	EC 1.6: Campos Vacíos.	El sistema muestra un mensaje especificando los campos que deben ser llenados por el usuario.
	EC 1.7: Mostrar Datos de Ruta.	El sistema muestra una tabla con los datos de cada tramo que conforma la ruta mínima entre los puntos de origen y destino.
	EC 1.8: Cancelación de la funcionalidad.	El sistema oculta la ventana “Combinaciones de Rutas”.

Tabla 6: Caso de prueba caso de uso " Generar Ruta”. Variables.

Descripción de Variable				
No.	**Nombre de campo**	**Clasificación**	**Valor Nulo**	**Descripción**
1	Origen	Lista desplegable.	No	Se debe seleccionar un origen

2	Destino	Lista desplegable.	No	Se debe seleccionar un destino

Tabla 7: Caso de prueba caso de uso " Generar Ruta". Matriz de Datos.

Escenario	Variable 1 Origen	Variable 2 Destino	Respuesta del Sistema	Flujo Central
EC 1.1: Selección de la opción "Obtener Combinaciones de Rutas".	NA	NA	El sistema muestra un formulario con los campos a seleccionar por el usuario: • Origen. • Destino.	• Seleccionar en el menú la opción "Combinaciones de Rutas"
EC 1.2: Desplegar Origen.	NA	NA	El sistema despliega una lista con los puntos de origen.	• Seleccionar en el menú la opción "Combinaciones de Rutas". • Desplegar el *combobox* "Origen".
EC 1.3: Desplegar Destino.	NA	NA	El sistema despliega una lista con los puntos de destino.	• Seleccionar en el menú la opción "Combinaciones de Rutas". • Desplegar el *combobox* "Destino".
EC 1.4: Contraer Origen.	NA	NA	El sistema contrae la lista de los puntos de origen.	• Seleccionar en el menú la opción "Combinaciones de Rutas". • Desplegar el *combobox* "Origen".
EC 1.5: Contraer Destino.	NA	NA	El sistema contrae la lista de los puntos de destino.	• Seleccionar en el menú la opción "Combinaciones de Rutas". • Desplegar el *combobox* "Destino".
EC 1.6: Campos	()	()	El sistema muestra	• Seleccionar en el menú la

Vacíos.			un mensaje informando que existen campos vacíos: “Seleccione un punto de origen”.	opción “Combinaciones de Rutas”. • Seleccionar la opción “Aceptar”.
	“UCI- a Centro Habana ”	()	El sistema muestra un informando que existen campos vacíos: “Seleccione un punto de destino”.	
EC 1.7: Mostrar Datos de Ruta.	“UCI- a Centro Habana ”	“Cine Riviera”	El sistema muestra los datos de cada tramo que representa la ruta entre los dos puntos.	• Seleccionar en el menú la opción “Combinaciones de Rutas”. • Seleccionar campo en el *combobox* “Destino”. • Seleccionar campo en el *combobox* “Origen”. • Seleccionar la opción “Aceptar”.
EC 1.7: Cancelación de la funcionalidad.	NA	NA	El sistema oculta la ventana “Combinaciones de Rutas”.	• Seleccionar en el menú la opción “Combinaciones de Rutas”. • Seleccionar la opción “Cancelar”.

Resultados

Se realizaron dos iteraciones del caso de prueba encontrándose en la primera cuatro no conformidades las cuales fueron solucionadas. En la segunda iteración no se encontraron no conformidades. Los resultados de las pruebas realizadas se consideran satisfactorios para el equipo de desarrollo.

4.3.2. Casos de pruebas estructurales

La prueba Caja Blanca o estructural es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Este tipo de prueba se aplica fundamentalmente sobre la estructura del código para probar que todos los caminos independientes de cada módulo se ejecuten al menos una vez, todas las decisiones lógicas sean realizadas en sus vertientes verdadera y falsa, se ejecuten todos los bucles en sus límites y con sus límites operacionales; y se

ejerciten las estructuras internas de datos para asegurar su validez. La *técnica del camino básico* es una prueba de caja blanca que permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida corno guía para la definición de un conjunto básico de caminos de ejecución. Esta técnica define como primer paso el cálculo de la complejidad ciclomática del grafo que representa el código en cuestión, brindando esta la cantidad de posibles caminos a recorren en el grafo. Después de definir cuáles son estos caminos se recorre cada sentencia del programa, a partir de valores definidos, para verificar que todos los caminos sean recorridos al menos una vez en distintas situaciones (Pressman, 2005). Teniendo en cuenta las características antes descritas se selecciona el método de diseño de casos de pruebas estructural haciendo uso de la *técnica del camino básico* para validar el funcionamiento de los métodos más importantes implementados en el módulo.

Pruebas Caja Blanca aplicadas al sistema

Clase DijkstraGrafo. Función calcularCamino (Ver Anexo 4, Figura 15):

Paso #1: Grafo de flujos de datos (Figura 11).

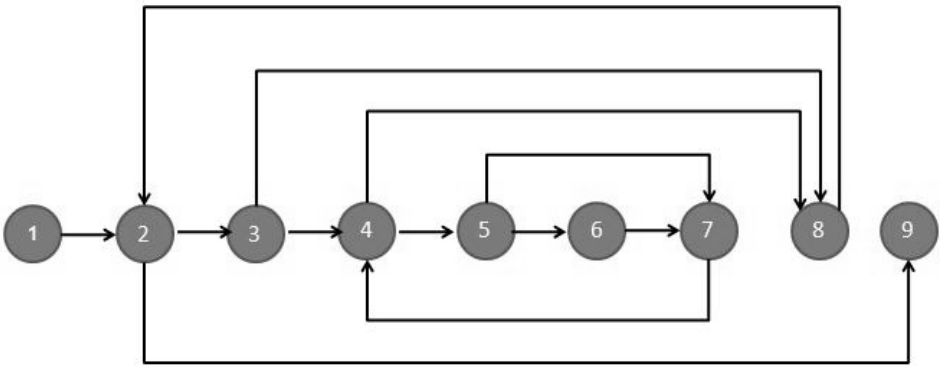

Figura 11: Grafo de flujo de datos. Clase DijkstraGrafo. Función calcularCamino.

Paso #2: Complejidad ciclomática.

V (G) =Cantidad Aristas - Cantidad Nodos + 2

V (G) = 12 – 9 + 2

V (G) = 7

Paso #3: Caminos básicos.

CB1: 1, 2, 9

CB2: 1, 2, 3, 8, 2, 9

CB3: 1, 2, 3, 4, 8, 2, 9

CB4: 1, 2, 3, 4, 5, 7, 4, 8, 2, 9,

CB5: 1, 2, 3, 4, 5, 6, 7, 4, 8, 2, 9

Paso #4: Casos de prueba.

Tabla 8: Método calcularCamino. Caso de Prueba #1

Caso de Prueba #1	
Camino:	1, 2, 3, 4, 5, 6, 7, 4, 8, 2, 9
Caso de Prueba:	Probando la función calcularCamino**.**
Entrada:	Para un valor de: origen=”1401”, visitar.longitud=15, mataday.longitud= 15x15.
Resultado:	Calcular el camino mínimo desde origen hasta todos los nodos de la lista visitar.

Clase ServerCombinaciones. Función GenerarGrafo (Ver Anexo 4, Figura).

Paso #1: Grafo de flujos de datos (Figura 12).

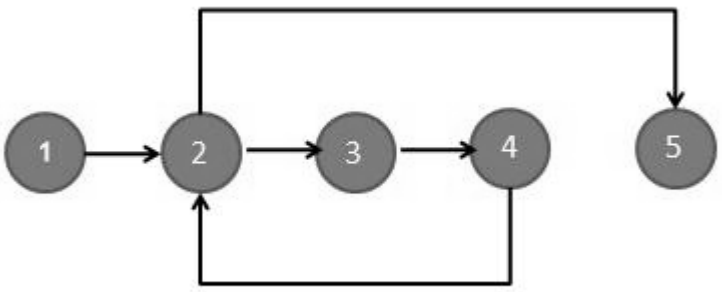

Figura 12: Grafo de flujo de datos. Clase ServerCombinaciones. Función GenerarGrafo.

Paso #2: Complejidad ciclomática.

V (G) =Cantidad Aristas - Cantidad Nodos + 2

V (G) = 5 – 5 + 2

V (G) = 2

Paso #3: Caminos básicos.

CB1: 1, 2, 5

CB2: 1, 2, 3, 4, 2, 5

Paso #4: Casos de prueba.

Tabla 9: Método GenerarGrafo. Caso de Prueba #1

Caso de Prueba #1	
Camino:	1, 2, 3, 4, 2, 5
Caso de Prueba:	Probando la función GenerarGrafo**.**
Entrada:	Para un valor de: $ret != null.
Resultado:	Generar un grafo a partir de los datos obtenidos en la variable $ret.

Tabla 10: Resultado de los casos de pruebas.

No. del camino	**Caso de Prueba**	**Objetivo**	**Resultado**
5	Probando la función calcularCamino**.**	Calcular el camino mínimo desde origen hasta todos los nodos de la lista visitar.	Satisfactorio
2	Probando la función GenerarGrafo**.**	Generar un grafo a partir de los datos obtenidos en la variable $ret.	Satisfactorio

4.4. Conclusiones del capítulo

Los artefactos generados durante la implementación representan la estructura y relaciones necesarias para un correcto funcionamiento del sistema. Con la descripción del modelo de implementación del módulo se ha podido ilustrar de manera efectiva el funcionamiento interno de la aplicación y como fueron implementados los requisitos funcionales en términos de componentes. Las pruebas aplicadas al sistema demostraron el correcto funcionamiento del mismo por lo que se puede concluir que el sistema es completamente funcional.

CONCLUSIONES GENERALES

Durante el desarrollo de la presente investigación se ha arribado a las siguientes conclusiones:

Con la utilización e integración del sistema desarrollado al Sistema de Información Geográfica para el transporte obrero de la Universidad de las Ciencias Informáticas se incrementan las facilidades que brinda el mismo a los usuarios. Este módulo reduce de manera considerable el tiempo que tendría que emplear un usuario para definir una ruta óptima por la cual trasladarse desde la universidad hasta un punto de la ciudad de La Habana. El módulo desarrollado constituye además una herramienta de gran utilidad y demanda por parte de los usuarios ya que facilita la gestión de información de cada tramo que conforma la ruta óptima para el traslado entre dos puntos, además de que su representación visual en el mapa contribuye en gran medida a una mejor interpretación de la información brindada.

Por lo anterior planteado se puede concluir que con el Componente para la generación y representación de combinaciones de rutas para el Sistema de Información Geográfica del transporte obrero de la UCI se logra incrementar las facilidades brindadas por el sistema que tiene como objetivo facilitar la toma de decisiones de los usuarios en cuanto a transportación se refiere. Además, se cumplieron todos los objetivos planteados al inicio de la investigación lográndose el desarrollo de un componente completamente funcional. Los artefactos generados durante todo el proceso de desarrollo servirán para que otros desarrolladores tengan una visión general del sistema permitiéndoles hacer cambios o agregar nuevas funcionalidades al mismo. De manera general las herramientas utilizadas para el desarrollo del módulo son libres lo que reduce el gasto en cuanto al pago de licencias de software.

RECOMENDACIONES

Luego de haber concluido el sistema y cumplido los objetivos planteados se realizan las siguientes recomendaciones:

- Valorar la inclusión de otros datos que surjan, de acuerdo a las necesidades de los usuarios, dentro de la información de la ruta generada.
- Incrementar los niveles de información que se tienen de las rutas incluyendo nuevos datos como el tiempo de recorrido y el flujo de tráfico en cada tramo, los cuales servirían como nuevos criterios de optimización de ruta.
- Realizar un estudio acerca de las posibilidades de integrar el componente desarrollado en otros sistemas para la gestión de redes de transporte.

GLOSARIO DE TÉRMINOS

ArchiMate: (*Enterprise Architecture Framework*): Framework de Arquitectura Empresarial.

BPMN: (*Business Process Modeling Notation*) Notación de Modelado de Procesos de Negocio.

CASE (*Computer Aided Software Engineering*) Ingeniería de Software Asistida por Computadora.

ERD: (*Entity Relationship Diagram***)** Diagrama entidad relación.

GDAL (*Geospatial Data Abstraction Library*) Librería para formatos de datos geoespaciales ráster.

GOF: (*Gang of Four*) "Banda de cuatros", hace referencia a los cuatro autores del libro *Design Patterns* (Patrones de Diseño) donde se recopilaron por primera vez estos patrones.

GRASP: (*General Responsibility Asignment Software Patterns*) Patrones generales de software para asignar responsabilidades.

HTML: (*HyperText Markup Language)* Lenguaje para escribir páginas *web*.

OGC: (*Open Geospatial Consortium)* Es un consorcio industrial internacional de 482 empresas, agencias gubernamentales y universidades que participan en un proceso de consenso para desarrollar estándares de interfaz.

OGR: Es una librería de código abierto para el procesamiento de datos vectoriales.

PHP: (*Hypertext PreProcessor).*

SysML: (*Systems Modeling Lenguage*) Es un lenguaje de modelado para aplicaciones de ingeniería de sistemas.

BIBLIOGRAFÍA REFERENCIADA

Achour, Mehedi, y otros. 2005. PHP Manual. 2005.

Balduino, Ricardo. 2007. Eclipse. [En línea] Agosto de 2007. [Citado el: 12 de Diciembre de 2012.] http://www.eclipse.org/.

Booch, Grady, Rumbaugh, James y Jacobson, Ivar. 2006. El Lenguaje Unificado de Modelado. Madrid : Pearson Addison-Wesley, 2006. ISBN: 8478290761.

—. 2000. Proceso Unificado de Desarrollo de Software. Madrid : Editorial Addison Wesley , 2000. ISBN: 74-7829-036-2.

Bosque, Sendra J. 1992. Sistemas de Información Gegráfica. 2 Madrid : Ediciones Rialp, SA, 1992. ISBN: 84-321-3154-7.

Calvo, Melero M. 1993. Sistemas de Información Geográfica Digitales. Vitoria : Instituto Vasco de Administración Pública, 1993. ISBN: 84-7777-101-4.

Cardozo, Osvaldo, Gómez, Erica y Parras, Miguel. 2009. Teoría de Grafos y Sistemas de Información Geográfica aplicados al Transporte público de Pasajeros en Resistencia. Buenos Aires : Revista Transporte y Territorio, 2009. 1, págs. 89-111. ISSN: 1852-7175.

Carrillo Pérez, Isaías, Pérez González, Rodrigo y Rodríguez Martín, Aureliano David. 2008. Metodología de Desarrollo del Software. 2008.

Carrillo Romero, Germán Alonso. 2008. GeoTux. [En línea] Abril de 2008. [Citado el: 5 de Diciembre de 2012.] http://geotux.tuxfamily.org/.

Chorley, R. 1987. Handling Geographic Information. Report of the Committee of Enquiry. Londres : Department of the Environment HMSO, 1987. ISSN 0015-752X.

de la Torre LLorente, César, y otros. 2010. Guía de Arquitectura N-Capas Orientada al Domino. Madrid : Krasis Press, 2010. ISBN: 978-84-936696-3-8.

Development Group, The PostgreSQL Global. 2005. PostgreSQL 8.1.0 Documentation. 2005.

Diestel, Reinhard. 2000. *Graph Theory.* New York : Electronic Edition, 2000. ISBN 0-387-98976-5.

Doovive. 2011. Doovive. [En línea] 2 de abril de 2011. [Citado el: 10 de diciembre de 2012.] http://doovive.com/herramientas-web-movilidad-ciudad/.

Dorigo, Marco y Stützle, Thomas. 2004. *Ant Colony Optimization.* Massachusetts, USA : Massachusetts Institute of Technology, 2004. ISBN 0-262-04219-3.

Eguíluz Pérez, Javier. 2008. *Introducción a AJAX.* 2008.

Esri. 2010. ArcGIS Resources. [En línea] 20 de Diciembre de 2010. [Citado el: 12 de Enero de 2013.] http://webhelp.esri.com/arcgisserver/9.3/java/geodatabases/topology_basics.htm.

Frederick, Shea, Ramsay, Colin y Blade, Steve. 2008. Learning ExtJS. Birmingham : Packt Publishing Ltd., 2008. ISBN 978-1-847195-14-2.

Gamma, Erich, y otros. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. s.l. : Addison Wesley., 1994. ISBN 0-201-63361-2.

Garcia, Jesus. 2009. ExtJS in Action. s.l. : Manning Publications, 2009. ISBN: 9781935182115.

Garzón Pérez, Ma. Terersa. 2010. Sistemas Gestores de Bases de Datos. Granada : s.n., 2010. ISSN 1988-6047.

Google. 2013. *Google Transit.* [Imagen] 2013.

Gracia Aguila, Adrian. 2011. Plataforma tecnológica para el almacenamiento, representación y análisis de información espacial. Holguin. : s.n., 2011.

Grupo de Autores. 2010. ISO TC 211 Standars Guide Spanish. Mexico : Instituto Panamericano de Geografía e Historia, 2010. 541. ISBN: 978-607-7842-03-3.

Gutmans, Andi, Bakken, Stig Sæther y Rethans, Derick. 2004. PHP 5 Power Programming. Indianapolis : PRENTICE HALL, 2004. ISBN 0-131-47149-X.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana : Felix Varela, 2004. Vol. 1. 0-201-57169-2.

Larman, Craig. 1999. UML Y PATRONES: Introducción al análisis y diseño orientado a objetos. Mexico : PARENTICE HALL, 1999. ISBN: 970-17-0261-1.

MapServer. MapServer. *Open Source Web Mapping.* [En línea] Universidad de Minesota. [Citado el: 13 de Diciembre de 2012.] http://mapserver.org.

Martinez, Evelio. 2002. Red de Transporte. Mexico DF : s.n., Diciembre de 2002. Edición Especial.

Molinero, Angel y Sánchez Arellano, Luis Ignacio. 2005. *Transporte Público: Planeación, Diseño, Operación y Administración.* Toluca, México DF : Universidad Autónoma del Estado de México, 2005. ISBN 968-835-353-1.

Mora, Francisco. 2003. Uml. Alicante : Universidad de Alicante, 2003.

Oracle Corporation. NetBeans Ide. [En línea] [Citado el: 13 de Diciembre de 2012.] http://netbeans.org.

Powe, Thomaas. 2001. Diseño de Sitios web. Madrid : McGraw-Hill Editorial , 2001. ISBN: 8448129059.

Pressman, Roger S. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educación S.A., 2000. 84-7829-036-2.

—. 2005. Ingeniería de Software. Un enfoque práctico. s.l. : Mc Graw Hill, 2005. ISBN: 9701054733.

Rational Software. 2012. Rational Information Center. [En línea] Diciembre de 2012. [Citado el: 18 de Diciembre de 2012.] publib.boulder.ibm.com.

Rodríguez Villalobos, Alejandro. 2006. Rutas, Software para el cálculo de rutas de vehículos y gestión de flotas. [En línea] 2006. [Citado el: 10 de Diciembre de 2012.] http://personales.upv.es/arodrigu/rutas/.

Skiena, Steven S. 2008. The Algorithm Desing Manual. 2 London : Springer-Verlag, 2008. ISBN: 978-1-84800-069-8.

Universidad de Jaén. 2006. Departamento de Informatica de la Universidad de Jaén. [En línea] Marzo de 2006. [Citado el: 10 de Enero de 2013.] http://wwwdi.ujaen.es.

Visual Paradigm. UML, BPMN and Enterprise Architecture Tool for Software Development. [En línea] [Citado el: 13 de Diciembre de 2012.] http://www.visual-paradigm.com/.

BIBLIOGRAFÍA

Achour, Mehedi, y otros. 2005. PHP Manual. 2005.

Alarcón, Raúl. 2000. Diseño Orientado a Objetos UML. Madrid : EIDOS, 2000.

Ambler, Scott M. 2006. *Agil Unified Process.* [Documento html] s.l. : Ambysoft Inc.'s Agile UP Product, 2006.

2011 ArcGIS Resource Centerhttp://resources.arcgis.com/es/home/

Balduino, Ricardo. 2007. Eclipse. [En línea] Agosto de 2007. [Citado el: 12 de Diciembre de 2012.] http://www.eclipse.org/.

Barahona, Sebastian. 2005. Aplicación de Geotecnologías Open Source para implementar Sistemas de Información Geográfica en Internet. Santiago de Chile. : Universidad Santiago de Chile., 2005.

Booch, Grady, Rumbaugh, James y Jacobson, Ivar. 2006. El Lenguaje Unificado de Modelado. Madrid : Pearson Addison-Wesley, 2006. ISBN: 8478290761.

—. 2000. Proceso Unificado de Desarrollo de Software. Madrid : Editorial Addison Wesley , 2000. ISBN: 74-7829-036-2.

Bosque, Sendra J. 1992. Sistemas de Información Gegráfica. 2 Madrid : Ediciones Rialp, SA, 1992. ISBN: 84-321-3154-7.

Calvo, Melero M. 1993. Sistemas de Información Geográfica Digitales. Vitoria : Instituto Vasco de Administración Pública, 1993. ISBN: 84-7777-101-4.

Cardozo, Osvaldo, Gómez, Erica y Parras, Miguel. 2009. Teoría de Grafos y Sistemas de Información Geográfica aplicados al Transporte público de Pasajeros en Resistencia. Buenos Aires : Revista Transporte y Territorio, 2009. 1, págs. 89-111. ISSN: 1852-7175.

Carrillo Pérez, Isaías, Pérez González, Rodrigo y Rodríguez Martín, Aureliano David. 2008. Metodología de Desarrollo del Software. 2008.

Carrillo Romero, Germán Alonso. 2008. GeoTux. [En línea] Abril de 2008. [Citado el: 5 de Diciembre de 2012.] http://geotux.tuxfamily.org/.

Chorley, R. 1987. Handling Geographic Information. Report of the Committee of Enquiry. Londres : Department of the Environment HMSO, 1987. ISSN 0015-752X.

Cortizo Pérez, José Carlos, Expósito Gil, Diego y Ruiz Leyva, Miguel. 2004. Extreme Programming 1. España : s.n., 2004.

Cruz, Elmidio Estévez. 2009. Apuntes sobre estimación de recursos y reservas. 2009.

de la Torre LLorente, César, y otros. 2010. Guía de Arquitectura N-Capas Orientada al Domino. Madrid : Krasis Press, 2010. ISBN: 978-84-936696-3-8.

Development Group, The PostgreSQL Global. 2005. PostgreSQL 8.1.0 Documentation. 2005.

Díaz Ruiz, Olga. 2012. UCI: la mirada de un decenio. *Granma.* Jueves 11 de Octubre, 2012, 281.

Diestel, Reinhard. 2000. *Graph Theory.* New York : Electronic Edition, 2000. ISBN 0-387-98976-5.

Doovive. 2011. Doovive. [En línea] 2 de abril de 2011. [Citado el: 10 de diciembre de 2012.] http://doovive.com/herramientas-web-movilidad-ciudad/.

Dorigo, Marco y Stützle, Thomas. 2004. Ant Colony Optimization. Massachusetts, USA : Massachusetts Institute of Technology, 2004. ISBN 0-262-04219-3.

Ecured. [En línea] http://www.ecured.cu.

Eguíluz Pérez, Javier. 2008. *Introducción a AJAX.* 2008.

Eriksson, Hans Erik y Penker, Magnus. 1998. UML Toolkit. New York : John Wiley & Sons, 1998.

Espinosa, José Manuel Becerra. Estadística Descriptiva. [En línea] [Citado el: 15 de Diciembre de 2012.] http://www.fca.unam.mx/docs/apuntes_matematicas/34.%20Estadistica%20Descriptiva.pdf.

Esri. 2010. ArcGIS Resources. [En línea] 20 de Diciembre de 2010. [Citado el: 12 de Enero de 2013.] http://webhelp.esri.com/arcgisserver/9.3/java/geodatabases/topology_basics.htm.

Frederick, Shea, Ramsay, Colin y Blade, Steve. 2008. Learning ExtJS. Birmingham : Packt Publishing Ltd., 2008. ISBN 978-1-847195-14-2.

Free Software Foundation. 2012. GNU Operating System. [En línea] 31 de Enero de 2012. [Citado el: 1 de Enero de 2013.] http://www.gnu.org/software/gsl/.

—. 2013. Sistema Operativo GNU. [En línea] 2013. [Citado el: 8 de Enero de 2013.] http://www.gnu.org/software/pspp/manual/html_node/Statistical-Functions.html#Statistical-Functions.

FUNDIBEQ. 2010. Fundación Iberoamericana para la Gestión de la Calidad. [En línea] 2010. [Citado el: 4 de 12 de 2012.] http://www.fundibeq.org/opencms/export/sites/default/PWF/downloads/gallery/methodology/tools/histograma.pdf.

Gamma, Erich, y otros. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. s.l. : Addison Wesley., 1994. ISBN 0-201-63361-2.

Garcia, Jesus. 2009. ExtJS in Action. s.l. : Manning Publications, 2009. ISBN: 9781935182115.

Garcilaso, Jordana. 2008. Introducción Open Up. [En línea] Octubre de 2008. [Citado el: 14 de Enero de 2013.] http://www.mug.org.ar/Descargas/Jornadas/Downloads_GetFile.aspx?id=3136.

Garzón Pérez, Ma. Terersa. 2010. Sistemas Gestores de Bases de Datos. Granada : s.n., 2010. ISSN 1988-6047.

GeoTux. 2009. GeoTux. *Soluciones Geoinformáticas Libres.* [En línea] 1 de Agosto de 2009. [Citado el: 07 de Diciembre de 2012.] http://geotux.tuxfamily.org/index.php/es/component/k2/item/213-trasmisig-calculo-de-rutas-con-pyqgis-y-postgresql/postgis.

Google. 2013. *Google Transit.* [Imagen] 2013.

Gracia Aguila, Adrian. 2011. Plataforma tecnológica para el almacenamiento, representación y análisis de información espacial. Holguin. : s.n., 2011.

Grupo de Autores. 2010. ISO TC 211 Standars Guide Spanish. Mexico : Instituto Panamericano de Geografía e Historia, 2010. 541. ISBN: 978-607-7842-03-3.

Guerrero, Rafael Martínez. 2010. PostgreSql-es. [En línea] 2010. [Citado el: 12 de Enero de 2013.] http://www.postgresql.org.es/sobre_postgresql#caracteristicas.

Gutmans, Andi, Bakken, Stig Sæther y Rethans, Derick. 2004. PHP 5 Power Programming. Indianapolis : PRENTICE HALL, 2004. ISBN 0-131-47149-X.

Iglesias, Elizabeth Martínez. 2007. Adaptación de la metodología RUP a los nuevos módulos del Proyecto Registros y Notarías. La Habana : Universidad de las Ciencias Informáticas, 2007.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana : Felix Varela, 2004. Vol. 1. 0-201-57169-2.

Larman, Craig. 1999. UML Y PATRONES: Introducción al análisis y diseño orientado a objetos. Mexico : PARENTICE HALL, 1999. ISBN: 970-17-0261-1.

MapServer. MapServer. *Open Source Web Mapping.* [En línea] Universidad de Minesota. [Citado el: 13 de Diciembre de 2012.] http://mapserver.org.

Martinez, Evelio. 2002. Red de Transporte. Mexico DF : s.n., Diciembre de 2002. Edición Especial.

Minitab Inc. 2013. Software for Quality Improvement. [En línea] 2013. [Citado el: 10 de Enero de 2013.] http://www.minitab.com/es-MX/products/minitab/.

Molinero, Angel y Sánchez Arellano, Luis Ignacio. 2005. Transporte Público: Planeación, Diseño, Operación y Administración. Toluca, México DF : Universidad Autónoma del Estado de México, 2005. ISBN 968-835-353-1.

Mora, Francisco. 2003. Uml. Alicante : Universidad de Alicante, 2003.

Oracle Corporation. NetBeans Ide. [En línea] [Citado el: 13 de Diciembre de 2012.] http://netbeans.org.

Patón, Eduardo Frenández-Medina. 2006. Alarcos. [En línea] 2006. [Citado el: 1 de Febrero de 2013.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema04.pdf.

Pérez, María Teresa Garzón. 2010. Sistemas Gestores de Bases de Datos . Granada : s.n., 2010. 1988-6047.

Powe, Thomaas. 2001. Diseño de Sitios web. Madrid : McGraw-Hill Editorial , 2001. ISBN: 8448129059.

Pressman, Roger S. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educación S.A., 2000. 84-7829-036-2.

—. 2005. Ingeniería de Software. Un enfoque práctico. s.l. : Mc Graw Hill, 2005. ISBN: 9701054733.

Rational Software Corp. 1997. The Unified Modeling Language for Object-Oriented Developmen. 1997.

Rational Software. 2012. Rational Information Center. [En línea] Diciembre de 2012. [Citado el: 18 de Diciembre de 2012.] publib.boulder.ibm.com.

Rodríguez Villalobos, Alejandro. 2006. Rutas, Software para el cálculo de rutas de vehículos y gestión de flotas. [En línea] 2006. [Citado el: 10 de Diciembre de 2012.] http://personales.upv.es/arodrigu/rutas/.

Sanchez, María A. Mendoza. 2004. Metodologías de Desarrollo de Software. Perú : s.n., 2004.

Skiena, Steven S. 2008. The Algorithm Desing Manual. 2 London : Springer-Verlag, 2008. ISBN: 978-1-84800-069-8.

Sommerville, Ian. 2005. Ingeniería del Software. Madrid : Pearson Educación S.A., 2005.

The PostgreSQL Global Development Group. 2010. PostgreSQL 9.1.0 Documentation. California : University of California, 2010.

UCI. 2012. Universidad de Las Ciencias Informáticas. [En línea] 2012. [Citado el: 15 de enero de 2013.] www.uci.cu.

Universidad de Jaén. 2006. Departamento de Informatica de la Universidad de Jaén. [En línea] Marzo de 2006. [Citado el: 10 de Enero de 2013.] http://wwwdi.ujaen.es.

Villar, Malay Rodríguez. 2007. Introducción de procedimientos ágiles en la produccón de software en la Facultad 7 de la Universidad de las Ciencias Informáticas. La Habana : Universidad de las Ciencias Informáticas, 2007.

Visconti, Marcello y Astudillo, Hernán. 2004. Departamento de Informática. [En línea] Universidad Técnica Federico Santa María, 22 de Octubre de 2004. [Citado el: 21 de 3 de 2013.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf.

Visual Paradigm. UML, BPMN and Enterprise Architecture Tool for Software Development. [En línea] [Citado el: 13 de Diciembre de 2012.] http://www.visual-paradigm.com/.

ANEXOS

Anexo 1: Descripción textual de los casos de uso del sistema

Tabla 11: Caso de Uso "Visualizar Ruta".

Caso de Uso	Visualizar Ruta	
Actores		
Resumen	El caso de uso inicia cuando el sistema recibe datos de una ruta específica.	
Precondiciones	Debe haberse realizado el caso de uso “Calcular Ruta”.	
Referencias	RF 8	
Prioridad	No critica.	
Flujo normal de eventos		
Acción del Actor		**Respuesta del Sistema**
		Obtiene los datos espaciales de la ruta calculada. Se muestra la representación espacial de la ruta en un mapa.

Tabla 12: Caso de uso "Generar Ruta". Sección “Calcular Ruta”.

Sección “Calcular Ruta”.	
Acción del Actor	**Respuesta del Sistema**
	El sistema recibe como parámetros de entrada un grafo y dos vértices del mismo. El sistema aplica el algoritmo de Dijkstra para la obtención de camino mínimo entre los dos vértices dados sobre el grafo. El sistema envía la información del camino mínimo generado.

Tabla 13: Caso de uso "Generar Ruta". Sección "Generar Grafo".

Sección "Generar Grafo".	
Acción del Actor	**Respuesta del Sistema**
	El sistema recibe como parámetro de entrada una lista de arista. El sistema genera, a partir de dicha lista, una lista de vértices y una matriz de adyacencia que representa las uniones entre dichos vértices. El sistema devuelve la lista de vértices y la matriz de adyacencia.

Anexo 2: Diagramas de clases del diseño

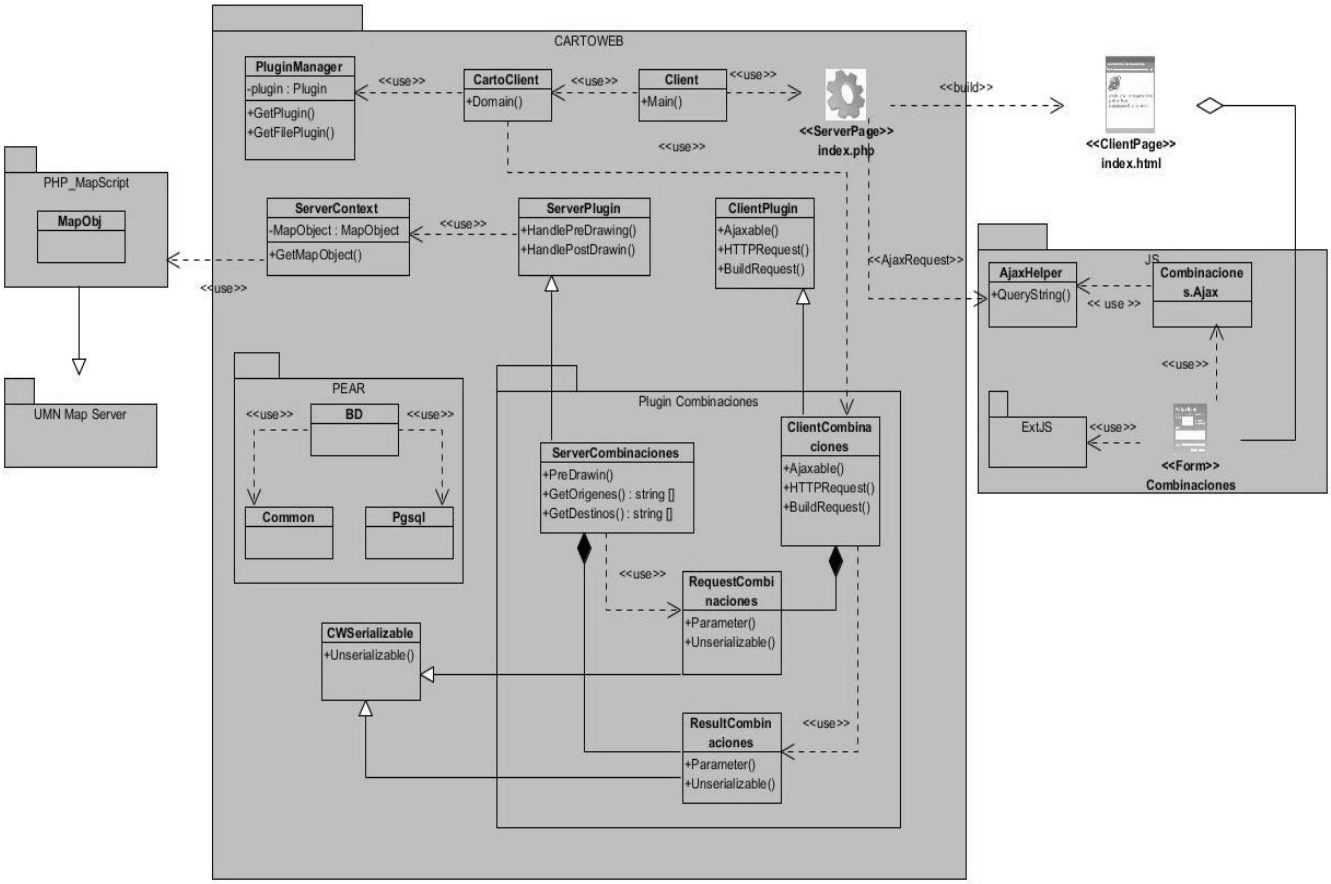

Figura 13: Diagrama de clases del diseño CU "Visualizar Rutas".

Anexo 3: Descripción de las clases del diseño.

Teniendo en cuenta que los diagramas de Clases del Diseño poseen clases que son comunes se realiza una descripción general de cada clase.

Clase index.html: Es la encargada de construir el formulario para la interacción del usuario con los datos.

Clase index.php: Tiene como propósito controlar la realización del CU en sí, recibe las peticiones realizadas por el cliente, gestiona las mismas y construye la ClientPage.

Clase Client: Contiene todos los archivos específicos de PHP del lado de CartoClient y permite la interacción entre la index.php y la CartoClient.

Clase CartoClient: Integra y recoge todos los datos y funciones realizadas por cada una de las .js que intervienen en el Caso de Uso, y es donde se define un conjunto de variables globales que van a ser utilizadas por la aplicación.

Clase PluginManager: Clase que se utiliza para gestionar la base de plugins.

Clase ClientPlugin: Contiene las interfaces necesarias para los plugins del lado del cliente.

Clase ServerPlugin: Esta clase proporciona la base de herramientas para el desarrollo de *plugins*.

Clase ServerContex: Es la contenedora de la información común que ha de ser utilizada por la parte cliente y la servidora, empleando la información seleccionada como un objeto para un fácil manejo de los datos.

Clase MapObj: Es donde se definen los métodos, funciones, además del lenguaje para el intercambio de datos con el servidor de mapa (MapServer).

Clase BD: Es la clase encargada de establecer la conexión con el servidor de base de datos para procesar los objetos a editar.

Clase Common: Es la encargada de administrar las conexiones a la BD para ejecutar las consultas a la misma satisfactoriamente, esto incluye tratamiento de los datos.

Clase Pgsql: Gestiona desde PHP las funciones de PostgreSql.

Clase CwSerializable: Se encarga de serializar todas aquellas clases que pueden ser serializadas, con el objetivo de transferir objetos a través de SOAP, permitiendo la comunicación entre el Client y el Server del *plugin*.

Clase AJAXHelper: Tiene como propósito enviar las respuestas de los plugins “AJAX”, para alimentar a los *plugins* que responden a las peticiones del usuario.

Anexo 4: Diagrama de componentes caso de uso "Visualizar Ruta"

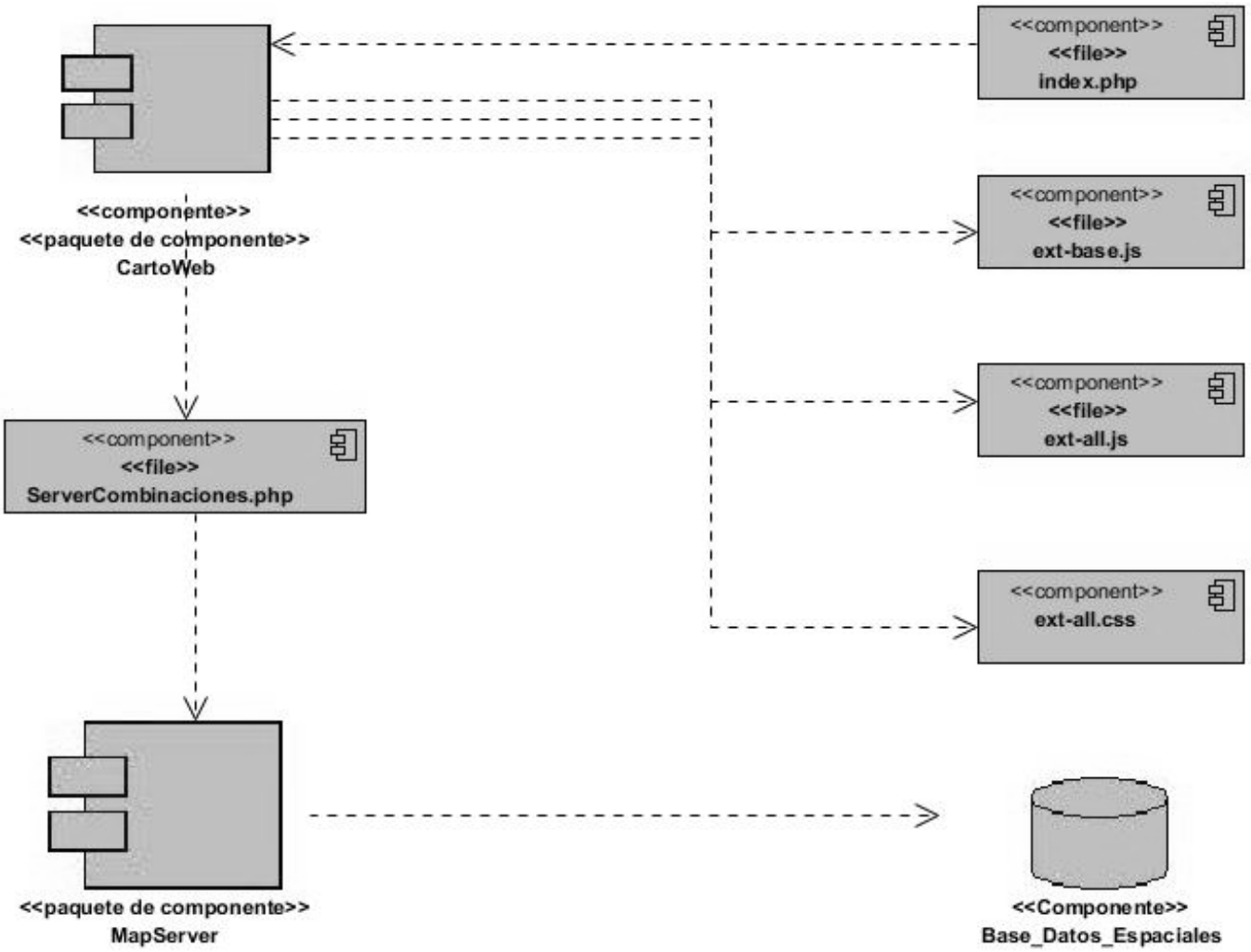

Figura 14: Diagrama de Componentes caso de uso "Visualizar Ruta".

Anexo 5: Códigos de las funciones asociadas a las pruebas de caja blanca

```
public function calcularCamino($origen) {
    $ret = array(array());                                                    1
    $this->visitar = $this->vertices;
    $slv_origen = $origen;
    $this->ini($slv_origen);
    while (count($this->visitar)) {                                           2
        $nodo = $this->buscarMin();
        if ($this->distMinOrigen[$nodo] != DijkstraGrafo::DISTANCIA_IMPOSIBLE) {   3
            foreach ($this->visitar as $visit) {                              4
                if ($this->distMinOrigen[$visit] > $this->distMinOrigen[$nodo] +
                        $this->mataday[$nodo][$visit]) {                      5
                    $this->distMinOrigen[$visit] = $this->distMinOrigen[$nodo] +
                            $this->mataday[$nodo][$visit];                    6
                    $this->preds[$visit] = $nodo;
                }
            }                                                                 7
        }
        $this->visitados[] = $nodo;
    }                                                                         8
    $ret[$slv_origen]['dist'] = $this->distMinOrigen;                         9
    $ret[$slv_origen]['pred'] = $this->preds;
    return $ret;
}
```

Figura 15: Función calcularCamino de la clase DijkstraGrafo.

```
private function GenerarGrafo()
{
    $db = $this->getDB();                                                    1
    $res = $db->query("Select * from public.grafo order by id");
   //Crear grafo lista vert - mataday
   while ($res->fetchInto($dato)) {                                          2
       $this->aristas[] = array('nodo_ini' => $dato[0],                      3
           'nodo_fin' => $dato[1],
           'dir_ini' => $dato[3],
           'dir_fin' => $dato[4],
           'id' => $dato[5],
           'dist' => $dato[6],
           'ruta' => $dato[7],
           'reverse' => $dato[8]);
       $this->dijkstra->setVertice($dato[0]);
       $this->dijkstra->setVertice($dato[1]);
       $this->dijkstra->setDistancia($dato[0], $dato[1], $dato[6], $dato[8]);
   }                                                                         4
}                                                                            5
```

Figura 16: Función generarGrafo de la clase ServerCombinaciones.