SOFTWARE PARA LA REPRESENTACIÓN GRÁFICA A LA SOLUCIÓN DE RUTEO Y CARGA DE VEHÍCULOS EN ALGORITMO DE TIPO 2L-CVRP

Hugo Urbano[1], Angel A. Juan[2]

RESUMEN

El siguiente artículo presenta el trabajo realizado en la creación de una aplicación de software libre que representa gráficamente las rutas generadas y la distribución de los elementos transportados al interior de un vehículo de carga, solución generada por un algoritmo 2L-CVRP, el cual propone un esquema de enrutamiento de vehículos y empaquetamiento de los elementos a ser distribuidos en su interior. El software fue construido siguiendo una metodología resultante de la combinación en las especificaciones de MSF y las metodologías ágiles, generando algunos diagramas UML que describen su comportamiento y documentan su funcionalidad.

PALABRAS CLAVE: Enrutamiento de vehículos, programación orientada a objetos (POO), software libre.

INTRODUCCIÓN

Los diferentes paradigmas de programación han desencadenado una variedad de escenarios de construcción de aplicaciones, pasando por una diversidad de enfoques funcionales que proporcionaban soluciones orientadas a las necesidades de los usuarios pero que transcurrido algún tiempo de uso y mantenimiento de la aplicación, ésta adquiere un nivel de complejidad considerable al tratar de analizar las estructuras y código fuente utilizados.

Partiendo de la necesidad de construir aplicaciones a partir de las especificaciones materiales del mundo real, surge un paradigma que pretende organizar las estructuras lógicas en forma similar a los objetos existentes en el mundo. Es así como la programación orientada a objetos proporciona un esquema de características y comportamientos a los elementos funcionales que integran una solución de software, combinando elementos de modelamiento unificado a través de UML y representado su dependencia a través de la sintaxis propia del lenguaje de programación (Kendal S, 2009).

La presente propuesta combina las características de la POO (Warmer, et al. 2003), generando como resultado un software de fuente abierta que representa gráficamente la solución al problema de enrutamiento de vehículos (Vehicle Routing Problem – en adelante identificado por las siglas VRP) que persigue como objetivo la entrega de bienes a un conjunto de clientes distribuidos geográficamente por un espacio conocido (Laporte, G. 1992), pero focalizando su solución en un algoritmo de tipología 2L-CVRP, el cual es una variante al VRP y elemento complementario a las restricciones de carga, problema que fue abordado y resuelto inicialmente por Iori (Iori et al. 2010). El problema resultante 2L-CVRP básicamente consiste en buscar un conjunto de rutas para la flota de vehículos, de tal forma que se minimice el coste de transporte, cumpliendo las restricciones de carga mencionadas anteriormente. La introducción de la restricción de carga en dos dimensiones (2L), al problema original CVRP, aumenta la complejidad del problema, reduciendo el espacio de soluciones posibles al tiempo que aumenta los costes de la solución (Gendreau, M. et al. 2008).

[1] Master en software libre – Universitat Oberta de Catalunya. Email: hurbano@uoc.edu
2 Department of Computer Science, Multimedia, and Telecommunication. IN3-Open University of Catalonia, 08018 Barcelona, Spain. email:ajuanp@uoc.edu

El funcionamiento del presente software, se basa completamente en la solución entregada por (Juan, Dominguez et al, 2013), en donde se propone un eficiente algoritmo, que reduce el número de parámetros, solucionando el problema de carga en dos dimensiones de vehículos de capacidad limitada y los problemas de enrutamiento, algoritmo que adopta la tipología 2L-CVRP. El tipo de problema mencionado, relaciona dos importantes tareas, como son los problemas de: ruteo y empaquetamiento. La propuesta contempla la carga sin restricción y la posibilidad de aplicar rotación de 90° para cada una de las figuras rectangulares en el momento de ser cargadas al interior del vehículo, además, tiene en cuenta la variedad de ejecución en un ambiente altamente parametrizable. Permite múltiples inicios, en donde cada uno utiliza un algoritmo de enrutamiento basado en ahorro, siendo una versión heurística de empaquetamiento clásico y produciendo soluciones factibles para la propuesta de tipo 2L-CVRP. Los resultados obtenidos mejoran las aproximaciones existentes al problema, entre éstos, el tiempo de cálculo en la visualización de resultados. La solución proporcionada por el algoritmo, evidencia una distribución pseudo-geométrica propuesta por otras investigaciones, de la misma manera, durante el procesamiento, se tiene en cuenta que el peso total de los ítems cargados al vehículo, sea inferior a la capacidad de carga, además, en el momento del empaquetamiento, los ítems se disponen descartando solapamiento.

Uno de los principales objetivos del presente del presente artículo, es describir las actividades realizadas en la construcción del software que representa gráficamente las rutas seguidas por los vehículos que integran la solución entregada por el algoritmo mencionado, así como la distribución de los elementos transportados al interior de la zona de carga. Por lo que a continuación se presenta la metodología utilizada, una revisión bibliográfica respecto al tema, la propuesta de software así como los problemas generados y las conclusiones del trabajo.

METODOLOGÍA

A pesar de considerarse como un proyecto pequeño – en comparación a soluciones empresariales – la propuesta fue construida considerando las etapas que la ingeniería del software y el modelo de procesos definen como son, entre otras: la obtención y análisis de requerimientos, el modelamiento de la solución, la definición de las pruebas funcionales, la codificación, la estabilización y ejecución de los casos de prueba (Pressman, R 2001, Microsoft. 2002). Las etapas mencionadas, fueron articuladas dentro de una metodología de desarrollo ágil, como es el caso de XP (Stephens, et al. 2003), en combinación con la estructura documental propuesta por MSF (Microsoft. 2002), hecho que favoreció estructurar funcionalmente los requerimientos del software, pues se sintetizó cada uno de los entregables del ciclo iterativo MSF, reduciendo las exigencias por etapa y generando como resultado: la definición de casos de uso, documentando los escenarios funcionales y casos de prueba, definiendo a través de un diagrama de clases la relación de los componentes lógicos que integrarían la solución, codificando la solución en lenguaje de programación Java y ejecutando las pruebas funcionales para certificar el cumplimiento de requisitos.

Como el software debía corresponder en su totalidad al resultado del algoritmo 2L-CVRP, el análisis, modelamiento y codificación de la solución tuvo su base sobre la estructura del archivo plano resultado del algoritmo de enrutamiento (Juan, Dominguez et al, 2013) de esta manera, y de acuerdo a la estructura del fichero de salida, en donde se representan las características de la flota de vehículos, los elementos a distribuir, los nodos con su ubicación y demanda y las rutas que satisfacen las exigencias de los nodos, el modelamiento del software tuvo como eje central

cada una de las rutas definidas por algoritmo, teniendo en cuenta que éstas se complementan en información por los vehículos, los nodos, y la característica de los elementos de carga, conformando de esta manera un completo esquema CVRP o capacited vehicle routing problem, que define la existencia de una flota o conjunto de vehículos repartidores, cuya capacidad de carga es uniforme (Juan et al. 2013) y de un solo producto, que pretenden atender la demanda de los nodos clientes, buscando la minimización de los costos asociados al proceso (Toth et al. 2002).

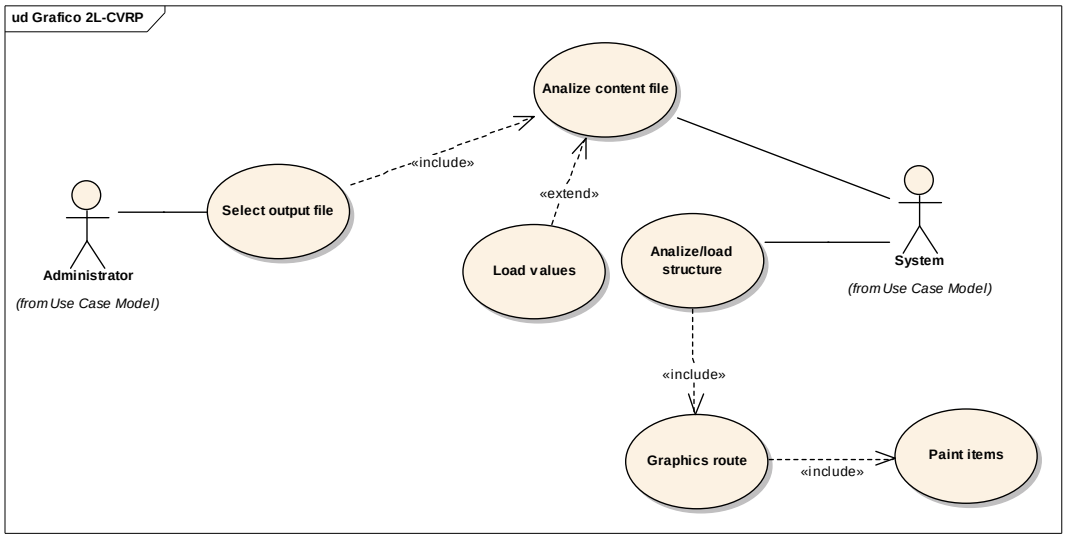

Figura 1. Diagrama de casos de uso

La figura 1 representa el diagrama de casos de uso generado como representación funcional del software construido, para el cual solamente son definidos dos actores: administrador y sistema. Para el caso del administrador, su función se limita a la selección del fichero de entrada y fuente de información en el proceso, dejando al actor sistema las tareas de analizar su contenido, cargar las estructuras lógicas necesarias, graficar las rutas y representar los ítems al interior del vehículo de carga.

ESTADO DEL ARTE

(Duhamel, C. 2011), representa gráficamente el resultado del estudio comparativo entre (Gendreau, M. et al. 2006) y (Fuellerer, G. et al. 2010), obteniendo como resultado una aplicación de software libre[3], que representa la solución gráfica al problema 3L-CVRP construido para plataforma Windows y distribuido bajo alguna de las licencias GNU.

(Da Silva, O. 2009) propone la creación de una aplicación de software, también distribuida como open source, que resuelve el problema de enrutamiento de vehículos con múltiples centro de distribución o nodos centrales (MDVRP - Multiple Depot vehicle routing problem). El proyecto inicialmente fue desarrollado complementando la funcionalidad de la aplicación OpenJUMP, actividad que fácilmente se realizaba, pues OpenJUMP se distribuye también bajo

[3] Sitio web de publicación del trabajo realizado en la representación gráfica de un algoritmo 3L-CVRP. http://www.isima.fr/~lacomme/3lcvrp/3lcvrp.html

las políticas del software libre. Posteriormente, se generan las librerías que al copiarse en el directorio raíz de la instalación de OpenJUMP, complementaba su funcionalidad (como elemento particular, la librería debe generarse en formato .jar). Jump utiliza la topología funcional JTS, la cual es un API de Java que implementa un número de operaciones de datos especiales utilizando modelos explícitos de precisión y robustos algoritmos geométricos, adicionalmente, esta topología es importante para la comunicación con otros sistemas, proporcionando portabilidad a los datos y la facilidad de importación de información de otros sistemas. El software permite, en combinación con elementos controlados en parte por OpenJUMP, restricciones de carga, volumen del depósito del vehículo y las posibles restricciones generadas con la red de carreteras.

Una aplicación comercial, distribuida por una comunidad Alemana denominada DNA Evolutions[4], pone a disposición del mercado una funcionalidad de ruteo elaborada en la plataforma .NET, que abarca también librerías para Java. JOpt.ASP, JOpt.SDK, JOpt.NET y JOpt.J2EE se conciben como complemento funcional que puede llegar a mejorar los productos de terceros, adhiriendo características innovadoras y conocimientos especiales en el problema de ruteo de vehículos, presentando una optimización que no es aplicable únicamente al VRP, sino al CVRP y VRPTW.

Xtreme Route[5] es otra aplicación comercial que proporciona al mercado una solución de planeación de rutas, isócronas y enrutamiento de flotas de vehículos. La librería está desarrollada en C# .NET. Uno de los objetivos perseguido con la librería, es la velocidad y rapidez en los cálculos, así como la versatilidad de uso en servicios web, aplicaciones de escritorio y su posibilidad de uso en ambientes enriquecidos visualmente como Silverlight.

PROPUESTA Y RESULTADOS OBTENIDOS

La aplicación de software construida corresponde a una solución open source, y se basa directamente en el algoritmo construido por (Juan, Dominguez et al, 2013), limitando su esquema funcional a la respuesta dada por los autores al problema de tipo 2L-CVRP por lo que si se desea conocer más, se recomienda verificar su publicación.

Como se ha comentado, la construcción del software fue estructurada en una combinación de MSF junto con los principios de las metodologías ágiles de construcción de aplicaciones, en la medida en que solamente se tuvieron en cuenta algunos aspectos que permitían definir su funcionamiento y modelar con UML su comportamiento. De esta manera, se definió que la solución debía abordarse en dos etapas, la primera de ellas se define como el análisis del fichero de entrada, fuente de información y elemento resultante del trabajo realizado por (Juan, Dominguez et al. 2013), en ésta etapa es necesario establecer la correcta estructura del archivo, para dar inicio al recorrido línea a línea estableciendo la existencia de las etiquetas identificadoras de la información relevante. Al establecerse la existencia de un elemento importante para la aplicación, es necesario convertirlo en un elemento lógico, almacenándolo al interior de una estructura organizada propia de la plataforma de desarrollo Java utilizada, como es el caso de las colecciones (Watt, D. et al. 2001). Al conservar éste elemento – ahora llamado objeto – dentro de una colección, fácilmente se puede localizar para complementarlo en información o simplemente para hacer uso de su contenido.

[4] DNA Evolutions. http://www.dna-evolutions.com
[5] Xtreme Route. http://xtremeroute.com/

Hasta este punto, se tiene cargada la memoria con los objetos generados a partir de las clases que integran el modelo y correspondientes a la información contenida en el fichero de entrada. Es importante resaltar, que solamente se hace uso del fichero *output* resultado del algoritmo mencionado, pues ésta salida se constituye en entrada para el software construido.

La siguiente etapa dentro del proceso, corresponde a utilizar los objetos generados y almacenados en los contenedores para generar la representación gráfica de la solución. A este punto se considera importante describir un poco la estructura de los objetos generados, pues ésta determina el funcionamiento del aplicativo. La aplicación construye un objeto vehículo, encargado de conservar la información característica de cada camión. El objeto nodo contiene la información y ubicación del cliente, así como su ubicación geográfica dentro del plano, esto es, el punto en el eje vertical y horizontal, además, cada nodo contiene a manera de colección, la demanda o ítems que deben distribuírsele, especificando para cada caso el ancho y alto del elemento. Finalmente, y no menos importantes, existen los objetos ruta, que combinan todos los anteriores objetos descritos, pues una ruta es seguida por un vehículo de carga, el cual contiene unos elementos que deben ser llevados a unos clientes para satisfacer su requerimiento o demanda. Es importante resaltar que todos los objetos son almacenados al interior de colecciones, las cuales garantizan que la información sea accesible y verificable, dejándola disponible en cualquier instante de tiempo sin la necesidad de recorrer nuevamente el archivo plano buscando cualquier dato.

Con toda la estructura del fichero e información importante para el proceso cargada en memoria, se procede a representar gráficamente el recorrido que cada uno de los vehículos realiza para satisfacer los requerimientos de cada nodo, considerándose ésta como la segunda etapa dentro del proceso. La representación gráfica inicia recorriendo la colección que contiene la información de los objetos ruta, estableciendo para cada una de ellas una convención a nivel de colores y tipo de línea y generando el elemento visual para cada una de ellas a partir de la ubicación del nodo anterior, o nodo origen según sea el caso. El proceso descrito con anterioridad, debe realizarse mientras existan rutas al interior de la colección que las contiene, este proceso iterativo fue elaborado con código reutilizable y gracias a las facilidades que la POO proporciona en este sentido, elaborando de esta manera un único elemento funcional traducido en procedimiento que invoca métodos complementarios para cada iteración, más adelante se realizará la descripción a nivel de diagrama de flujo de los procedimiento descritos.

Una vez se ha representado la solución gráfica para cada ruta (evidenciado en la figura 3), se rotulan los nodos con un número consecutivo generado y asignado por el algoritmo mencionado fuente de información. A continuación, se lleva a cabo la representación gráfica de la distribución de los elementos de carga al interior de cada vehículo (figura 4), siguiendo la misma metodología mencionada con anterioridad, en la cual se recorren las estructuras lógicas contenedoras de la información, extrayendo para cada vehículo los datos necesario que permiten esquematizar la solución y posteriormente rotulando cada elemento con su identificador y el del nodo al que pertenece.

La siguiente figura representa el diagrama de clases que satisface los requerimientos funcionales de la aplicación, diseño para el cual se tuvo en cuenta la creación de clases que permitían cargar lógicamente la información del fichero de entrada, generando una persistencia adaptada al contexto funcional que garantiza la disponibilidad de la información pues implementa estructuras dinámicas de fácil acceso y manipulación, como ya se ha comentado.

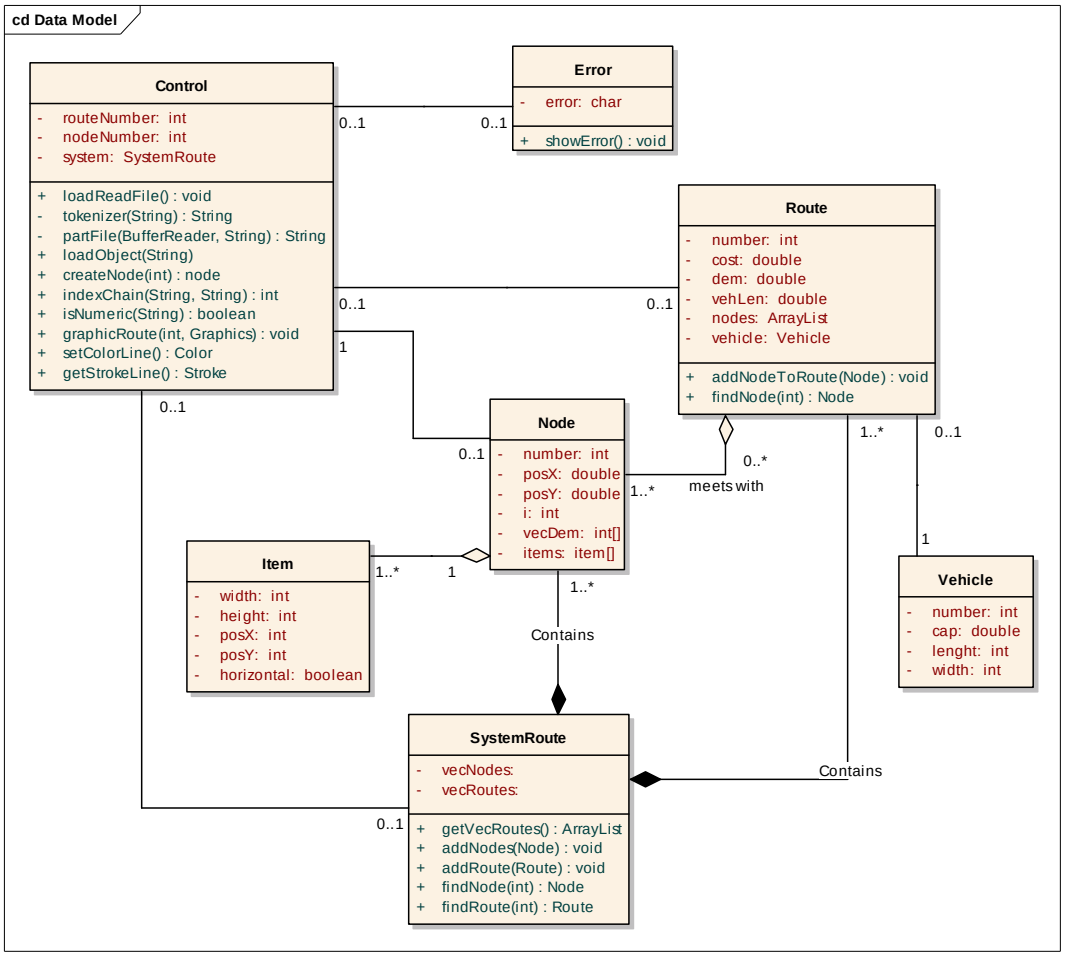

Figura 2. Diagrama de clases

Durante el proceso, surgieron algunos inconvenientes como son, en primer lugar, el fichero fuente de información no contenía etiquetas o elementos diferenciadores que permitieran establecer en qué momento se describía a las rutas, a los nodos o a los ítems, por tal motivo fue necesario adicionar tres elementos que serán denominados etiquetas y que permiten establecer la información de las rutas, los vehículos y los elementos de carga (ítems), para lo que se utilizó **&routes**, **&vehicle** y **&packing** respectivamente, antes de cada sección, y así tener una guía que permitiera identificar la ubicación de la información relevante. Diferenciado lo anterior de la demás información resumen resultado del algoritmo, se procedió al análisis por línea, extrayendo por posición lo que se consideraba útil para construir la gráfica.

Otro inconveniente presentado, se encuentra referido a la escala utilizada para la ubicación de los nodos en el plano y de los ítems al interior del vehículo de carga, pues la herramienta de programación utilizada, especifica como punto inicial el ubicado en la parte superior izquierda de la pantalla, hecho que dificultaba el proceso, máxime cuando se trataba de nodos cuya ubicación obedecía a valores negativos en el plano, casos para los cuales fue necesario dividir la zona de graficación en partes iguales a nivel vertical y horizontal, generando así un nuevo punto de origen, aumentando o disminuyendo en cada uno de sus ejes, los valores de la posición indicada por el nodo. Para el caso de los ítems, fue necesario tomar como punto de referencia el rectángulo que representa la zona de carga, graficando en su interior los ítems de acuerdo a la

posición proporcionada por el algoritmo, aumentando en cada uno de los ejes el valor indicado descartando una ubicación directa sobre el plano gráfico, pues este hecho no permitía evidenciar en forma organizada los ítems al interior del vehículo.

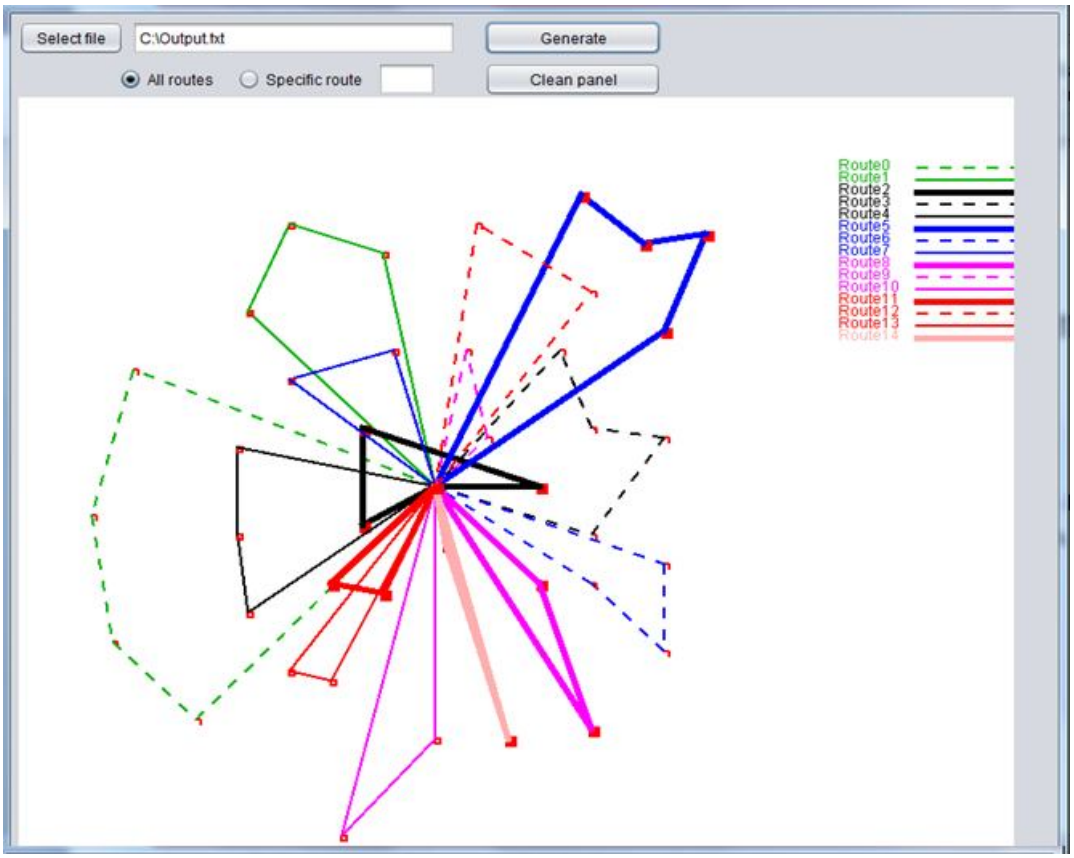

Figura 3. Representación gráfica de las rutas

Figura 4. Representación de los elementos de carga

CONCLUSIONES

En este documento fue descrito el proceso de construcción de la aplicación de software libre encargada de la representación gráfica al algoritmo de ruteo 2L-CVRP elaborado por (Juan, Dominguez et al. 2013), proceso al cual fueron adaptadas las actividades de la ingeniería del software, proponiendo una combinación metodológica que generó unos diagramas UML que representan y documentan el esquema funcional definido para cumplir con el requerimiento de representación visual del algoritmo.

La aplicación puede ser evolucionada y adaptada al proceso original encargado de solucionar el problema 2L-CVRP definido por (Juan, Dominguez et al. 2013), permitiendo no solo la generación del fichero con la solución al problema, sino la articulación con su representación gráfica.

BIBLIOGRAFÍA

Da Silva, O. 2009. Roteirização de veículos de carga com múltiplos depósitos em sistema de informação geográfica livre. *Dissertação (mestrado) Instituto Militar De Engenharia. Rio de Janeiro CEP: 22290-270.*

Duhamel, C., Lacomme, P., Quilliot, A., Toussaint, A., 2011. "A GRASPxELS approach for the VRP with three dimensional loading constraints". *Research Report LIMOS / RR-11-01.*

Fuellerer, G., Doener, K., Hartl, R., Iori, M. 2010. "Meta-Heuristics for Vehicle Routing with three dimensional loading constraints". *European Journal of Operational Research, 201(3):751-759.*

Gendreau, M., Iori, M., Laporte, G., Martello, S. 2008. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks 51:4–18.*

Gendreau, M., Iori, M., Martello, S. 2006. A tabu search algorithm for a routing and contrainer loading problem. *Transportation Science, 40(3):342-350.*

Iori, M. and Martello, S. 2010. Routing problems with loading constraints. *TOP, 18:4-27.*

Juan, A., Faulin, J., Agustin, A., Caceres, J. 2013. A Successive Approximations Method for solving the Heterogeneous Vehicle Routing Problem and analyzing different Fleet Configurations. *Journal of the Operational Research Society. ISSN.0160-5682.*

Juan, A., Dominguez, O., Faulin, J., Agustin, A. 2013. A Multi-Start Algorithm for Solving the Two-Dimensional Vehicle Routing Problem with Items Rotation. International *Transactions on Operational Research (indexed in ISI SCI). ISSN: 0969-6016.*

Kendal, S. 2009. Object Oriented Programming using Java. 209 p. Ventus publishing ApS. *ISBN 978-87-7681-501-1.*

Laporte, G. 1992 The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European journals of operation research. 59: 345-358.*

Microsoft. (2002, June). MSF Process Model v. 3.1. White paper overview. [artículo en línea]. [Fecha de consulta: 17 de enero de 2013].
<http://www.microsoft.com/en-us/download/details.aspx?id=24993>.

Pressman, R. 2001. Software Engineering: A practitioner's approach. *(5th edition). New York: McGraw-Hill. ISBN 0073655783.*

Stephens, M., Rosenberg, D. 2003. Extreme programing refactored: the case against XP. *New York: Springer-Verlag. ISBN 1590590961.*

Toth, P. and Vigo, D. 2002 Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics 123 (2002) 487 – 512. Università di Bologna.*

Warmer, J., Kleppe, A. 2003. The Object constraint language. (2th *edition). Boston: Pearson education. ISBN 0321179366.*

Watt, D. and Brown, D. 2001. Java Collections: An Introduction to Abstract Data Types, Data Structures and Algorithms. *New York: ACM. ISBN:047189978X.*

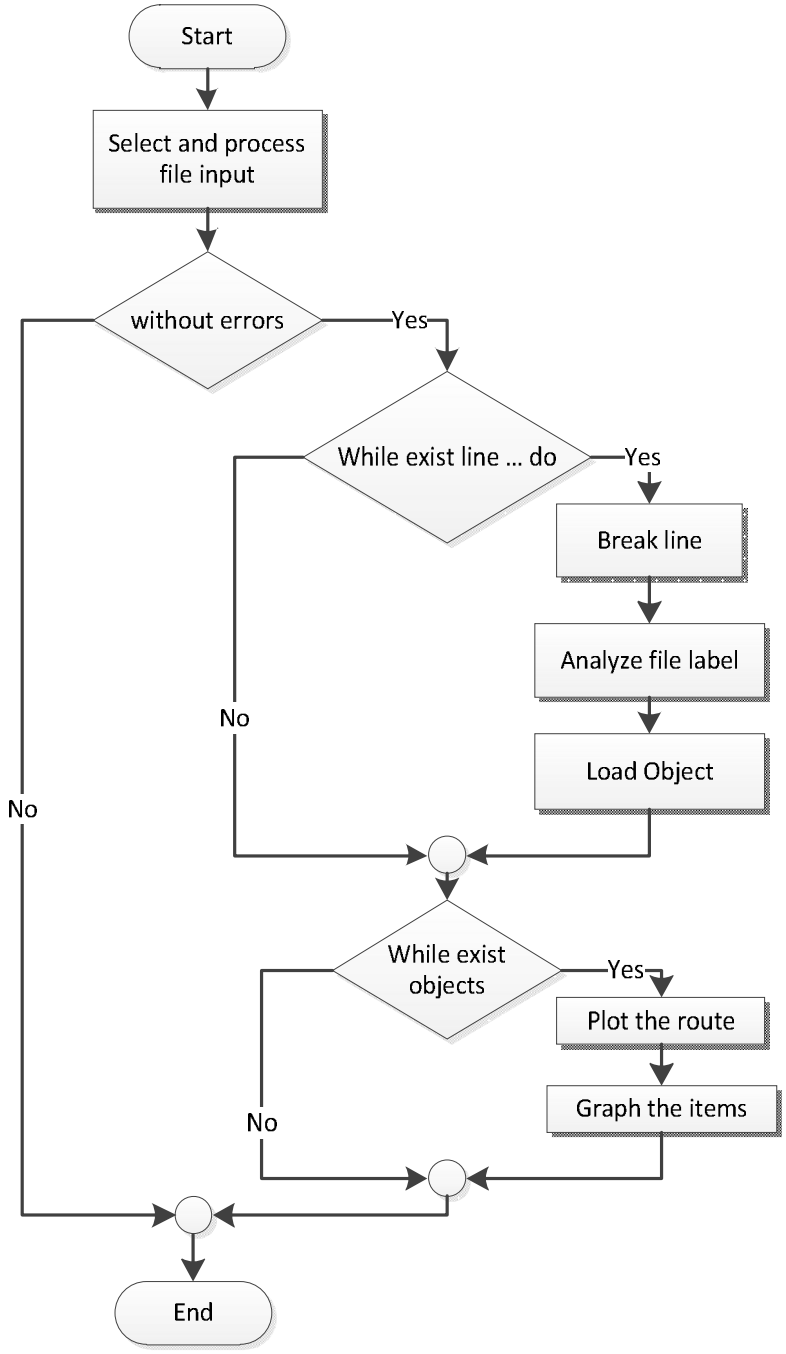

Diagrama de flujo general

```
public void graphicRoute(Graphics2D grp,int number,int scale,double avgX,double avgY) throws Error{
01      if(averageNodePosition()){
02         averageX = avgX;
03         averageY= avgY;
04      }
05      while(existElements()){
06          alNodesRoute = contentElement();

          //line style
07          stroke = getStrokeLine();

          //line String
08              drawString("Route"+route.getNumber() , 630, 50+line);

09          while(existNodes()){
            //graphic line
10              setColor(setColorLine());
11              drawLine(posX+averageX, posY+averageY,((node.getPosX())*scale)+averageX, ((node.getPosY())*scale)+averageY);
12              posX = (node.getPosX())*scale;
13              posY= (node.getPosY())*scale;
14              node.setPosX(posX+averageX);
15              node.setPosY(posY+averageY);

            //graphic point
16              fillOval(posX, posY, 5, 5);
17           }
18         }
19      }
```

Método encargado de graficar las rutas

```
private void fillRectWithLine(xi,  vi, xf, vf){
01        xf = xi+xf://Exchange the positions  value
02        yf = yi+yf;
03         //Draw perpendicular  line into rect
04           xil=yil=xi;
05           xfD=yfD=yi;
06           while(temp==0) {
07                  grp drawLine(xil yil,  xfD,yfD );
08                  if ((xfD>=xf)){// width  limit
09                     xfD=xf;
10                     yfD+=10;
11                  }
12                  else
13                  {
14                     xfD+=10;
15                  }
16                  if (yil>=yf){
17                     yil=yf;
18                     xil+=10;
19                  }
20                  else{
21                     yil+=10;
22                  }

23                  if((xil>=xf)&&(yil>=yf)&&(xfD>=xf)&&(yfD>=yf)){
24                  temp=1;
25                  }
26       }
27                drawItemsIntoRect(xi,  yi, xf, yf);
28 }
```

Método encargado de graficar los elementos al interior del vehículo

Caso de Uso	Seleccionar fichero Output	**Identificador**	CU_01
Tipo	Primario		
Actor(es)	Usuario		
Propósito	Seleccionar el fichero output generado por el algoritmo 2L-CVRP		

CURSO TÍPICO DE EVENTOS	
Precondiciones Existencia del fichero output	
Poscondiciones Desencadenar funcionalidad del caso de uso **Analizar contenido fichero**	
Usuario 1. Ingresar a la solución 2. Presionar el botón seleccionar archivo 3. Buscar el archivo y seleccionarlo 4. Presionar el botón cargar	**Solución** 5. Analizar formato del fichero seleccionado 6. Desencadenar funcionalidad del caso de uso **Analizar contenido fichero**
CURSOS ALTERNATIVOS (EXCEPCIONES)	
Formato de fichero output incorrecto	
Precondiciones Presionar el botón cargar Analizar formato del fichero seleccionado	
Poscondiciones Notificar al usuario la inconsistencia en el formato del fichero	
Usuario	**Solución** 1. Al analizar el formato del fichero output y de encontrarse irregularidades en el formato esperado, un mensaje de error debe visualizarse al usuario indicándole el formato permitido por la aplicación
Error en la selección del fichero y/o análisis del formato	
Precondiciones	
Poscondiciones Notificar al usuario el error sucedido, interrumpiendo la ejecución	
Usuario	**Solución** 1. Al generarse cualquier tipo de error en el proceso de selección o análisis del fichero, debe reintentarse el proceso, y de persistir, debe notificarse al usuario interrumpiendo el proceso de ejecución

Formato de documentación de los casos de uso