APLICACIÓN CON GOOGLE MAPS

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Juan Ruperto García Agudo
i dirigit per
Joan Sorribes Gomis
Bellaterra,16 de setembre de 2009

Universitat Autònoma de Barcelona

etse

Escola Tècnica Superior d'Enginyeria

El sotasignat, ...

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat: .....

Bellaterra, ...de.....de 200.....

Índice

Contenido

1.- INTRODUCCIÓN

1.1-. Tema del proyecto

En los últimos años, la proliferación de los sistemas GPS portátiles ha hecho que dichos dispositivos pasen a formar parte de nuestra vida cotidiana. Desde la aparición de los primeros sistemas, destinados únicamente a mostrarnos las coordenadas relativas a nuestra posición, hasta los últimos modelos de teléfonos móviles dotados de GPS con imágenes reales y pasando por los modelos destinados a automóviles, son muchos los distintos tipos de dispositivos disponibles hoy en día. Esto, unido a la constante bajada de sus precios, junto con la inclusión de diferentes funcionalidades (radio, mp3, detector de radares, manos libres, integración con diferentes servicios de fotografía, etc., etc.) Hace que cada vez sea más común el uso de estos dispositivos diferentes aspectos de nuestra vida.

En cuanto a lo que se refiere a las aplicaciones web, este hecho no podía pasar desapercibido. Coincidiendo con el auge de las redes sociales, son cada vez más los sitios web que ofrecen compartir las diferentes rutas que los usuarios de dichas webs suben, animados por la facilidad de uso de los dispositivos en lo que se refiere su utilización para planificar rutas, así como la cantidad de software disponible para exportar los recorridos realizados en diferentes formatos de archivo. A todo esto ha contribuido de manera crucial el gigante Google y su filosofía de permitir a todos los usuarios el trabajar con todos sus productos mediante API's desarrolladas por ellos mismos. De este modo, en junio de 2005, tan sólo cuatro meses después de anunciar el servicio Google Maps en fase beta, se disponía ya de una primera versión de la API de Google Maps, abriendo así la veda para la inclusión de dicho servicio en cualquier pagina web haciendo modificable casi cualquier aspecto de la interfaz original.

Este hecho, unido a la necesidad de aumentar las funcionalidades (en principio únicamente para temas relacionados con el ocio personal) de éstos aplicativos webs que utilizan estas API's, es lo que nos ha llevado a idear, analizar y desarrollar un aplicativo como el que se describirá a continuación.

1.2.- Objetivos

El principal objetivo de este proyecto es realizar un aplicativo web que utilice las API's anteriormente citadas para que, a partir de un conjunto de rutas almacenadas en el sistema, éste sea capaz de generar nuevas rutas a partir de éstas, de modo que dicho sistema no se comporte como un mero "almacén" de rutas introducidas por los usuarios para poder ser utilizadas por los demás posteriormente, sino que podamos ser nosotros mismos los que decidamos ciertos aspectos de la ruta que deseamos que el sistema nos proporcione. Así, el aplicativo desarrollado, partiendo de un conjunto previo de rutas, será capaz de generar nuevas rutas pasando lo más cerca posible de un conjunto de puntos definidos por el usuario, pudiendo ser calculadas siguiendo diferentes parámetros de optimización tales como el menor número de rutas utilizadas en el cálculo o el menor número de intersecciones entre ellas. Estas rutas, posteriormente, pueden visualizarse en la aplicación mediante el API de Google Maps, o pueden ser descargadas para utilizarlas en nuestro dispositivo GPS portátil.

1.3.- Organización de la memoria

La presente memoria está dividida en tres grandes bloques. El primer gran bloque, que incluye este primer capítulo y los dos siguientes, se pretende introducir la aplicación dentro de su contexto para entender mejor cuáles son sus pretensiones. También es inevitable hacer un repaso de la parte teórica necesaria para abordar la solución y como, en base a este conocimiento teórico y los requisitos del aplicativo, explicar qué tipo de solución se ha escogido y el porqué de esta elección.

En el siguiente gran bloque está compuesto por el capítulo cuarto y en él se procede a explicar el análisis y el diseño del aplicativo según la solución escogida. Se argumentarán aspectos de la elección tales como el lenguaje escogido, APIS's a utilizar, librerías externas, etc. Dentro del diseño de la aplicación se describirán cada uno de los diferentes módulos que la componen, y, entre ellos, el más importante, el algoritmo de cálculo de rutas, piedra angular de todo el proyecto.

El ultimo gran bloque, comprendido entre los capítulos quinto y sexto, incluye una descripción del uso del aplicativo, la enumeración del juego de pruebas a que ha sido sometido el sistema y un apartado en el que se mencionan las diferentes líneas de continuidad a las que podría llevarnos el proyecto de cara a futuras ampliaciones de funcionalidades así como las conclusiones finales del proyecto.

2-. INTRODUCCIÓN A LOS SISTEMAS DE INFORMACIÓN GEOGRÁFICA (SIG)

Un **Sistema de Información Geográfica** (SIG, o GIS en su equivalente en inglés) se define como un sistema que integra elementos de software, hardware y datos geográficos y que proporciona la capacidad de capturar, almacenar, manipular y mostrar toda esta información georeferenciada para solucionar problemas de gestión o planificación. Genéricamente, también se denomina SIG a las herramientas que permiten utilizar esta información para editar o crear mapas, realizar consultas interactivas y mostrar los datos de estas operaciones.

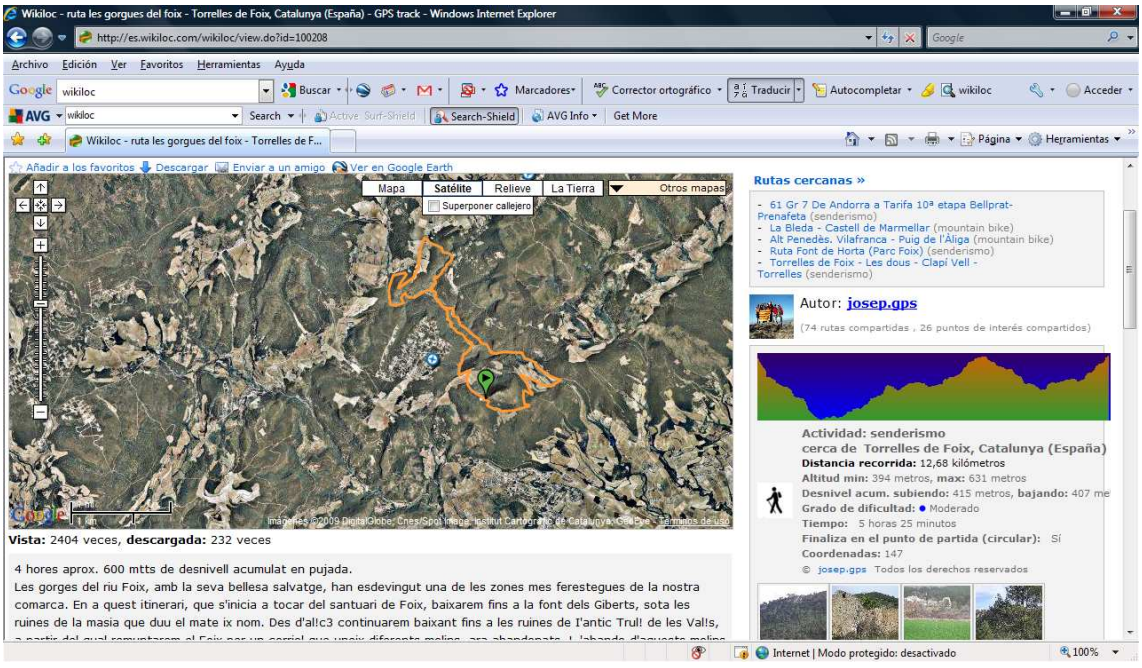


Fig 1. Sistema de Información Geográfica

2.1 Funcionamiento

Un sistema de información geográfica no es más que una base de datos con información geográfica cuyos objetos, al igual que otros tipos de bases de datos, están referenciados por un identificador común. De este modo, podemos consultar este objeto en la base de

datos cartográfica y obtener todos sus atributos. La razón para utilizar un SIG radica fundamentalmente en la gestión de este tipo de información. Un SIG permite separar la información en diferentes capas, y posibilita trabajar con cada una ellas independientemente, y generar información que de otro modo no podríamos obtener.

Un SIG puede solucionar los siguientes aspectos:

Localización: Un SIG ha de poder proporcionar información relativa a las distintas características de un lugar concreto.

Condición: Hacer cumplir al sistema unas determinadas condiciones.

Tendencia: Comparación espacial o temporal de alguna característica completa. (Mapas de temperatura anual, etc.)

Rutas: Cálculo de rutas óptimas entre dos o más puntos.

Pautas: Detección de pautas espaciales.

Modelos: Generación de modelos de datos a partir de actuaciones o fenómenos simulados.

2.2. Creación de los datos.

En los sistemas modernos, la captura de datos se realiza mediante programas de diseño asistido por ordenador (CAD), que cuentan con una serie de capacidades añadidas de georeferenciación. Dada la gran cantidad de ortofotografías (fotografías de una superficie terrestre con la validez de un plano cartográfico) existentes hoy en día, la mayoría de estos procesos consisten en digitalizar la información necesaria utilizando directamente como fuente de información este tipo de fotografías.

2.2.1 Representación de los datos.

En estos sistemas, lo que se representan son objetos correspondientes al mundo real, tales como carreteras, vías de tren, ciudades, etc.

Se pueden hacer dos tipos de abstracciones con estos datos:

Objetos discretos: Se engloban dentro de esta clase objetos como una casa, un puente, etc.

Objetos continuos: En esta clase de datos estarían englobados objetos más abstractos tales como una elevación de terreno, la cantidad de lluvia caída en un lugar, etc.

Estos tipos de objetos, a su vez, se pueden almacenar de dos maneras diferentes: Ráster y vectorial:

Ráster:

El tipo de datos ráster es esencialmente una imagen digital representada como una matriz. Se centra principalmente en las propiedades del espacio más que en la precisión de su localización. En este tipo de representación, se divide el espacio en una matriz y cada uno de sus elementos representa un único valor.

Esta estructura de datos está formada normalmente por un conjunto de filas y columnas donde cada elemento almacena un valor y que puede ser de cualquier tipo (continuo o discreto). Además, dicha estructura puede aumentar en dimensiones para almacenar más cantidad de información si asociamos a cada celda de la representación una tabla con un conjunto de atributos a los que referirse a cada celda.

Ventajas:

Estructura de datos simple.

Operaciones de superposición sencillas

Óptimo para variaciones altas de datos

Buen almacenamiento de imágenes digitales

Inconvenientes:

Cantidad de memoria requerida.

Dificultad de generar reglas topológicas.

Dependiente de la resolución. Las salidas son menos vistosas.

Vectorial:

Este tipo de representación se centra principalmente en mantener la precisión de los datos digitalizados sobre el espacio, y preserva las características geométricas de los

objetos que representa, normalmente en forma de vectores. Cada uno de los elementos representados está asociado a una base de datos donde tenemos una fila por cada uno de ellos, con todos sus atributos. (Por ejemplo, una base de datos de alturas)

Para generar cada elemento se suele utilizar lo que se denomina una red irregular de triángulos: Es una representación mediante una estructura de datos generada a partir de procesos de triangulación de una superficie continua.

Una malla TIN conecta una serie de puntos con coordenadas (x,y,z) mediante una red de triángulos cuyos vértices corresponden con dichos puntos.

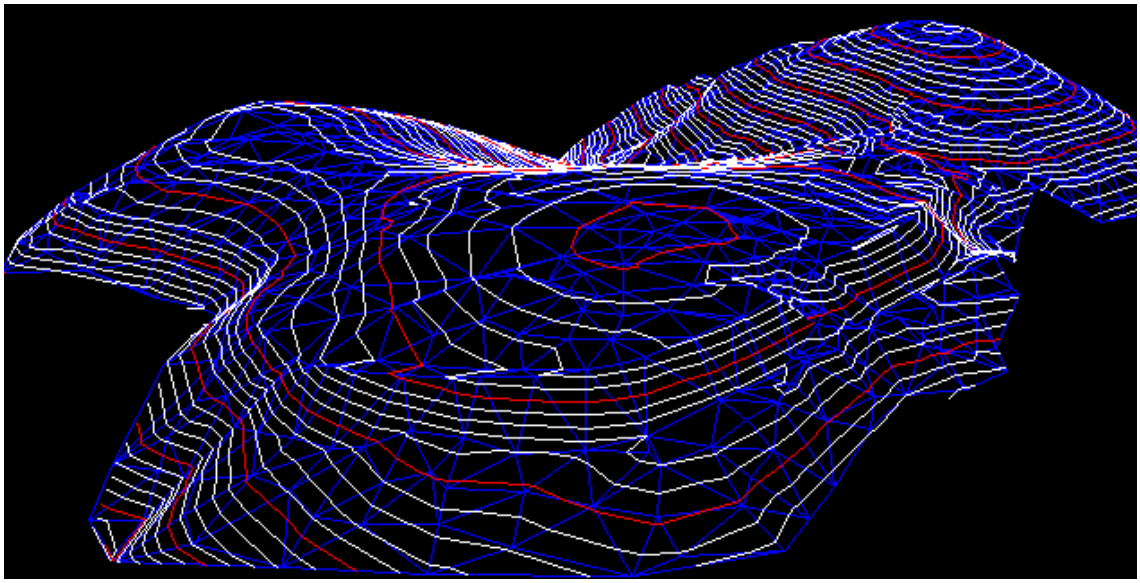


Fig. 2. Representación de una superficie tridimensional mediante una malla TIN.

Para representar estos objetos del mundo real se utilizan los siguientes elementos geométricos:

Puntos: Se utilizan para representar las entidades geográficas que pueden ser representadas por un único punto de referencia: Ubicaciones de objetos (ciudades en un mapa, cimas de elevaciones, o puntos de interés)

Líneas: Las líneas son utilizadas para representar objetos tales como vías de tren, ríos, carreteras, curvas de nivel…Este elemento posibilita la medida de la distancia entre dos objetos.

Polígonos: Se utilizan para representar objetos geográficos que cubren un área particular de la superficie de la tierra, pueden representar lagos, provincias, usos del suelo…En este tipo de estructura puede medirse el perímetro y el área

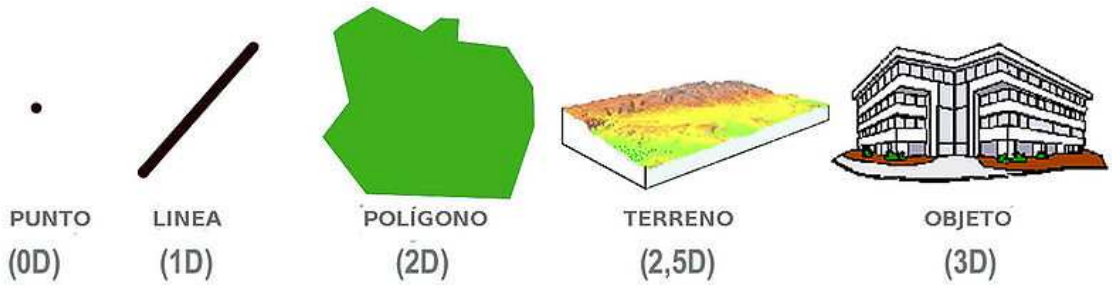


Fig. 3. Dimensión espacial de los diferentes objetos en un SIG.

Ventajas:

Requiere menos memoria para su almacenamiento y tratamiento.

Facilidad para realizar operaciones topológicas y espaciales.

Los gráficos vectoriales proporcionan mejor salida grafica y no se pierde definición.

Compatibilidad con bases de datos relacionales.

Las operaciones de escalado, proyección, etc., son más fáciles de realizar.

Mantenimiento de los datos.

Permite mayor capacidad de análisis de los datos.

Inconvenientes:

Estructura de datos compleja.

Las operaciones de superposición son más difíciles de implementar.

Si la variación en los datos es alta, su eficacia disminuye.

Actualización costosa.

Cantidad de información limitada.

2.2.2 Captura de datos.

La captura de los datos es el proceso que ocupa la mayor parte del tiempo cuando se desarrolla un sistema GIS.

La captura de los datos puede hacerse por las siguientes vías:

Datos obtenidos a través de mapas en formato papel, que posteriormente serán digitalizados para obtener datos vectoriales. Este proceso ha de ser revisado con posterioridad para obtener el nivel de precisión deseado.

Mediciones tipográficas: Pueden ser introducidas directamente en un SIG a través de instrumentos de captura de datos. También, directamente, coordenadas tomadas a través de un sistema de posicionamiento global (GPS)

Sensores remotos: Se pueden capturar datos utilizando cámaras, scáners, etc., acoplados a plataformas móviles como satélites, aviones, etc., siendo, actualmente, la interpretación de fotografías áreas de donde provienen la mayoría de los datos utilizados en la mayor parte de los sistemas SIG, utilizando herramientas que digitalizan los datos a través de pares estereoscópicos de fotografías digitales.

Observación por satélite: En este caso, los satélites utilizan diferentes sensores para medir diferentes partes del espectro electromagnético. Esto genera un tipo de datos ráster que puede procesarse utilizando diferentes bandas para determinar diferentes aspectos de interés.

Errores en la captura de datos:

En este punto, se ha de tomar un compromiso con la exactitud deseada de los datos a capturar, ya que una mejora en la precisión influye tanto en la interpretación posterior de la información como en el coste de su captura.

Durante los procesos de digitalización, es corriente que se den una serie de errores involuntarios y que deberán ser corregidos en un proceso posterior de "corrección topológica".

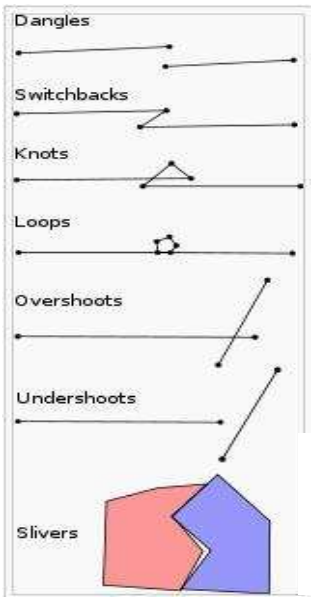


Fig 4. Errores topológicos y de digitalización.

. Conversión entre formatos ráster y vectorial

Un sistema SIG deber ser capaz de realizar conversiones entre estos dos tipos de datos. Normalmente, se hace utilizando las siguientes técnicas:

Ráster-Vectorial: Se recurre a técnicas de mejora del contraste, y el diseño de filtros mediante las transformadas de Fourier de dos dimensiones.

Vectorial-Ráster: Se utiliza el proceso de rasterización: se utilizan algoritmos utilizados en la generación de gráficos por ordenador tales como algoritmos de recorte, del z-buffer…

Proyecciones y sistemas de coordenadas:

Para poder representar los datos una vez capturados, primero se han de reproyectar todos a una misma proyección para que pueda para que puedan ser integrados dentro de un SIG.

Para representar estos datos de un modelo tridimensional (como es la tierra) a otro bidimensional como pueden ser una pantalla o un papel, se utilizan distintas proyecciones, dependiendo del mapa que se desea crear, y dependiendo de su uso.

Algunos tipos de proyecciones más usadas son las siguientes:

Proyección cilíndrica: En esta proyección, se proyecta el globo terrestre sobre una superficie cilíndrica. Es de las proyecciones más usadas, aunque de forma modificada, debido a las grandes distorsiones que presenta en zonas de latitud elevada. Algunos tipos de proyecciones cilíndricas son la de mercator y la de peters, siendo la primera de gran importancia, porque es ampliamente utilizada en la elaboración de planos terrestres y de navegación.

Proyección cónica: En este tipo de proyección se proyecta la tierra sobre una superficie cónica situando el vértice en el eje que une los dos polos. Esta proyección suele usarse en cartografía para ver los países y continentes.

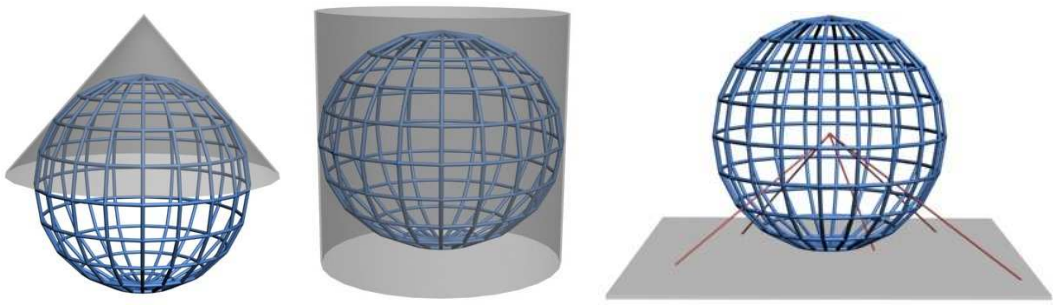


Fig. 5. Diferentes tipos de proyecciones: cónica, cilíndrica y cenital

Proyección azimutal o cenital: Se proyecta la tierra sobre un plano tangente al globo en un punto seleccionado. Así, se obtiene una visión de la tierra desde un punto exterior (proyección ortográfica) o interior (proyección gnomónica). Se utiliza principalmente para proyectar los polos y hemisferios.

Un sistema de información geográfica utiliza la potencia de procesamiento de un ordenador para transformar toda la información recibida en diferentes formas y proyecciones durante el proceso de captura para, finalmente, transformarla a un formato, una proyección y un sistema de coordenadas común. A todo este proceso, se le llama rectificación.

2.3. Software GIS

Existen numerosos sistemas GIS que realizan todas las tareas anteriormente mencionadas, algunos pertenecientes a empresas comerciales, como Mapinfo o Autodesk, y otros, de licencia GNU, de código abierto y libres, promovidos por gobiernos e universidades, tales como LocalGis o Capaware.

Hoy en día, el acceso a estos datos georeferenciados esta referenciado principalmente por herramientas on-line, como Google Maps, Google Earth y otros, utilizando tecnologías de web-clipping.

Es a finales de los noventa, con el acceso mayoritario a internet, cuando se da un cambio de perspectiva en el mundo de los sistemas GIS, pasando de comercializarse como un sistema cerrado y prácticamente inmodificable a un sistema compuesto por una combinación de aplicaciones interoperables y APIS, que posibilitan al usuario modificar algunos aspectos a su gusto.

Últimamente, con la inclusión masiva de chips GPS en dispositivos móviles de última generación, el campo de los sistemas GIS ha dado un salto espectacular, permitiendo no solo un gran crecimiento del número de ventas de estos dispositivos, sino un avance a todos los niveles en lo que un sistema GIS se refiere: (análisis de datos, adición de nuevas funcionalidades, etc.) Baste decir que, según un estudio de la empresa de cartografía digital TeleAtlas, le venta de dispositivos GPS en el año 2005 creció un 264% con respecto al año anterior, y, en este año, se duplicarán las ventas del año anterior, alcanzando un total de 1,5 millones de unidades vendidas.

2.4. Ámbitos de uso de los sistemas GIS:

Un Sistema de Información Geográfica puede aplicarse a multitud de ámbitos, utilizando la información que genera para establecer unas pautas con respecto a una serie de parámetros y establecer patrones, modelos de comportamiento y otra serie de datos, que pueden aplicarse a diferentes campos, como por ejemplo:

Control de incendios, estudio de las capacidades agrícolas de un terreno, gestión de emergencias medicas y analizar epidemias…

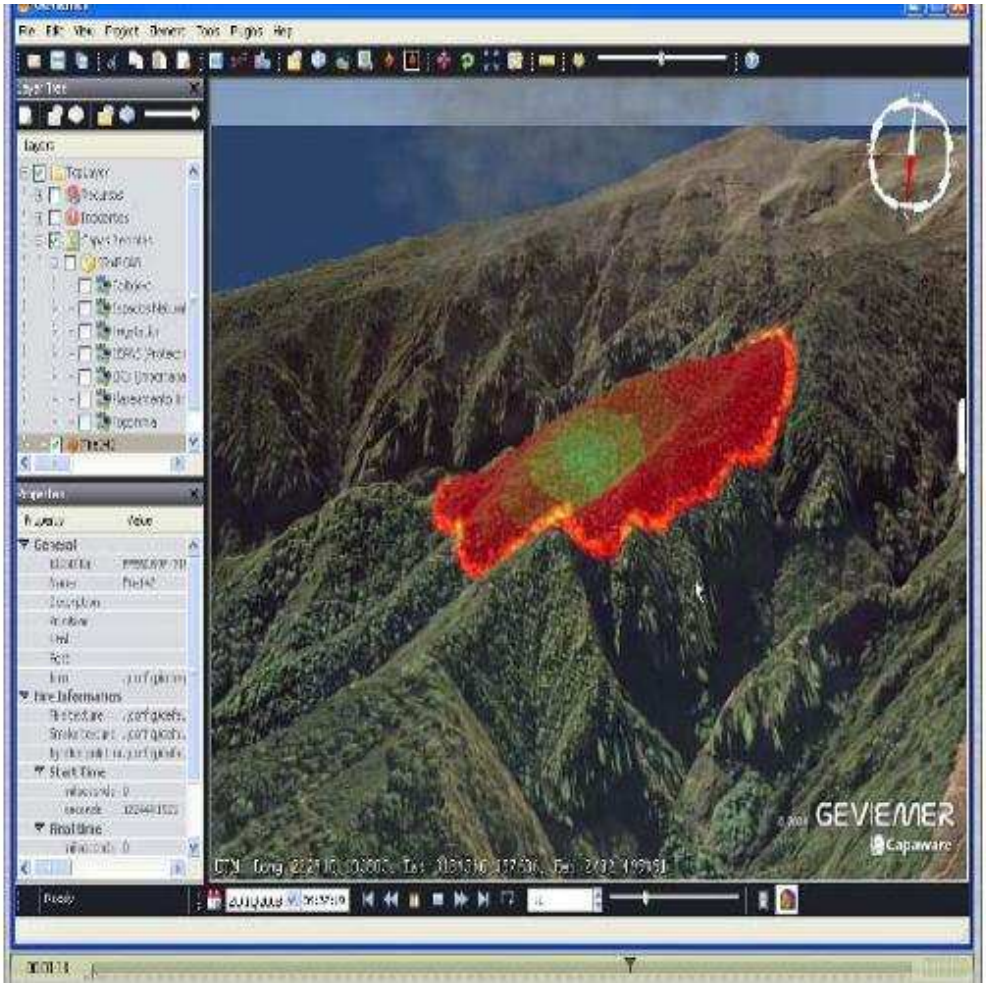


Fig. 6. SIG Capaware utilizado en la simulación de un incendio forestal.

2.5. Futuro de los sistemas SIG:

El futuro de los SIG viene ligado íntimamente con la última aparición de todo tipo de dispositivos móviles con GPS incorporado. Las aplicaciones basadas en los servicios de localización (Location Based Services o LBS), permiten a estos dispositivos GPS mostrar su posición con respecto a un punto de interés concreto tales como restaurantes, teatros, estaciones de metro, de autobús, etc. Pudiendo, incluso, localizar a personas, siendo esto de gran ayuda en personas, por ejemplo, con Alzheimer, comercializándose hoy en día productos en este sentido. Son este tipo de aplicaciones relacionadas con el ocio y la localización de personas, las que en un futuro se prevé que tengan mayor crecimiento.

3. Estudio de viabilidad

3.1. Introducción

Este estudio de viabilidad trata el desarrollo de un aplicativo de generación de rutas GPS a partir de una base de conocimiento previa compuesta de un conjunto de rutas, en vistas a ser utilizada posteriormente como una librería de una aplicación GIS mayor.

3.2. Objeto:

3.2.1. Descripción de la situación a tratar:

Se parte de una situación en la que tenemos un sistema con un conjunto de rutas GPS almacenadas en nuestro sistema.

- Cada ruta esta almacenada en un formato estándar de intercambio de datos en lo que se refiere a dispositivos y aplicaciones GPS
- Las rutas pueden visualizarse en un mapa.

3.2.2. Perfil de los usuarios.

Los usuarios de la aplicación serán de diversos tipos:

- **Administradores:** Que gestionaran la base de conocimiento, introduciendo nuevas rutas en ella y/o generando nuevas rutas a partir de las ya existentes.
- **Desarrolladores:** Usuarios que utilizaran el aplicativo en forma de librería para generar nuevas rutas.
- **Usuarios:** Estos usuarios utilizaran el aplicativo para generar nuevas rutas utilizando las ya existentes por medio de un interfaz web donde podrán parametrizar todos los aspectos de la ruta generada.

3.2.3. Objetivos:

Los objetivos que se pretenden cumplir son los siguientes:

Añadir una nueva funcionalidad a un sistema GIS que:

- Genere nuevas rutas a partir de una base de conocimiento previa del sistema.

- Incluya la posibilidad de seleccionar el punto de inicio y final y un número determinado de puntos intermedios por los que transcurrirá la ruta generada.
- Permita seleccionar el criterio de creación de la ruta.
- Muestre por pantalla la ruta generada, así como las rutas que la componen.
- Posibilite la descarga de estas rutas en un formato estándar de intercambio de datos, para que pueda ser utilizado, manipulado y visualizado en multitud de dispositivos GPS y aplicaciones.

3.3. Descripción del sistema a realizar.

La aplicación consiste en un generador de rutas GPS. Para cumplir su cometido, utilizará una base de conocimiento que se cargará al iniciarse el servidor donde está alojada, y se invocara vía interfaz web para que de este modo permita la interacción con la misma, modificando los diversos criterios disponibles en la creación de la ruta. Además, se hará uso del API de Google Maps para visualizar las rutas generadas y también, si se desea, las rutas que la componen.

Para generar la ruta podrán seleccionarse una seria de criterios, como son:

- Número de puntos que la delimitan(2-6)
- Radio de influencia de cada punto: Se define radio de influencia como el radio de la circunferencia dentro de la cual se consideran las rutas que pasan por ella. A mayor radio de influencia, tendremos más rutas a considerar para encontrar la solución.
- Criterio de evaluación de la ruta: Para evaluar la mejor ruta a mostrar, podemos elegir entre dos criterios:
- Ruta más corta: la mejor ruta es aquella que mide menos distancia entre su inicio y fin
- Ruta con menor número de transbordos: La mejor ruta es aquélla que para calcularla se han utilizado menos rutas de la base de conocimiento.

También se ha añadido una funcionalidad extra que consiste en listar todas las rutas que se encuentran a una distancia determinada del punto de origen. Aparte de ser una funcionalidad útil en algún momento determinado, cumple funciones de test, a la hora de definir un área con un número de rutas suficiente para realizar pruebas.

3.4. Método de desarrollo:

El método de desarrollo a seguir en este proyecto es el modelo lineal secuencial, que comprende los siguientes niveles:

- Análisis
- Diseño
- Generación del código
- Pruebas
- Mantenimiento

3.5. Recursos utilizados

Los recursos necesarios para la realización del proyecto son los siguientes:

Recursos Hardware

Ordenador portátil HP. Procesador Core 2 Duo, 4Gb de memoria.

Conexión a internet

Recursos Software:

Sistema Operativo Windows Vista Home Premium

Servidor Web con soporte para servlets y jsp

Entorno de desarrollo: Intellij Idea y eclipse 3.4

3.6. Elección de tecnologías:

- Lenguaje de programación: Java, ya que es el más utilizado para este tipo de aplicaciones.

- Base de conocimientos en disco: La base de conocimiento no será una base de datos, sino un conjunto de rutas en un directorio determinado de disco.

- Navegadores: Internet Explorer y Google Chrome: el primero por ser el cliente web más extendido, y el segundo, porque proporciona mayor rendimiento a la hora de visualizar los mapas generados utilizando el api de Google Maps.

4. Análisis

4.1. Análisis funcional

4.1.1. Requerimientos funcionales

Los requerimientos funcionales que del aplicativo que se han de presentar son los siguientes:

- Proveer al usuario de una interfaz web para que pueda interactuar con el sistema
- El sistema debe generar las rutas en un formato estándar entendible y utilizable por la mayoría de dispositivos GPS y aplicativos GIS.
- El sistema debe estar abierto a ampliarse en un futuro añadiendo nuevas funcionalidades.
- Su arquitectura no debe estar sujeta a ningún tipo específico de fuente a usar como base de conocimiento, pudiendo ser fácilmente adaptable para funcionar con cualquier fuente de datos.

4.1.2. Requerimientos no funcionales

En cuanto a los requerimientos funcionales de la aplicación a realizar, podemos decir lo siguiente:

- Ha de seguir las convenciones en cuanto a codificación java se refiere.
- Fácil navegabilidad
- Usabilidad
- Han de mostrarse los resultados en unos tiempos de respuesta aceptables.
- Se utilizara para la codificación del HTML del proyecto hojas de estilo en cascada o CSS, ya que puede de este modo diferenciarse el formato del resto de características del diseño de las páginas web, a la vez que se crea un formato uniforme para cada una de ellas.

4.1.3. Requerimientos Hardware

Para la realización de este proyecto no se requiere ningún componente hardware específico, salvo que quieran utilizarse las rutas generadas en un dispositivo GPS, para lo cual necesitaríamos un cable conector del dispositivo al ordenador del usuario, o cualquier otro componente HW que posibilite la conexión (dispositivo USB, infrarrojo, etc.) para transferir la ruta generada.

4.1.4. Requerimientos Software:

Se requiere:

- Servidor de aplicaciones web J2EE compatible
- Base de conocimiento de rutas.

En los clientes:

- Navegador web: Internet Explorer, Google Chrome, Mozilla

4.1.5. Diagrama de alto nivel:

La siguiente figura ilustra a gran escala el funcionamiento del aplicativo y cómo interactúan los diferentes elementos que la conforman:

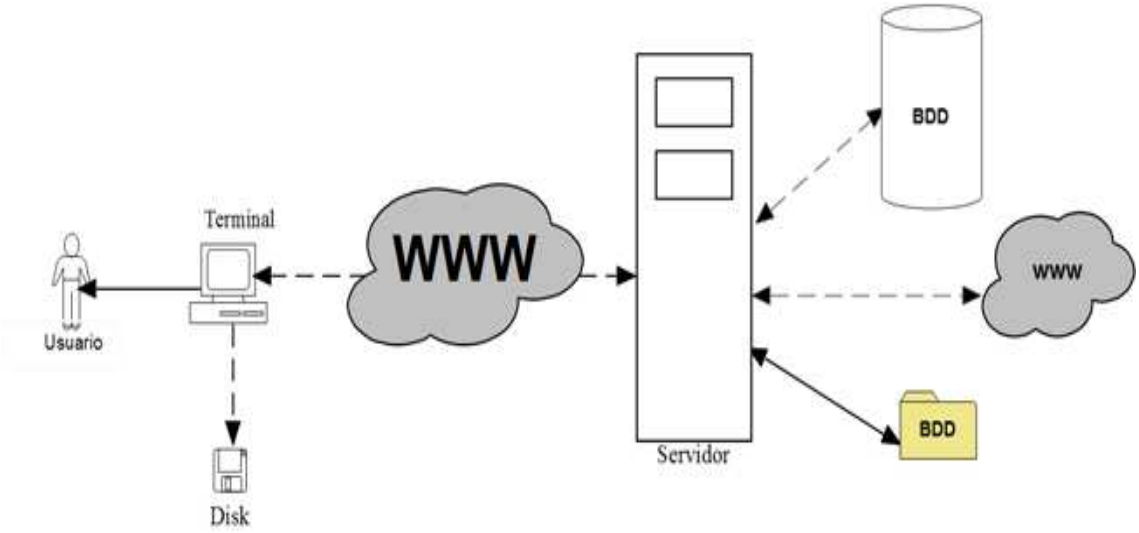


Fig 7. Diagrama de alto nivel.

Los usuarios se conectaran a la aplicación utilizando un navegador y a través de él se accederá a un servidor donde está alojado el aplicativo. Una vez se muestra la pantalla principal, el usuario puedo modificar los parámetros que desee para generar la ruta y, una vez creada, se muestra de nuevo en la pantalla, ofreciendo la posibilidad de descargarla a disco. Es importante reseñar que aun cuando para la realización de este proyecto se ha optado por una base de conocimientos alojada en un directorio de disco del servidor (se ha optado así para poder utilizarla a nuestro antojo) nada impide utilizar una base de conocimiento ubicada en cualquier otro sitio, como una base de datos externa, otro servidor, etc.

4.1.6. Diagrama de casos de uso:

El actor principal es el usuario.

Una vez que el usuario ha accedido a la aplicación, puede realizar las siguientes funciones:

- Refrescar la base de conocimiento (se regenera el grafo de intersecciones entre rutas)
- Calcular rutas cercanas a un radio definido de un punto en concreto.
- Definir una búsqueda de ruta que pase por unos puntos determinados y dentro de un radio determinado configurando todos los parámetros necesarios,
- Ver el detalle de la ruta calculada así como el detalle de cada una de las rutas cuyos segmentos la componen.
- Descarga de la rutas calculadas a disco.

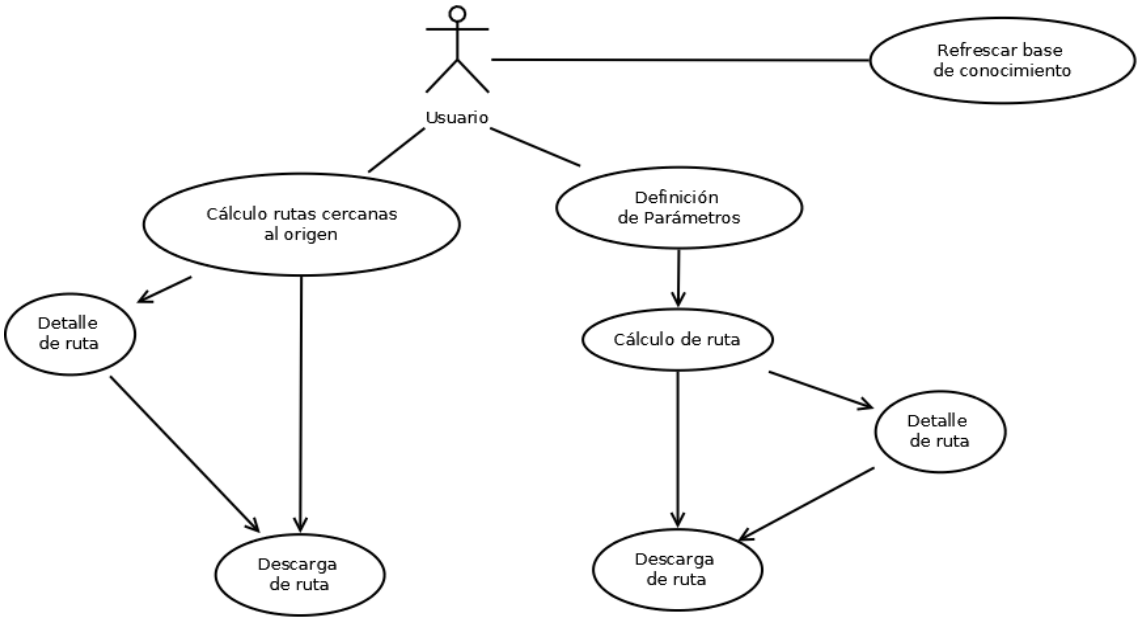


Fig 8. Diagrama de casos de uso

4.1.7. Diagrama de flujo web

A continuación se muestra un diagrama correspondiente a la navegación entre pantallas, Partiendo de la pantalla principal de la aplicación, se mostraran las diferentes pantallas del aplicativo dependiendo de las funciones que utilice el usuario. Cualquier error que se produzca se mostrara como un mensaje en la pantalla correspondiente.

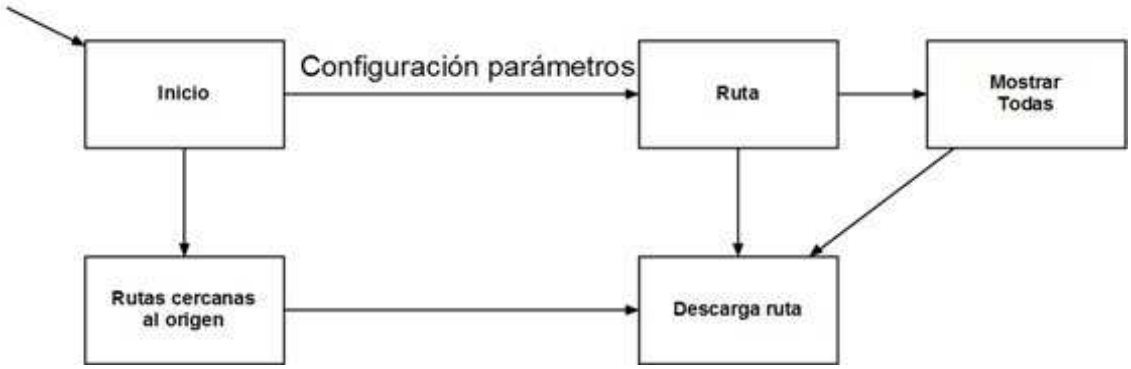


Fig 9. Diagrama de flujo web.

4.1.8. Diseño de la interfaz web

El diseño de la interfaz no ha sido un objetivo prioritario del proyecto, es por esto que para el diseño de las pantallas se ha utilizado una hoja de estilo. Actualmente existen en Internet muchos sitios desde los que es posible descargar estas plantillas sin ningún tipo de coste y adaptar el *look&feel* al estilo deseado por el cliente.
Concretamente el diseño utilizado en este proyecto se ha obtenido de
http://www.free-css-templates.com/preview/DKBlog/

4.2. Análisis técnico.

En este apartado se detallaran la arquitectura elegida, el diagrama de clases de la aplicación y el diseño del interfaz web de la misma.

4.2.1. Elección de tecnologías.

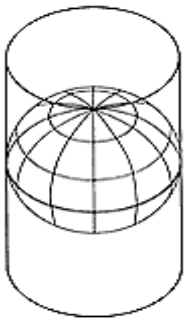
Una vez llegados a este punto, y una vez se tiene un análisis de todas las funcionalidades que debería tener la aplicación, se han de tomar una serie de decisiones que afectaran al diseño de la arquitectura y a la forma de solventar los diferentes problemas que se nos presentarán, de acuerdo con el diseño presentado.
A continuación se enumeran las decisiones que se han tomado en lo que se refiere a la elección del lenguaje de programación, sistema de proyección a utilizar, etc.

4.2.1.1. Lenguaje de programación escogido: Java. Se utiliza este lenguaje por ser el lenguaje mayoritario utilizado en la realización de aplicativos web, además de contar con una fuente casi inagotable de recursos gratuitos en internet (librerías, manuales, etc.) que nos servirán como material de apoyo y nos ayudaran a realizar la implementación más rápidamente.

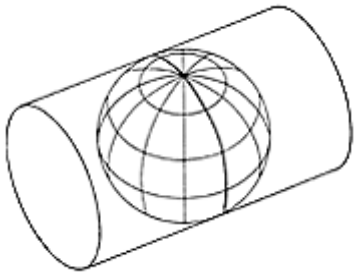
4.2.1.2. Sistema de coordenadas elegido: Sistema de Coordenadas Universal Transversal de Mercator (**UTM**). Se utiliza este sistema de coordenadas por ser una proyección cartográfica ampliamente utilizada en los sistemas GIS y que ofrece una seria de beneficios a la hora de utilizarlo. A continuación se explica este sistema de coordenadas con más detalle:

Sistema de coordenadas UTM: El sistema de coordenadas UTM (*Universal Transverse Mercator*, en Ingles) se basa en una proyección de mercator tangente a un

meridiano. Se diferencia del sistema de coordenadas tradicional, expresado en latitud y longitud, que el sistema de coordenadas UTM expresa sus magnitudes utilizando el sistema métrico.



Proyección Mercator



Proyección Transversal de Mercator

Fig. 10. Proyecciones Mercator

La proyección de Mercator es bastante precisa en puntos próximos al ecuador, pero pierde precisión rápidamente cuando nos alejamos de él. Esto es un gran problema, ya que las zonas más ampliamente pobladas (y por tanto, las que más se suelen mapear) no están cercanas al ecuador en dirección Oeste-Este, sino mas bien en dirección Norte-Sur.

Para solucionar este problema, se utiliza la Proyección Transversal de Mercator, que se hace tangente a un meridiano de la tierra. Sin embargo, no solucionamos el problema de la precisión. Es por eso que para realizar la Proyección UTM se hace lo siguiente:

Para realizar esta proyección, se divide la zona en 60 zonas de 6 grados de longitud, y para cada una de las zonas se aplica una proyección de Transversa de Mercator.

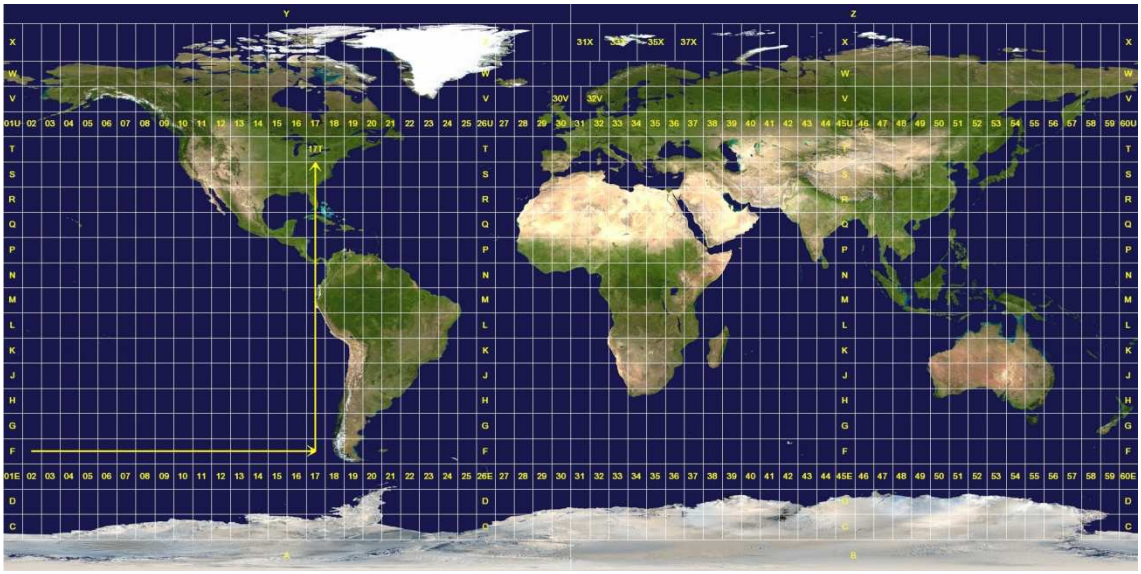


Fig 11. Zonas y husos UTM

Husos UTM: Son 60 husos que dividen la tierra cada 6 grados de longitud, numerados del 1 al 60. (A España, por ejemplo, le corresponden los usos 28 (para Canarias) 29, 30 y 31.

Zonas UTM: Se dividen en 20 zonas de 8 grados de latitud, con las letras de la C a la X (sin la I, O, Ñ). Las zonas polares no se consideran. Para estas zonas se utiliza el sistema UPS. La letra N separa las zonas que se encuentran en el hemisferio norte de las del sur.

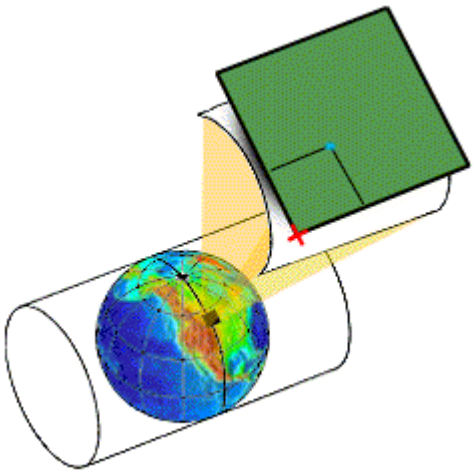


Fig. 12. Detalle de una zona UTM

Ventajas:

- Reduce el mapa a un conjunto de celdas rectangulares.
- No existen números relativos para referirse a una posición en una zona.

- Las coordenadas están basadas en el sistema decimal. No se definen por minutos, segundos, etc.
- Las coordenadas se miden utilizando el sistema métrico.

Desventajas:

- Por la propia definición de la proyección, no pueden combinarse puntos de diferentes zonas UTM, ya que son puntos que han sido proyectados usando distintos sistemas de referencia.

4.2.1.3. Fisonomía del grafo de creación de rutas:

Para la generación de las nuevas rutas a partir de la base de conocimiento, se opta por construir un grafo cuya estructura es básicamente una tabla hash donde para cada ruta mantenemos una lista de todas las intersecciones que tiene con las otras rutas.

Las intersecciones de por sí ya son un grafo, ya que dentro tienen Transiciones que la unen con otras Intersecciones

Una Intersección se une con otra mediante una **Transición** que sigue una ruta y que tiene un coste asociado (la distancia entre una intersección y otra siguiendo la ruta)

Ejemplo de creación del grafo:

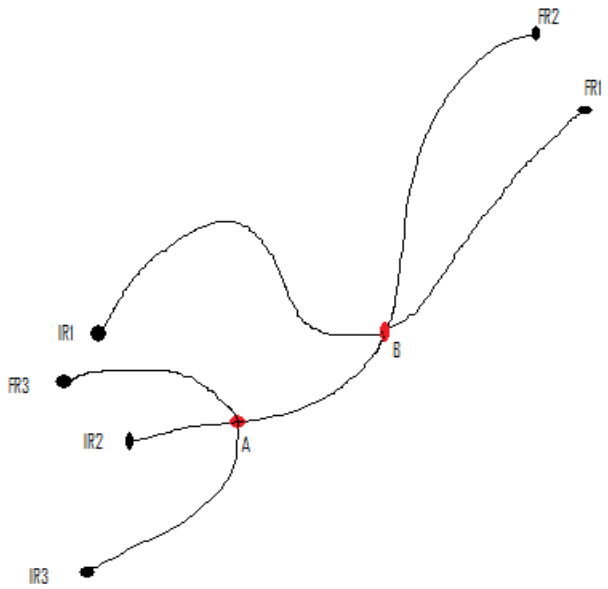


Fig 13. Conjunto de rutas

Si tenemos 3 rutas R1 , R2 y R3 donde R2 se interseca con las demás en los puntos A y B tendremos

Grafo

Id ruta	**Intersecciones con otras rutas**
R1	IR1 *Unida por una transición que sigue R1 con* B Unida por una transición que sigue R1 con FR1
R2	IR2 *Unida por una transición que sigue R2 con* A Unida por una transición que sigue R2 con B Unida por una transición que sigue R2 con FR2
R3	IR3 *Unida por una transición que sigue R3 con* A Unida por una transición que sigue R3 con FR3

Por lo tanto, en este ejemplo, tenemos un grafo compuesto por ocho intersecciones y siete transiciones.

Creación del grafo:

El grafo se crea al iniciar la aplicación o cada vez que utilizamos la opción “REFRESH”

Primero de todo, cargamos la capa de acceso a datos (DAO) a partir del directorio de disco donde tenemos ubicada la base de conocimiento. Este proceso lee los ficheros del disco y va creando una caché a la vez que va serializando los archivos.

Una vez se tienen cacheados y en memoria los ficheros que componen nuestra base de conocimiento, se procede a la creación del grafo de la siguiente forma:

1) Se carga el grafo serializado a disco si existe. Esto se hace así para evitar el cálculo del grafo cada vez que se arranca el servidor donde reside la aplicación.
2) Mira si En la capa de acceso a datos hay alguna ruta más de las que hay en el grafo serializado, lo que implicaría que se han añadido mas rutas
3) Calcula el grafo entre las rutas. Para hacer esto, se hacen combinaciones de rutas y por cada pareja de rutas que componen la base de conocimiento se buscan sus intersecciones.
4) Serializamos el grafo calculado a disco. Así evitamos tener que volverlo a calcular cada vez que se reinicia el servidor.
5) Calculamos las transiciones de forma ordenada, añadiendo costes asociados de ir de una intersección otra.

Búsqueda de caminos:

Una vez tenemos todo el grafo completado, para calcular las rutas se hace de la siguiente manera.

1) Se toman los puntos de inicio y final y los radios de influencia de cada uno y se buscan todas las rutas a estos puntos. Obtendremos una lista de puntos.

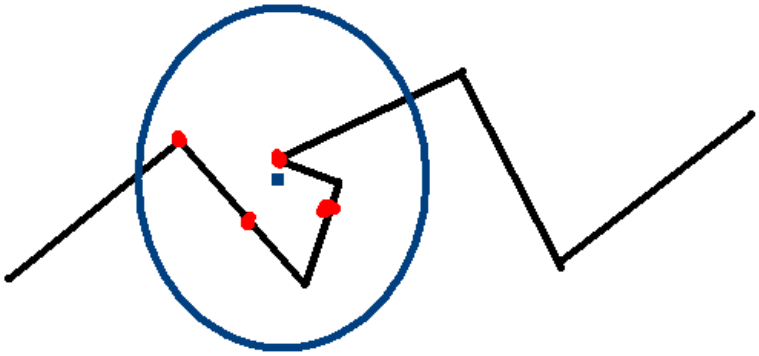


Fig 14. Creacion del grafo.

2) Construimos un grafo “ampliado” donde añadimos una transición por cada uno de los puntos encontrados al grafo que habíamos calculado previamente.
3) Se implementa un algoritmo de Dijkstra para encontrar una ruta que cumpla con los requisitos.
4) Obtenemos un path: Conjunto formado de intersecciones e intersección de ruta que los une. (Puntos A,B y C de la figura inferior)

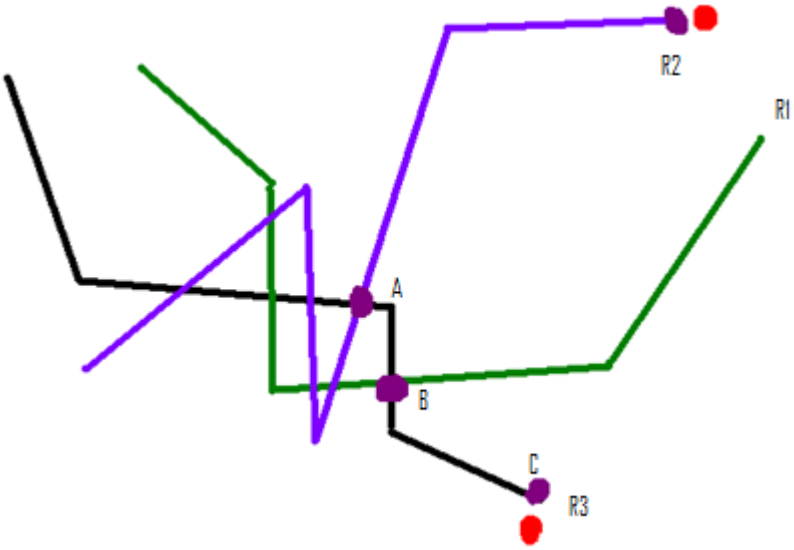


Fig 15. Cálculo del recorrido.

5) Los puntos resultantes se unen en orden inverso para construir la ruta que se mostrará, utilizando la información contenida en el grafo de intersecciones y transiciones, y tendrá más puntos de los mostrados en esta estructura ya que se

han de incluir los puntos de las rutas implicadas, aparte de los de las intersecciones (Puntos A-G en la figura inferior)

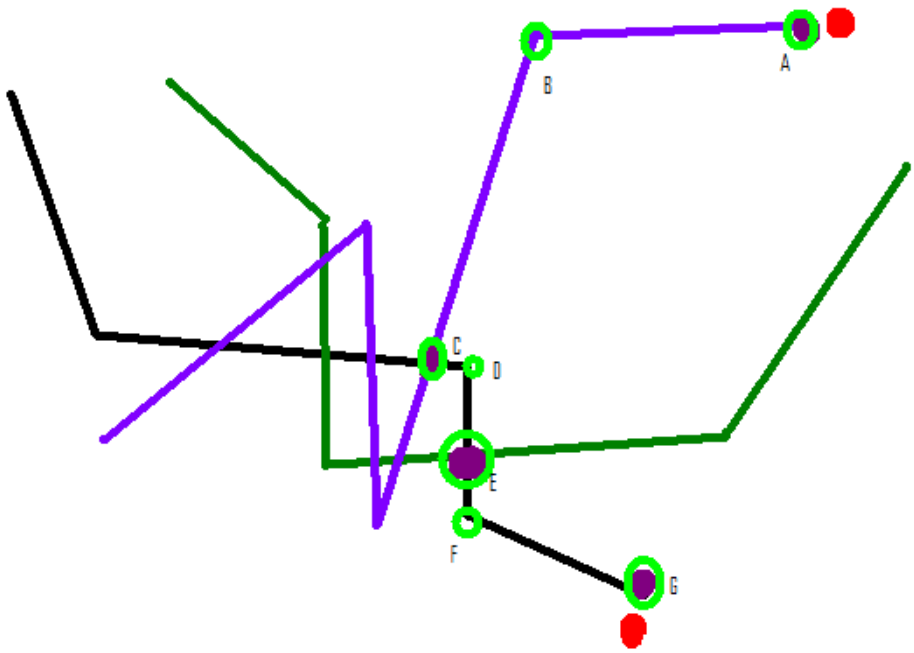


Fig 16. Generación de la ruta.

4.2.1.4. Formato de la base de conocimiento

Para la realización de este proyecto, se ha optado por utilizar una base de conocimiento ubicada en disco. Esto se hace así para agilizar el desarrollo de la aplicación, aún cuando su arquitectura permite, sin embargo, poder usar cualquier tipo de fuente de datos implementando los métodos de acceso a los datos que definirá una interfaz Así pues, definiendo una nueva clase que implemente ese interfaz de acceso a los datos, podremos obtenerlo desde cualquier fuente deseada.

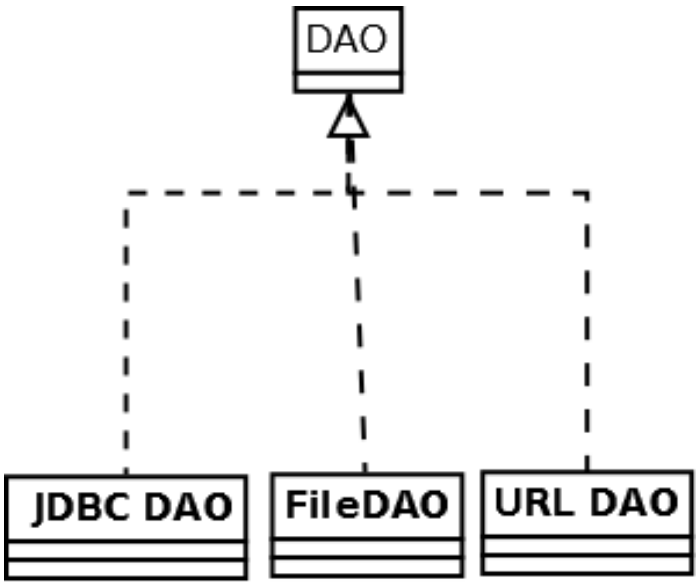


Fig 16. Capa de acceso a los datos.

4.2.1.5. Formato de las rutas:

El formato escogido para las rutas que componen la base de conocimiento así como para las rutas generadas ha sido el formato GPX, ya que es uno de los formatos estándar más utilizados en las aplicaciones GIS, existiendo en el mercado multitud de utilidades que permiten su conversión a otros formatos conocidos.

ESTRUCTURA GENERAL DE UN FICHERO GPX

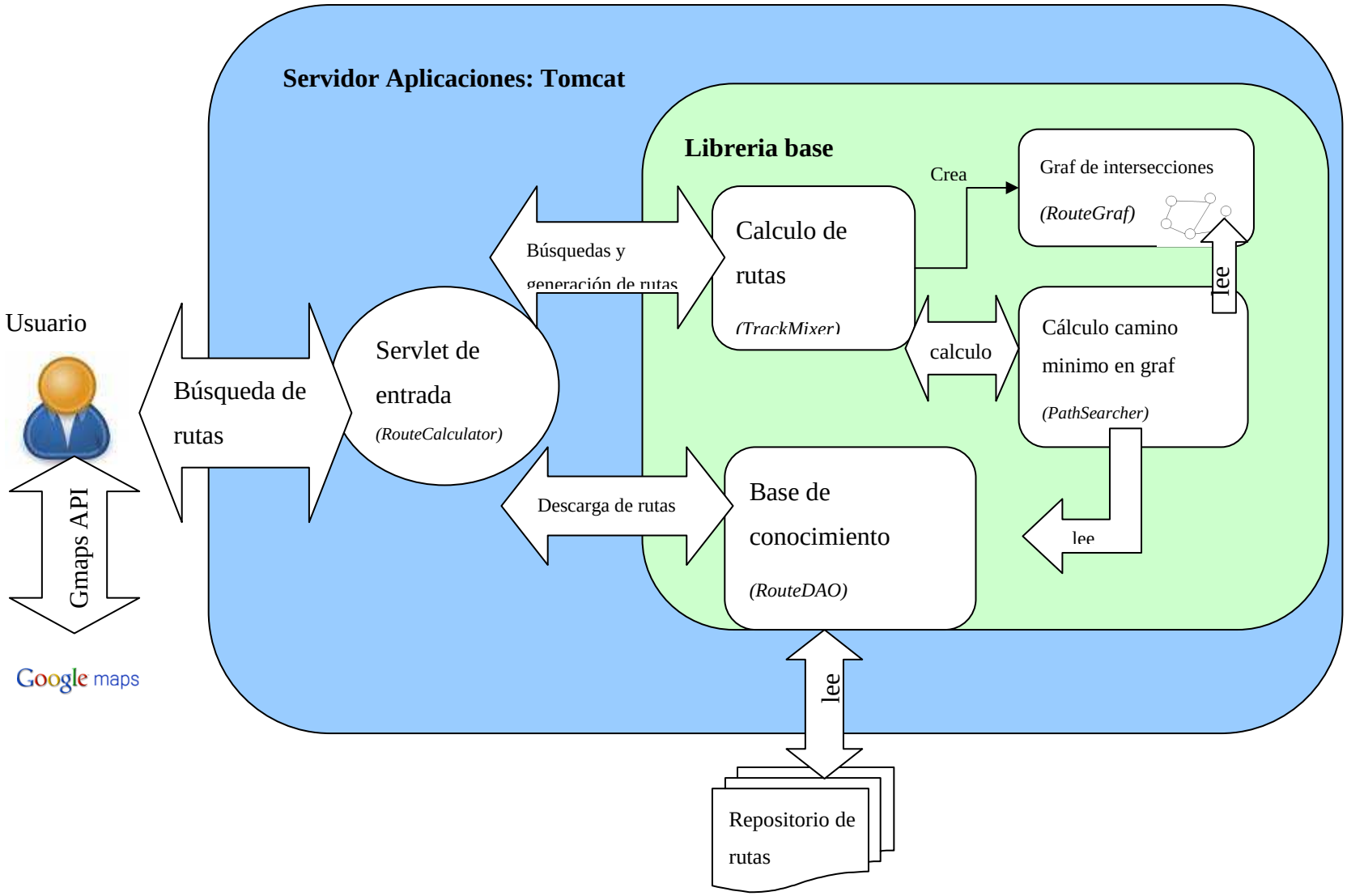
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gpx ...>
Metadatos
<metadata> ... </metadata>
Datos
Ejemplos: Track Waypoint
<trk> <wpt lat="#" lon="#">
<trkseg> <ele>#</ele>
<trkpt lat="#" lon="#"> <name>...</name>
<ele>#</ele> ...
</trkpt> </wpt>
<trkpt ...> <wpt ...>

```
   ...                          ...
  </trkpt>                        </wpt>
 </trkseg>
 <trkseg>
  <trkpt ...>
   ...
  </trkpt>
 </trkseg>
 ...
</trk>
Fin del fichero
</gpx>
```

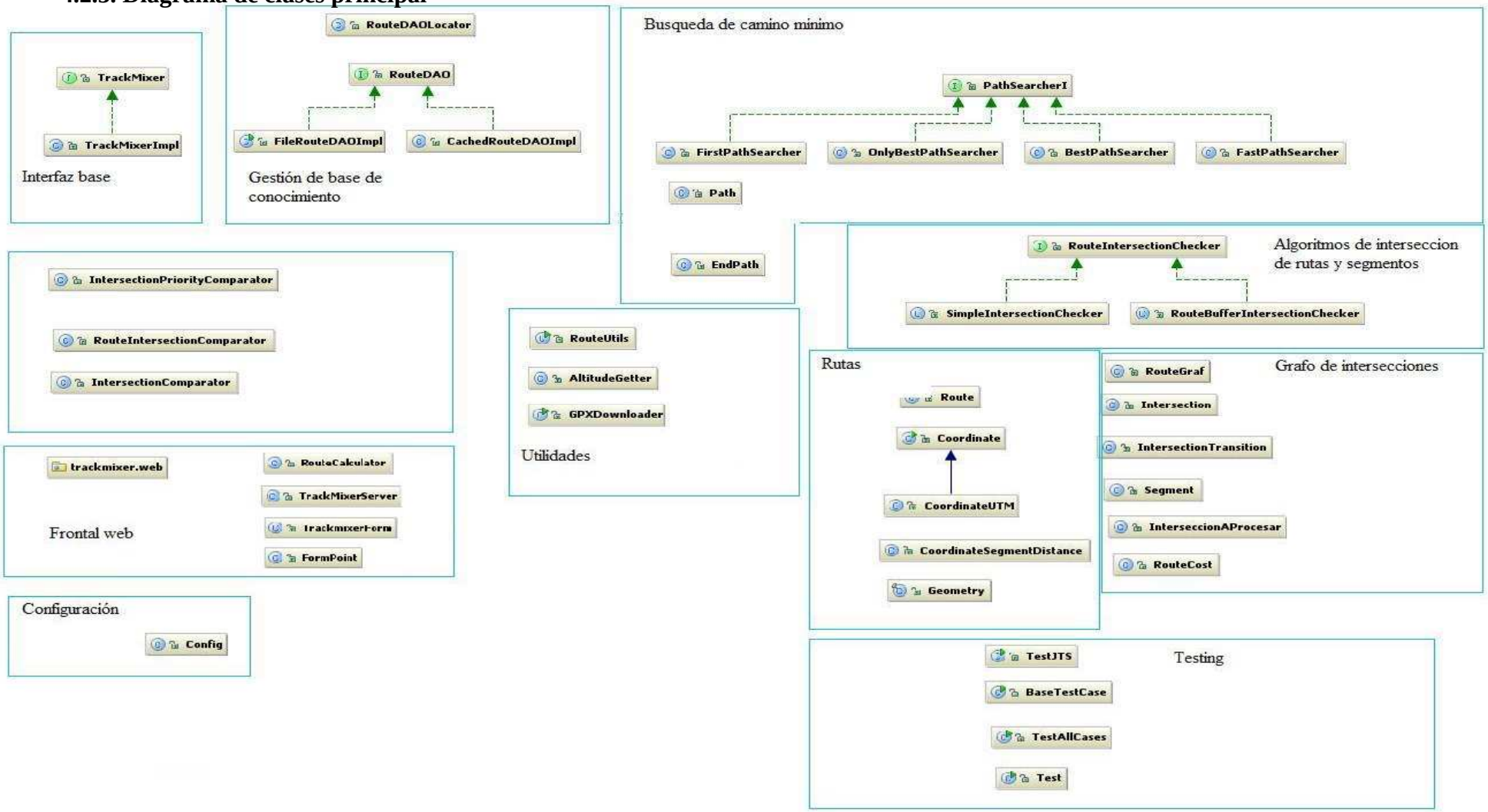
Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" creator="byHand" version="1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd">
  <wpt lat="39.921055008" lon="3.054223107">
    <ele>12.863281</ele>
    <time>2005-05-16T11:49:06Z</time>
    <name>Cala Sant Vicenç - Mallorca</name>
    <sym>City</sym>
  </wpt>
</gpx>
```

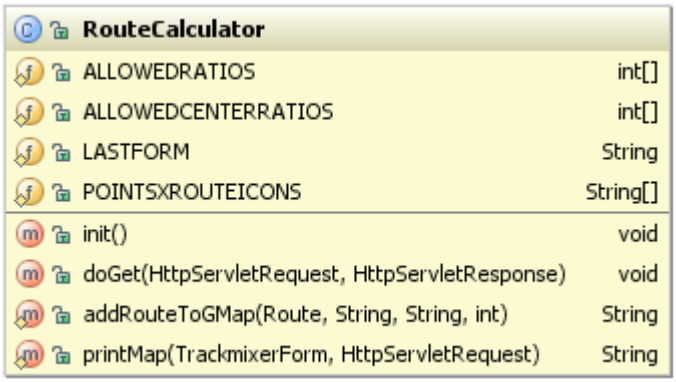
4.2.2. Diagrama de arquitectura.


Servidor Aplicaciones: Tomcat
Libreria base
Usuario
Búsqueda de rutas
Servlet de entrada
(RouteCalculator)
Búsquedas y generación de rutas
Calculo de rutas
(TrackMixer)
Crea
Graf de intersecciones
(RouteGraf)
lee
calculo
Cálculo camino minimo en graf
(PathSearcher)
lee
Descarga de rutas
Base de conocimiento
(RouteDAO)
lee
Repositorio de rutas
Gmaps API
Google maps

4.2.3. Diagrama de clases principal



Debido a la gran cantidad de clases que componen la aplicación, solo se están comentando las clases principales. El formato escogido es el formato de JavaDoc, por ser estándar para este lenguaje y generable automáticamente a partir de los comentarios del código fuente.



trackmixer.web

Class RouteCalculator

```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
          └trackmixer.web.RouteCalculator
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

```
public class RouteCalculator
extends javax.servlet.http.HttpServlet
```

Punto de entrada web de la aplicación de cálculo de rutas. Este servlet proporciona las siguientes funcionalidades 1) Consulta de la base de conocimiento de rutas y descarga de éstas 2) Calculo de rutas dados varios puntos 3) Consulta de las rutas cercanas aun punto 4) Descarga de rutas (existentes y generadas) 5) Ejecución del proceson de refresco de la base de conocimiento a patición del cliente 6) Generación del código script necesario para mostrar la interfaz de usuario mediante la API de google Maps.

See Also:

Serialized Form

Field Summary

static int[]	ALLOWEDCENTERRATIOS radios permitidos en la operación de cálculo de rutas que esten a menor distancia de un punto
static int[]	**ALLOWEDRATIOS** Radios permitidos en el cálculo de rutas
static java.lang.String	**LASTFORM**
static java.lang.String[]	**POINTSXROUTEICONS** colores de los iconos de puntos de ruta

Constructor Summary

RouteCalculator()

Method Summary

static java.lang.String	**addRouteToGMap**(trackmixer.Route r, java.lang.String color, java.lang.String size, int id) Método que genera el código Google Maps que visualiza una ruta, se llama una vez por cada ruta que se está visualizando en el mapa.

void	**doGet**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Punto de entrada del servlet, recibe las invocaciones http del cliente.
static trackmixer.RouteDAO	**getDAO**() Método que da acceso a la base de datos de conocimiento de rutas
void	**init**() Metodo de inicialización del servidor, este método lanza la carga inicial de servidor de ruta, de esta manera al iniciarse por primera vez el servlet se realiza el proceso inicial de carga del Grafo de intersecciones y de base de conocimiento
static java.lang.String	**printMap**(TrackmixerForm form, javax.servlet.http.HttpServletRequest request) Genera el código javascrit del mapa a visualizar.

Methods inherited from class javax.servlet.http.HttpServlet
doDelete, doHead, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet
destroy, getInitParameter, getInitParameterNames, getServletConfig,

`getServletContext, getServletInfo, getServletName, init, log, log`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

ALLOWEDRATIOS

`public static int[] ALLOWEDRATIOS`

Radios permitidos en el cálculo de rutas

ALLOWEDCENTERRATIOS

`public static int[] ALLOWEDCENTERRATIOS`

radios permitidos en la operación de cálculo de rutas que esten a menor distancia de un punto

---

LASTFORM

`public static java.lang.String LASTFORM`

POINTSXROUTEICONS

`public static java.lang.String[] POINTSXROUTEICONS`

colores de los iconos de puntos de ruta

Constructor Detail

RouteCalculator

```
public RouteCalculator()
```

Method Detail

init

```
public void init()
          throws javax.servlet.ServletException
```

Metodo de inicialización del servidor, este método lanza la carga inicial de servidor de ruta, de esta manera al iniciarse por primera vez el servlet se realiza el proceso inicial de carga del Grafo de intersecciones y de base de conocimiento

Overrides:

`init` in class `javax.servlet.GenericServlet`

Throws:

`javax.servlet.ServletException` - Error en caso que algo haya ido mal

---

getDAO

```
public static trackmixer.RouteDAO getDAO()
```

Método que da acceso a la base de datos de conocimiento de rutas

Returns:

Objeto que da acceso a la base de datos de conocmiento

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

Punto de entrada del servlet, recibe las invocaciones http del cliente. Las invocaciones recibidas desde el cliente pueden ser las siguientes: 1) Obtener las rutas que pasan cerca de un punto dado: para ello invocará al método getRoutesInCircle del objeto servidor 2)Refresco del servidor; hará que el servidor recalcule el grafo en función de si hay rutas nuevas en su base de conocimiento 3) Descarga; descargará una de las rutas de la base de conocimiento, o descargará la ruta calculada Este método delega la lógica de presentación a una jsp (main.jsp), la cual genera la página web que corresponde a la interfaz

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - datos de entrada en formato http request

`response` - datos de respuesta al cliente

Throws:

`javax.servlet.ServletException`

`java.io.IOException`

---

addRouteToGMap

```
public static java.lang.String addRouteToGMap(trackmixer.Route r,
                                              java.lang.String color,
                                              java.lang.String size,
                                              int id)
```

Método que genera el código Google Maps que visualiza una ruta, se llama una vez por cada ruta que se está visualizando en el mapa. Este método genera la ruta interpolandola a un máximo de puntos definidos en la variable (MAXGMAPROUTEPOINTS)

Parameters:

`r` - Ruta para la cual se generará el código GMaps

`color` - color de la ruta

`size` - parametro de polilyne GMaps

`id` - identificador del objeto Gmaps que se generará

Returns:

código del polilyne GMaps que visualiza la ruta

---

printMap

```
public static java.lang.String printMap(TrackmixerForm form,
javax.servlet.http.HttpServletRequest request)
```

Genera el código javascrit del mapa a visualizar.

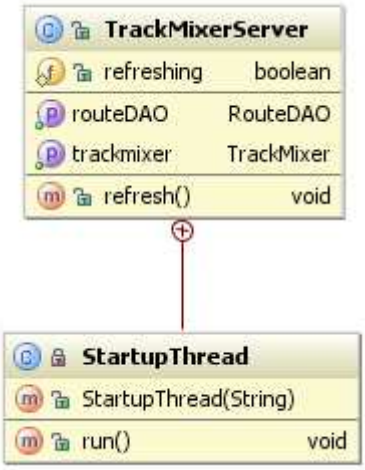
Parameters:

`form` - Formulario que contiene los datos que se quieren visualizar en el mapa

`request` - petición http del cliente

Returns:

código javascript que contiene el mapa a visualizar con todos los datos



trackmixer.web

Class TrackMixerServer

```
java.lang.Object
  └ trackmixer.web.TrackMixerServer
```

```
public class TrackMixerServer
extends java.lang.Object
```

Esta clase aglutina las funcionalidades de un servidor de rutas y de cálculo de estas Implementa el patrón Singleton ya que se quiere una única instancia para todows los clientes Este servidor principalmente mantiene los 2 objetos clave dentro del cálculo de rutas: DAO: objeto que contiene la base de conocimiento de rutas Trackmixer: Motor de cálculo de rutas A parte de mantener estos objetos, el servidor posibilita la operación de refresco de éstos, actualizando sus valores a partir de los cambios que se hayan realizado en el repositorio de rutas. La operación de refresco se lanza en un thread, esto permite asincronía en la operación, la cual puede tardar varios minutos

---

Field Summary

`static boolean`	**`refreshing`** esta variable actúa de indicador de que el servidor se está refrescando, mientras esté a verdadero no deberían lanzarse peticiones contra él.

Method Summary

`static TrackMixerServer`	**`getInstance`**`()` Retorna la instancia del servidor

trackmixer.RouteDAO	**getRouteDAO**() Proporciona el objeto que contiene la base de conocimiento de rutas
trackmixer.TrackMixer	**getTrackmixer**() Proporciona el objeto motor de cálculo de rutas
void	**refresh**() Desencadena los procesos de actualización incremental de la base de conocimiento y del motor de cálculo

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

refreshing

```
public static boolean refreshing
```

esta variable actua de indicador de que el servidor se está refrescando, dmiestras esté a verdadero no deberian lanzarse peticiones contra él.

Method Detail

getInstance

```
public static TrackMixerServer getInstance()
```

Retorna la instancia del servidor

Returns:

servidor

---

refresh

```
public void refresh()
            throws java.lang.Exception
```

Desencadena los procesos de actualización incremental de la base de conocimiento y del motor de cálculo

Throws:

`java.lang.Exception`

---

getRouteDAO

```
public trackmixer.RouteDAO getRouteDAO()
```

Proporciona el objeto que contiene la base de conocimiento de rutas

Returns:

DAO de rutas

---

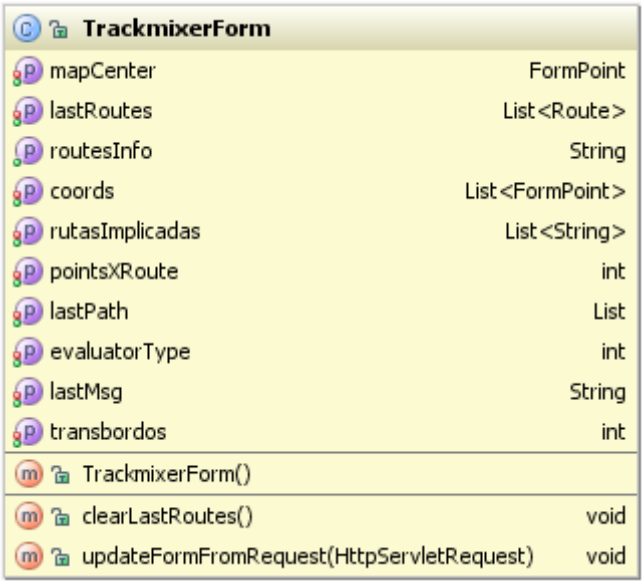
getTrackmixer

```
public trackmixer.TrackMixer getTrackmixer()
```

Proporciona el objeto motor de cálculo de rutas

Returns:

Trackmixer (motor de cálculo)



trackmixer.web

Class TrackmixerForm

```
java.lang.Object
  └ trackmixer.web.TrackmixerForm
```

```
public class TrackmixerForm
extends java.lang.Object
```

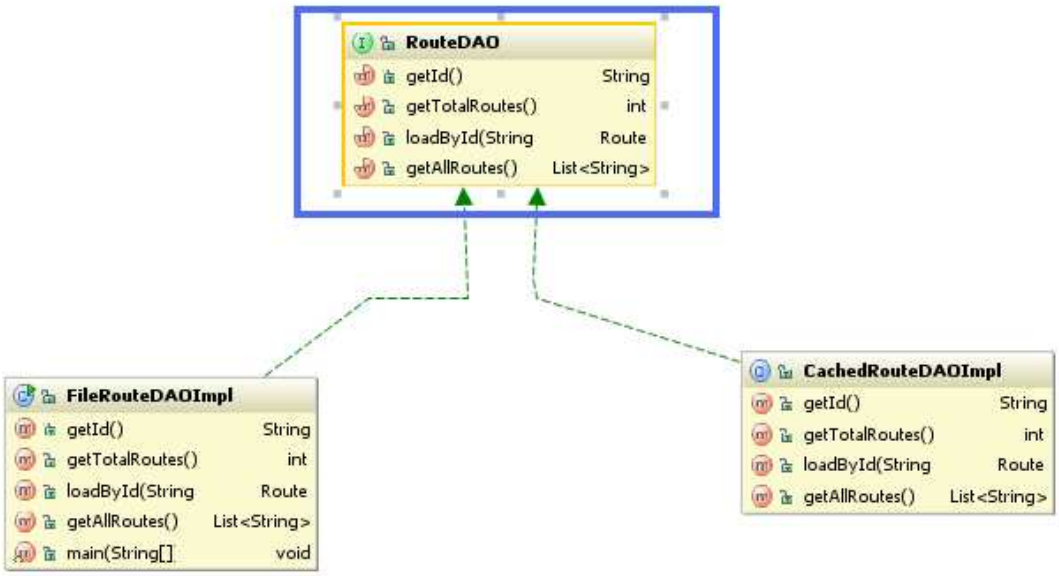
Clase que contiene los datos introducidos en el formulario web, esta clase se mantiene asociado a un cliente web durante toda su sesion de navegación. Los datos contenidos son:

1) Puntos seleccionados para el cálculo de rutas

2) Ruta resultado de la última búsqueda

3) Rutas resultado de la última consulta de rutas

4) Mensaje de respuesta de la última petición



trackmixer

Interface RouteDAO

All Known Implementing Classes:

CachedRouteDAOImpl, FileRouteDAOImpl

`public interface` **`RouteDAO`**

Representa un DAO de acceso a rutas por Identificador. Esta interfaz aglutina los métodos para acceder a un repositorio de rutas, las clases que implementen esta interfaz deberán realizar la lógica de acceso al repositorio de una naturaleza concreta (filesystem, base de datos, caché, capas SOA, etc)

Method Summary

`java.util.List<java.lang.String>`	**`getAllRoutes`**`()` Devuelve una lista con los identificadores de todas las rutas

java.lang.String	**getId**() retorna un identificador unico de este DAO
int	**getTotalRoutes**() Devuelve el numero de total de rutas actual del DAO
Route	**loadById**(java.lang.String routeId) Carga un routa a partir de su Id

Method Detail

getId

`java.lang.String` **`getId`**`()`

retorna un identificador unico de este DAO

Returns:

---

getTotalRoutes

`int` **`getTotalRoutes`**`()`

Devuelve el numero de total de rutas actual del DAO

Returns:

total routes

---

loadById

`Route` **`loadById`**`(java.lang.String routeId)`

Carga un routa a partir de su Id

Parameters:

`routeId` - identificador de ruta

Returns:

Routa o null si no existe

---

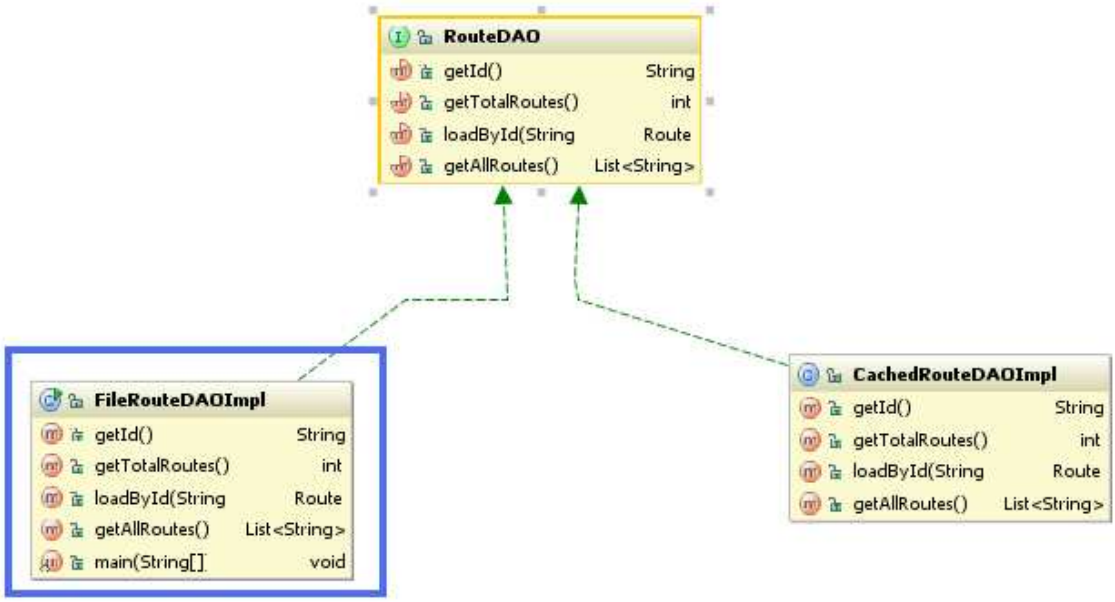
getAllRoutes

`java.util.List<java.lang.String>` **`getAllRoutes`**`()`

Devuelve una lista con los identificadores de todas las rutas

Returns:

lista con los identificadores de ruta



trackmixer

Class FileRouteDAOImpl

`java.lang.Object`
  `└` **`trackmixer.FileRouteDAOImpl`**

All Implemented Interfaces:

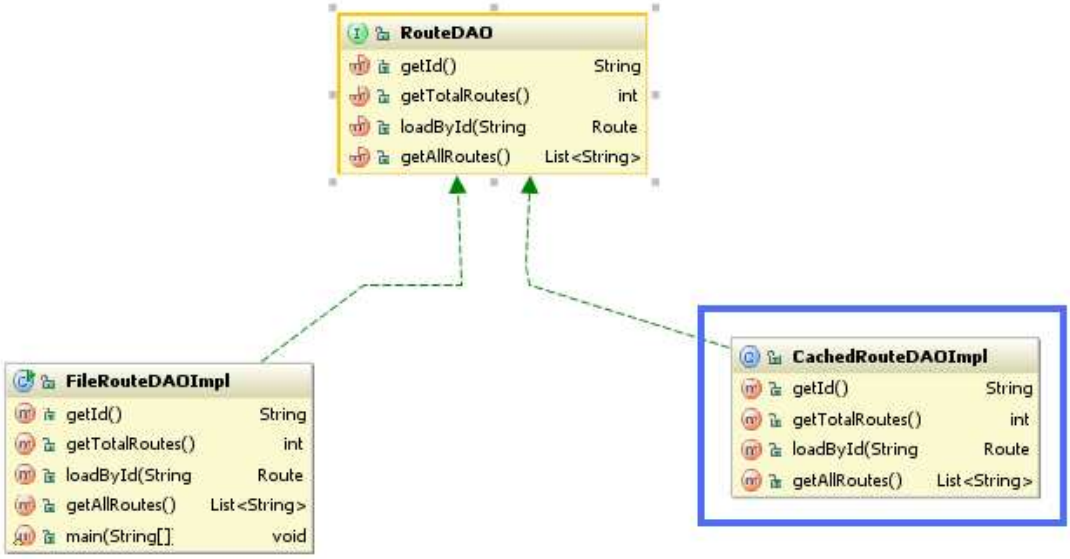
RouteDAO

```
public class FileRouteDAOImpl
extends java.lang.Object
implements RouteDAO
```

Implementa el DAO basándose en un directorio donde existen rutas en formato GPX. Esta clase tiene el siguiente funcionamiento:

1) Carga las rutas en formato GPX y las convierte en formato Route (java)

2) Serializa el objeto Route a disco, de esta manera los siguientes accesos se harán a partir de esa ruta serializada evitándonos así el tiempo de interpretación del GPX y de carga de los objetos de geometría de la ruta



trackmixer

Class CachedRouteDAOImpl

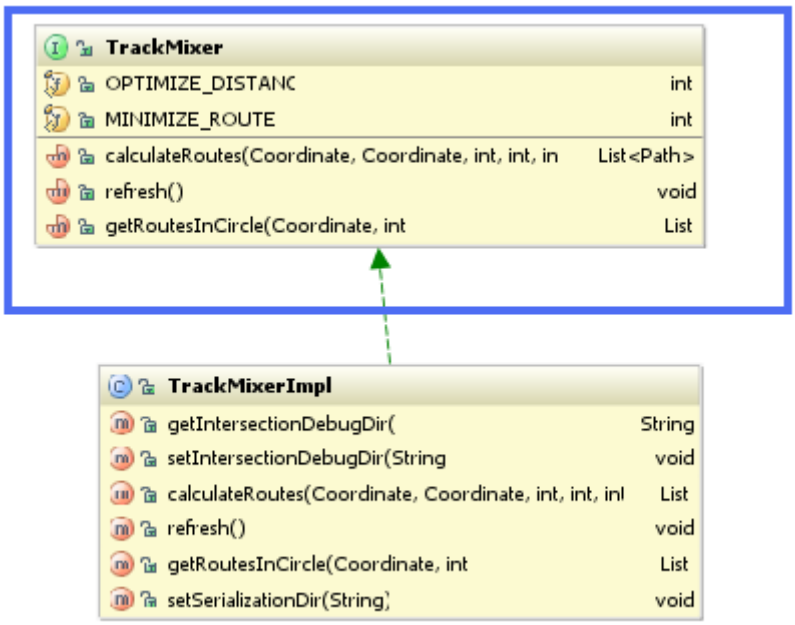
```
java.lang.Object
  └ trackmixer.CachedRouteDAOImpl
```

All Implemented Interfaces:

RouteDAO

```
public class CachedRouteDAOImpl
extends java.lang.Object
implements RouteDAO
```

Implementa una capa de cache de rutas sobre un DAO Esta clase constituye una capa proxy de cache entre un DAO de carga de rutas, lo cual acelera notablemente el acceso a los métodos del DAO ya que las rutas se encuentran en una cache en memoria Para la implementación de la cache utiliza la librería ehCache. La configuración de la caché a nivel de destino físico donde se colocan los objetos, política de evicción y cantidad de elementos en cache se encuentra en el fichero de propiedades estándar de la librería ehcache.



trackmixer

Interface TrackMixer

All Known Implementing Classes:

TrackMixerImpl

`public interface` **`TrackMixer`**

Realiza calculo de rutas entre 2 puntos, parte de una base de conocimiento de rutas (RouteDAO) La base de conocimiento de rutas se encuentra en una variable ThreadLocal

See Also:

`RouteDAOLocator`

Field Summary

`static int`	**`MINIMIZE_ROUTES`** identificador de parametro que indica que se debe aplicar la optimización de menos transbordos entre puntos en el cálculo de rutas
`static int`	**`OPTIMIZE_DISTANCE`** identificador de parametro que indica que se debe aplicar la optimización de distancia entre puntos en el cálculo de rutas

Method Summary

`java.util.List<Path>`	**`calculateRoutes`**`(Coordinate start, Coordinate end, int startRadix, int endRadix, int evaluatorType)`

	Calcula rutas entre 2 coordenadas
java.util.List	**getRoutesInCircle**(Coordinate start, int radix) Nos devuelve las rutas que pasan por un circulo especificado con coordenada y radio
void	**refresh**() Da la orden refrescar la clase a partir del DAO actual (add de rutas)

Field Detail

OPTIMIZE_DISTANCE

```
static final int OPTIMIZE_DISTANCE
```

Identificador de parámetro que indica que se debe aplicar la optimización de distancia entre puntos en el cálculo de rutas

See Also:

Constant Field Values

MINIMIZE_ROUTES

```
static final int MINIMIZE_ROUTES
```

Identificador de parámetro que indica que se debe aplicar la optimización de menos transbordos entre puntos en el cálculo de rutas

See Also:

Constant Field Values

Method Detail

calculateRoutes

```
java.util.List<Path> calculateRoutes(Coordinate start,
                                     Coordinate end,
                                     int startRadix,
                                     int endRadix,
                                     int evaluatorType)
```

Calcula rutas entre 2 coordenadas

Parameters:

`start` - coordenada destino

`end` - coordenada inicio

`startRadix` - radio de accion de cooordenada inicio

`endRadix` - radio de accion de cooordenada destino

`evaluatorType` - tipo de evaluador de mejor ruta

Returns:

una lista con las posibles rutas en formato Path

---

refresh

```
void refresh()
            throws java.lang.Exception
```

Da la orden refrescar la clase a partir del DAO actual (add de rutas)

Throws:

`java.lang.Exception`

---

getRoutesInCircle

```
java.util.List getRoutesInCircle(Coordinate start,
                                 int radix)
```

Nos devuelve las rutas que pasan por un circulo especificado con coordenada y radio

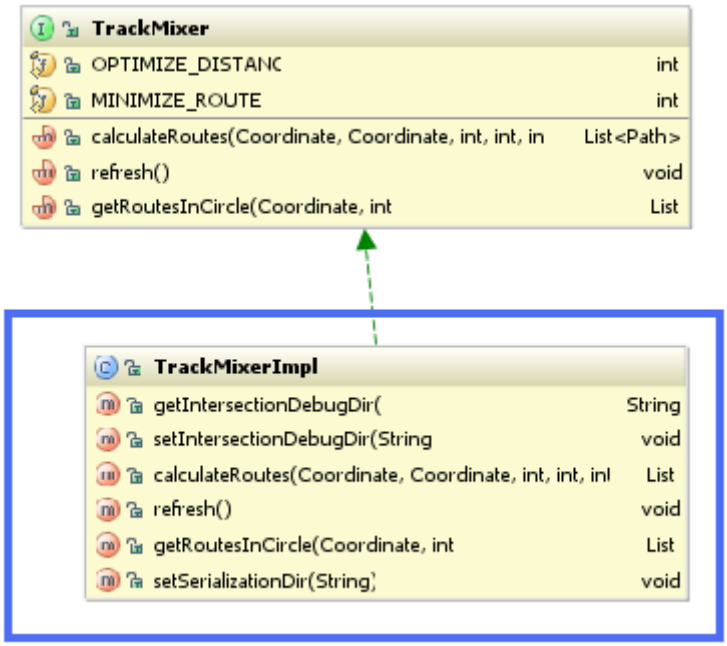
Parameters:

`start` - centro del circulo

`radix` - radio

Returns:

lista de rutas



trackmixer

Class TrackMixerImpl

```
java.lang.Object
  └ trackmixer.TrackMixerImpl
```

All Implemented Interfaces:

TrackMixer

```
public class TrackMixerImpl
extends java.lang.Object
implements TrackMixer
```

Implementación del motor de cálculo de rutas a partir de una base de conocimiento de rutas Esta implementación mantiene un grafo de intersecciones entre rutas los estados del grafo están conectados por transiciones que representan un coste de llegada de punto a punto utilizando una determinada ruta. El coste de cálculo del grafo es alto pero solo se debe recalcular ante modificaciones del DAO, por lo cual una vez creado se serializa

a disco para posteriormente cargarlo La clase es thread-safe y es conveniente que se utilice a modo singleton.

Field Summary

Fields inherited from interface trackmixer.TrackMixer
MINIMIZE_ROUTES, OPTIMIZE_DISTANCE

Constructor Summary

TrackMixerImpl()

Method Summary

java.util.List	**calculateRoutes**(Coordinate start, Coordinate end, int startRadix, int endRadix, int evaluatorType) Calcula rutas entre 2 coordenadas
java.lang.String	**getIntersectionDebugDir**()
java.util.List	**getRoutesInCircle**(Coordinate start, int radix) Nos devuelve las rutas que pasan por un circulo especificado

	con coordenada y radio
void	**refresh**() sincroniza datos contra el DAO
void	**setIntersectionDebugDir**(java.lang.String intersectionDebugDir) Fija un directorio donde se colocaran las intersecciones entre rutas en formato GPX Si no se informa no se almacena nada.
void	**setSerializationDir**(java.lang.String serDir) Fija el directorio de serializacion donde se almacena el grafo despues de calcularlo

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TrackMixerImpl

```
public TrackMixerImpl()
```

Method Detail

getIntersectionDebugDir

```
public java.lang.String getIntersectionDebugDir()
```

---

setIntersectionDebugDir

```
public void
setIntersectionDebugDir(java.lang.String intersectionDebugDir)
```

Fija un directorio donde se colocaran las intersecciones entre rutas en formato GPX Si no se informa no se almacena nada.

Parameters:

`intersectionDebugDir` -

---

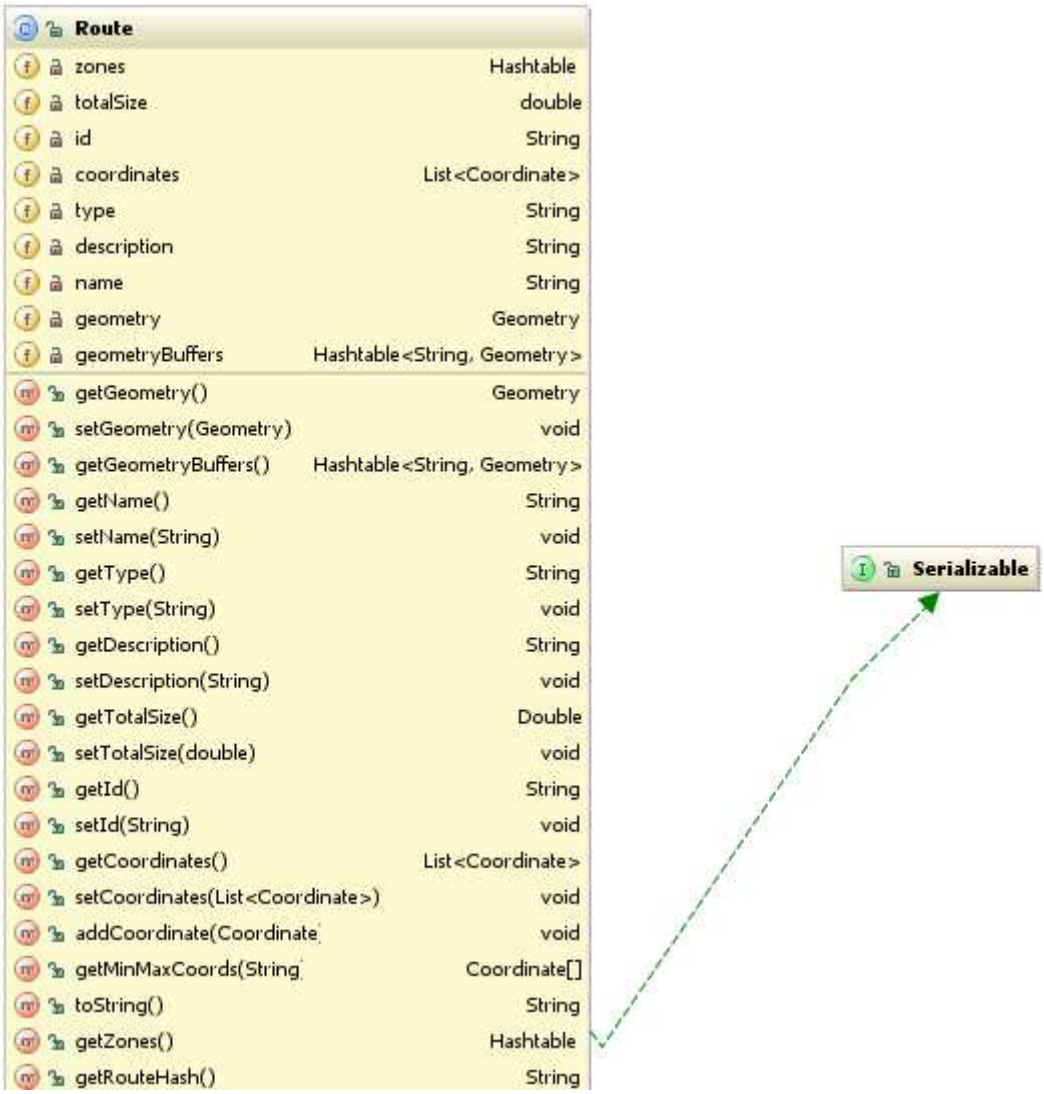
setSerializationDir

```
public void setSerializationDir(java.lang.String serDir)
```

Fija el directorio de serializacion donde se almacena el grafo despues de calcularlo

Parameters:

`serDir` - directorio de serializacion



trackmixer

Class Route

```
java.lang.Object
  └ trackmixer.Route
```

All Implemented Interfaces:

java.io.Serializable

```
public class Route
extends java.lang.Object
implements java.io.Serializable
```

Representa una ruta. Una ruta es un conjunto ordenado de una o más coordenadas geográficas. Principalmente contiene esta información:

1)Nombre de la ruta

2)Tipo de la ruta: el tipo es un identificador libre, de alguna manera permite agrupar rutas; por ejemplo rutas hechas con BTT, rutas hechas a pie, en coche, etc.

3)Información de la geometría: esta información es importante ya que posibilita utilizar operaciones geométricas con esta ruta (intersecciones, distancias, etc.)

4) Tamaño total de la ruta en metros

5) Descripción de la ruta

6) Identificador único de la ruta

See Also:

Serialized Form

Constructor Summary

Route()
Route(Route base)

Method Summary

void	**addCoordinate**(Coordinate coord) Añade una coordenada a la ruta
java.util.List<Coordinate>	**getCoordinates**()

java.lang.String	**getDescription**()
com.vividsolutions.jts.geom.Geometry	**getGeometry**() devuelve una representacion geometrica de la ruta, es util para operaciones geometricas
java.util.Hashtable<java.lang.String,com.vividsolutions.jts.geom.Geometry>	**getGeometryBuffers**()
java.lang.String	**getId**()
Coordinate[]	**getMinMaxCoords**(java.lang.String zone) Retorna la máxima y minima coordenada que tiene la ruta en una zona UTM
java.lang.String	**getName**()
java.lang.String	**getRouteHash**() Nos da el código hash de la ruta, es util para delimitar rutas duplicadas
java.lang.Double	**getTotalSize**() nos devuelve el tamanyo en metros de la ruita
java.lang.String	**getType**()
java.util.Hashtable	**getZones**() Nos devuelve las zonas UTM por las que pasa la ruta

void	**setCoordinates**(java.util.List<Coordinate> coordinates)
void	**setDescription**(java.lang.String description)
void	**setGeometry**(com.vividsolutions.jts.geom.Geometry geometry)
void	**setId**(java.lang.String id)
void	**setName**(java.lang.String name)
void	**setTotalSize**(double totalSize)
void	**setType**(java.lang.String type)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Route

```
public Route()
```

Route

```
public Route(Route base)
```

Method Detail

getGeometry

```
public com.vividsolutions.jts.geom.Geometry getGeometry()
```

devuelve una representacion geometrica de la ruta, es util para operaciones geometricas

Returns:

geometria de la ruta

---

setGeometry

```
public void setGeometry(com.vividsolutions.jts.geom.Geometry geometry)
```

---

getGeometryBuffers

```
public
java.util.Hashtable<java.lang.String,com.vividsolutions.jts.geom.Geometry> getGeometryBuffers()
```

---

getName

```
public java.lang.String getName()
```

---

setName

```
public void setName(java.lang.String name)
```

---

getType

```
public java.lang.String getType()
```

---

setType

```
public void setType(java.lang.String type)
```

---

getDescription

```
public java.lang.String getDescription()
```

---

setDescription

```
public void setDescription(java.lang.String description)
```

---

getTotalSize

```
public java.lang.Double getTotalSize()
```

nos devuelve el tamanyo en metros de la ruita

Returns:

---

setTotalSize

```
public void setTotalSize(double totalSize)
```

---

getId

```
public java.lang.String getId()
```

---

setId

```
public void setId(java.lang.String id)
```

---

getCoordinates

```
public java.util.List<Coordinate> getCoordinates()
```

---

setCoordinates

```
public void setCoordinates(java.util.List<Coordinate> coordinates)
```

---

addCoordinate

```
public void addCoordinate(Coordinate coord)
```

Añade una coordenada a la ruta

Parameters:

`coord` -

---

getMinMaxCoords

```
public Coordinate[] getMinMaxCoords(java.lang.String zone)
```

Retorna la máxima y minima coordenada que tiene la ruta en una zona UTM

Parameters:

`zone` -

Returns:

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

---

getZones

```
public java.util.Hashtable getZones()
```

Nos devuelve las zonas UTM por las que pasa la ruta

Returns:

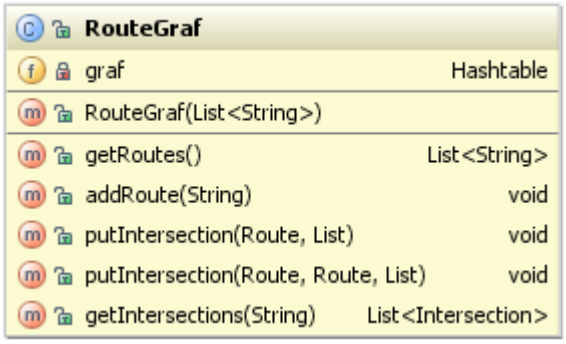
---

getRouteHash

```
public java.lang.String getRouteHash()
```

Nos da el código hash de la ruta, es util para delimitar rutas duplicadas

Returns:



trackmixer

Class RouteGraf

```
java.lang.Object
  └ trackmixer.RouteGraf
```

All Implemented Interfaces:

java.io.Serializable

```
public class RouteGraf
extends java.lang.Object
implements java.io.Serializable
```

Esta clase representa un grafo de rutas, contiene un conjunto de intersecciones entre rutas unidas entre ellas mediante transiciones y una lista de identificadores de las rutas implicadas en el grafo.

See Also:

Serialized Form

Constructor Summary

RouteGraf(java.util.List<java.lang.String> routes) Crea un graf a partir de una lista de indetificadores de rutas

Method Summary

void	**addRoute**(java.lang.String routeId) Añade una ruta al graf
java.util.List<Intersection>	**getIntersections**(java.lang.String routeId) Consulta las intersecciones en las que está implicada una ruta
java.util.List<java.lang.String>	**getRoutes**() Retorna todas las rutas que estan en el graf
void	**putIntersection**(Route a, java.util.List intersections) Añade la lista de intersecciones en las que está implicada una ruta
void	**putIntersection**(Route a, Route b, java.util.List intersections) Añade la lista de intersecciones entre 2

	rutas

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

RouteGraf

`public RouteGraf(java.util.List<java.lang.String> routes)`

Crea un graf a partir de una lista de indetificadores de rutas

Parameters:

`routes` -

Method Detail

getRoutes

`public java.util.List<java.lang.String> getRoutes()`

Retorna todas las rutas que estan en el graf

Returns:

---

addRoute

`public void addRoute(java.lang.String routeId)`

Añade una ruta al graf

Parameters:

`routeId` -

---

putIntersection

```
public void putIntersection(Route a,
                            java.util.List intersections)
```

Añade la lista de intersecciones en las que está implicada una ruta

Parameters:

`a` - ruta

`intersections` - intersecciones

---

putIntersection

```
public void putIntersection(Route a,
                            Route b,
                            java.util.List intersections)
```

Añade la lista de intersecciones entre 2 rutas

Parameters:

`a` - ruta 1

`b` - ruta 2

`intersections` - interesecciones

---

getIntersections

```
public java.util.List<Intersection>
getIntersections(java.lang.String routeId)
```

Consulta las intersecciones en las que está implicada una ruta

Parameters:

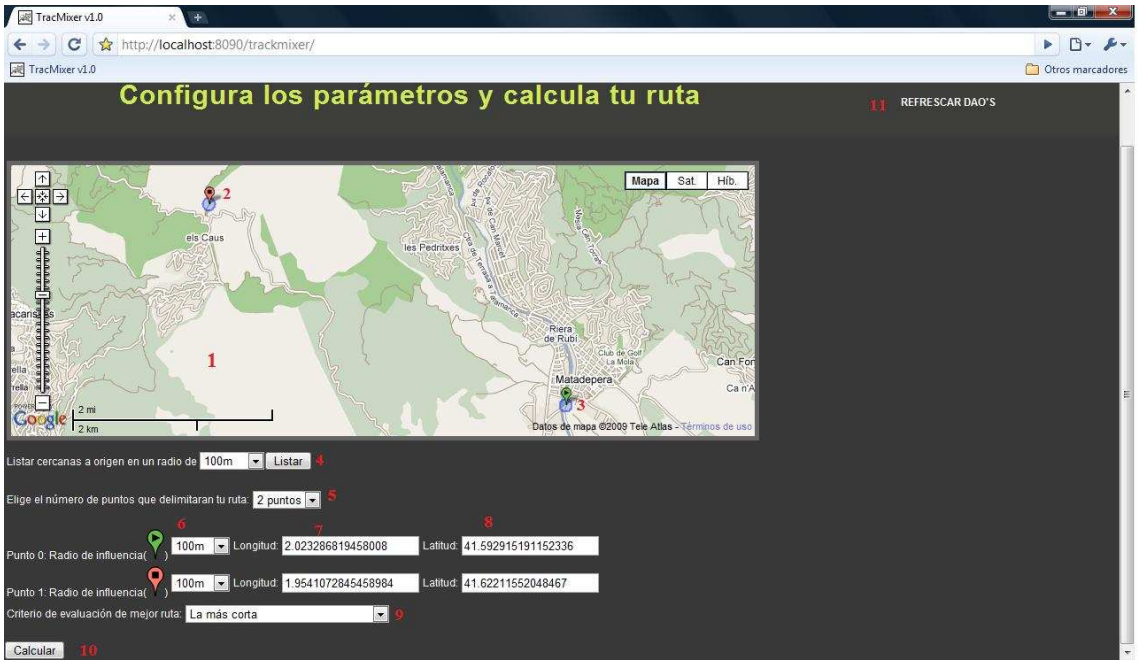
`routeId` - identificador de la ruta

Returns:

lista de intersecciones

4.2.4. Diseño de la interfaz web.

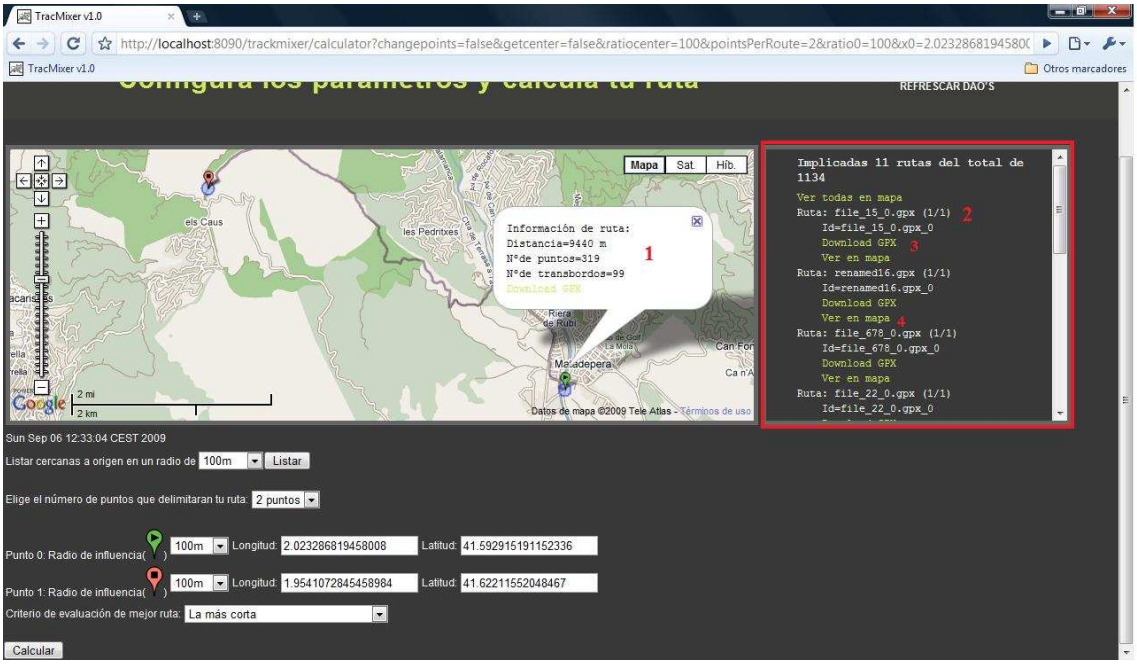
4.2.4.1. Pantalla inicial:



Esta es la pantalla principal de la aplicación. En ella, podemos observar los diferentes controles que la componen y que posibilitan la configuración del cálculo de una ruta de acuerdo con nuestras necesidades:

1. Mapa de Google Maps.
2. Punto final del la ruta a calcular.
3. Punto inicial de la ruta a calcular.
4. Control que nos sirve para calcular la ruta cercana al punto de origen de la ruta dentro de un radio determinado.
5. Control que nos permite seleccionar el numero de que delimitaran nuestra ruta. Es el conjunto de puntos a los cuales nuestra ruta calculada es cercana. (2-6)
6. Radio de influencia de cada uno de los puntos.
7. Longitud del punto seleccionado.
8. Latitud del punto seleccionado.
9. Control que nos permite seleccionar el criterio de calculo de la ruta.
10. Control que inicia el cálculo de la ruta.
11. Control que recalcula el grafo de transiciones entre intersecciones de rutas de la base de conocimiento.

4.2.4.2. Pantalla de resultados:



Pantalla de resultados: Una vez se ha calculado nuestra ruta, se nos muestra la siguiente pantalla, donde veremos los resultados obtenidos. En esta pantalla, los elementos a destacar se encuentran resaltados en rojo.

1. Ruta generada y características de la misma.

Dentro del recuadro:

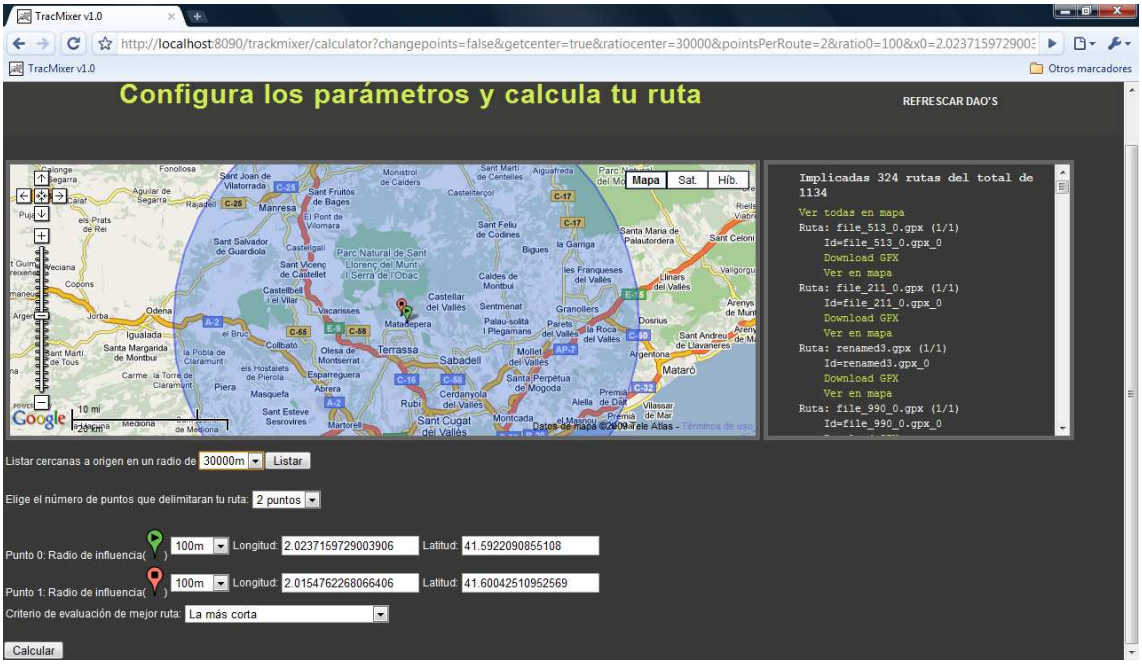
2. Identificador de las rutas que componen la ruta generada.
3. Enlace que nos permite visualizar una ruta concreta de la lista en el mapa de resultados.
4. Enlace para descargarnos la ruta a disco.

5. Pruebas.

Para la realización del proyecto se han efectuado una serie de pruebas unitarias y de integración para asegurar el correcto funcionamiento de la aplicación.

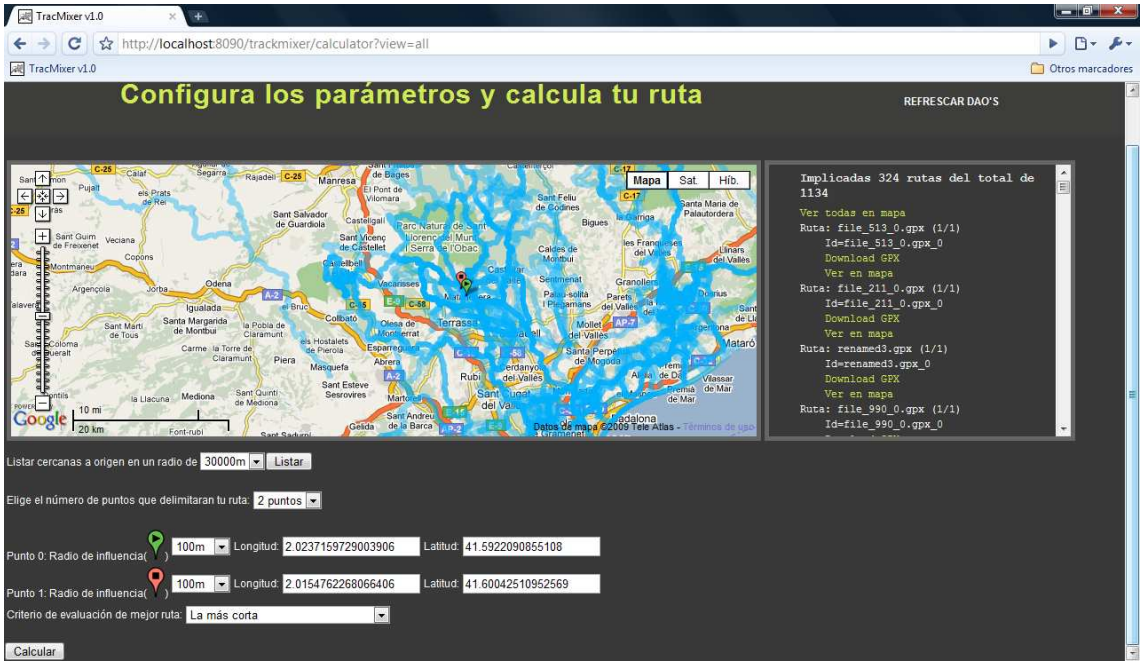
Estas pruebas se realizaron definiendo diferentes bases de conocimiento de rutas en función de la elección del radio a explorar, para asi tener menos rutas iimplicadas, y poder analizar con más detenimiento los resultados.

5.1. Rutas en una distancia definida a un punto.



De este modo, podemos observar que las rutas que obtenemos dentro de ese radio en concreto se elevan a 324. Las rutas implicadas en la búsqueda pueden visualizarse todas o una a una.

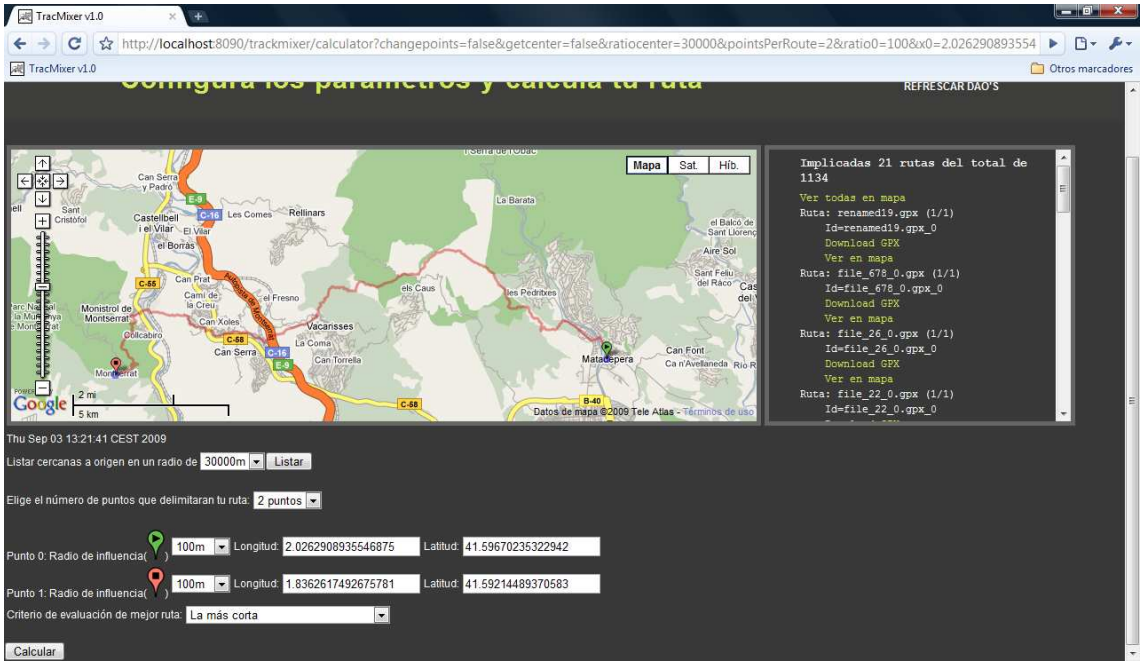
Mostramos todas en el mapa:



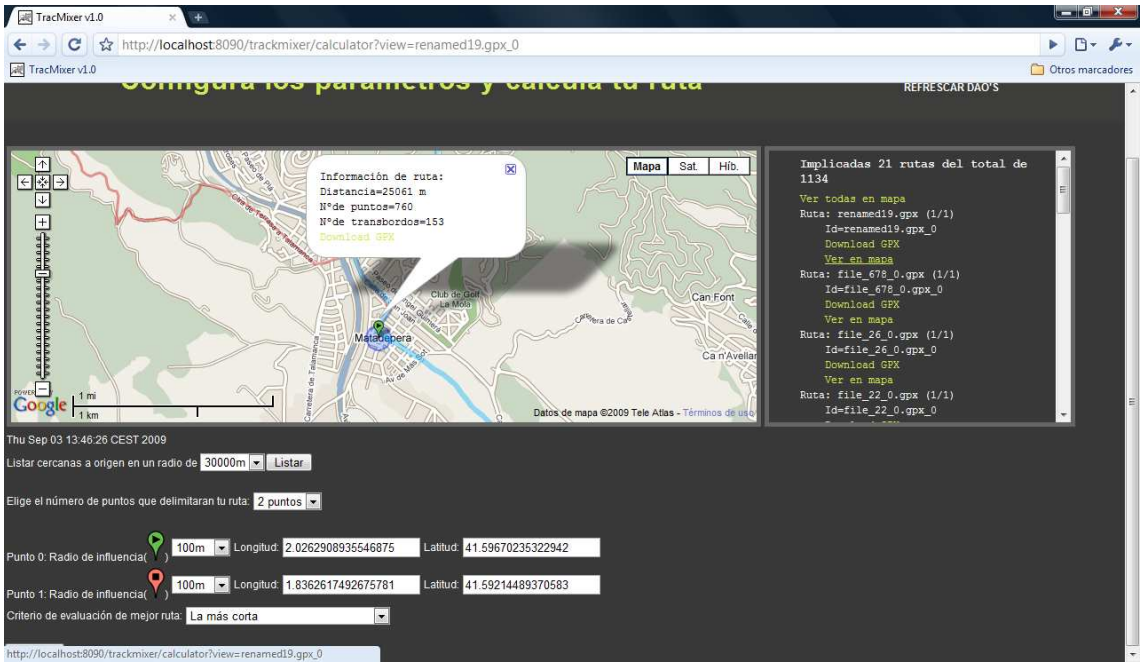
5.2. Calculo de una ruta sencilla definida por dos puntos.

Para calcular una ruta utilizando como base de conocimiento estas 324 rutas, lo que se hace es definir una búsqueda dentro del radio anteriormente utilizado, así será más fácil visualizar las rutas que componen el cálculo.

Por ejemplo, configuramos el cálculo para una ruta que este definida por solamente dos puntos, con radios de influencia de 100 metros por punto, partiendo desde el punto inicial que utilizamos en el primer ejemplo, y como final seleccionamos una posición que se encuentre dentro del radio de 30 kilómetros.

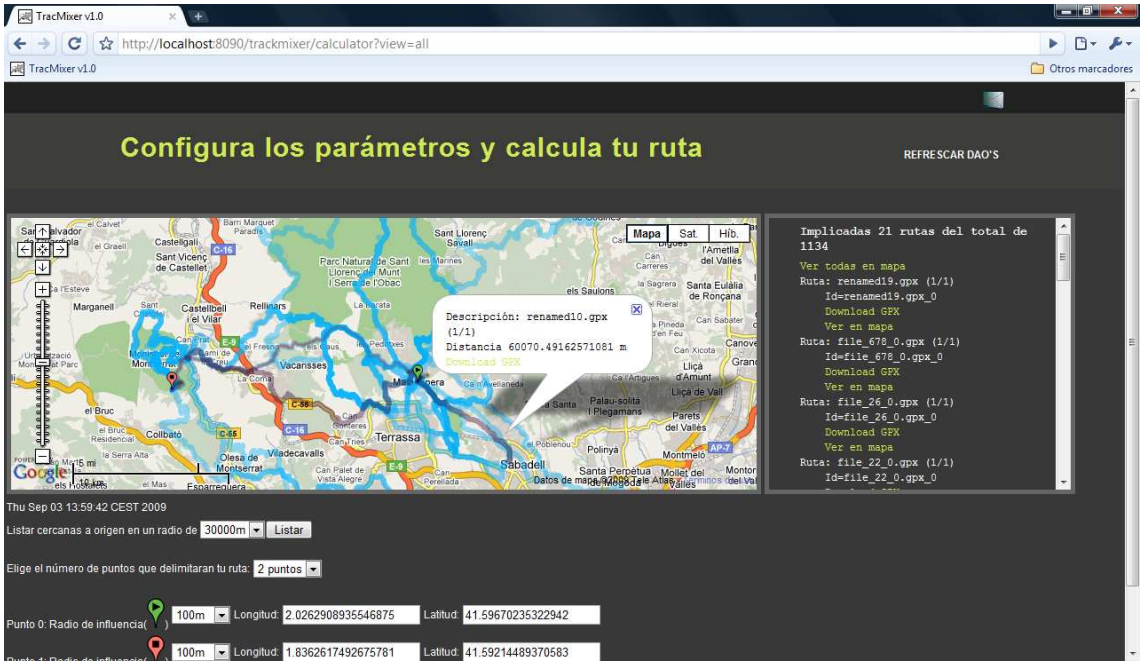


Vemos que calcula una ruta utilizando 21 rutas de la base de conocimiento. Una vez tenemos la ruta calculada, podemos comprobar mostrando las rutas implicadas una a una que, efectivamente, éstas forman parte de la ruta calculada, seleccionando “Ver en mapa” en cada una de las rutas implicadas.



La ruta calculada se representa en color rojo, mientras que la ruta implicada seleccionada de la lista de la izquierda se representa en color azul. También observamos los detalles de la ruta que hemos generado si pulsamos sobre ella.

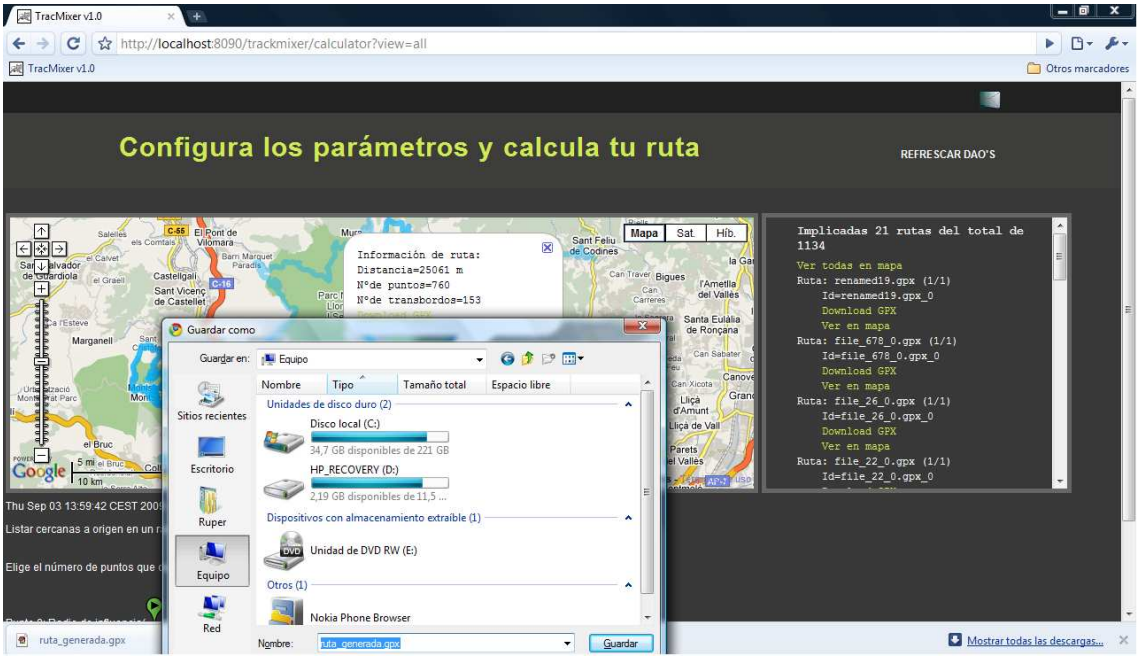
Si las mostramos todas, podremos ver el recorrido resaltado de cada una de las rutas que la componen pasando el mouse sobre ellas, y al pulsar, obtendremos información de la misma, y como siempre, tendremos la opción de descargarnos la ruta.



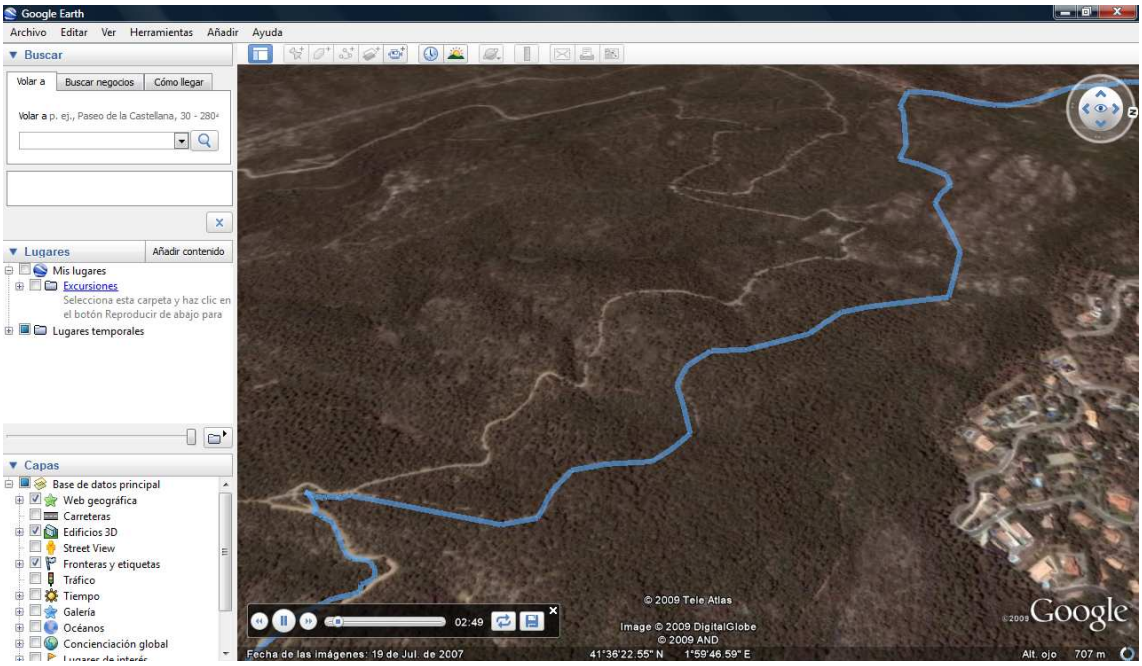
5.3. Descarga de una ruta y visualización en Google Earth.

Una vez tenemos nuestra ruta calculada, podemos descargarla en un fichero en formato GPX y visualizarla en un aplicativo externo, como por ejemplo, Google Maps.

Primero descargamos el fichero:

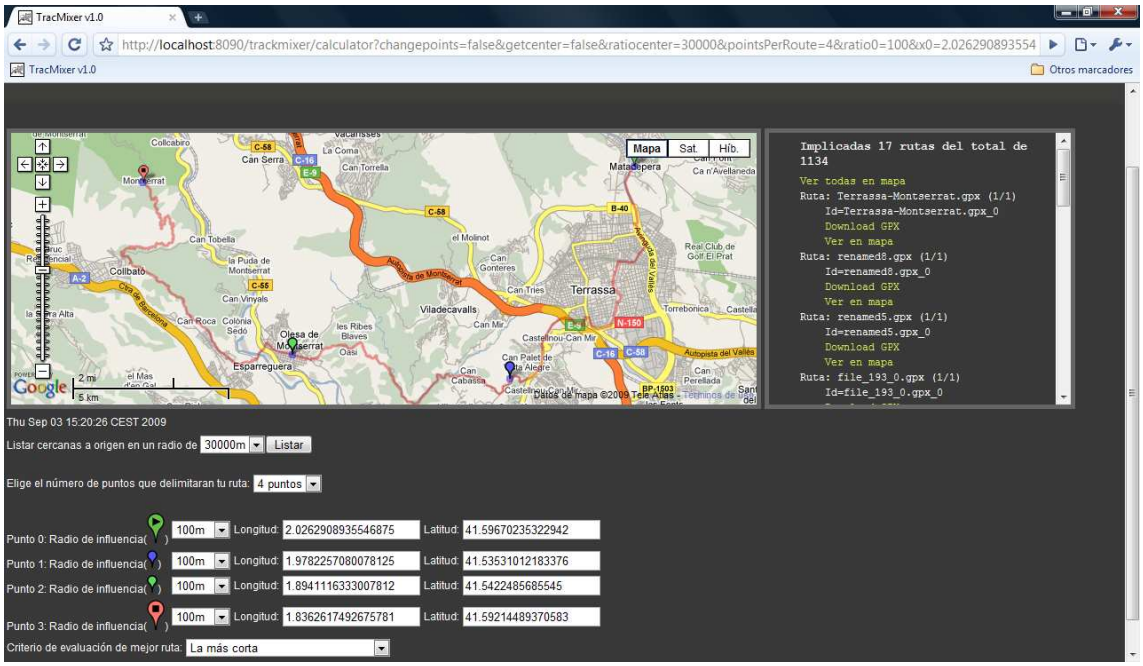


Y una vez lo tenemos descargado, podemos abrirlo con Google Maps para visualizar la ruta generada en esta aplicación.



5.4. Cálculo de ruta compleja definida por cuatro puntos.

En este ejemplo vamos a calcular una ruta que pase por cuatro puntos:

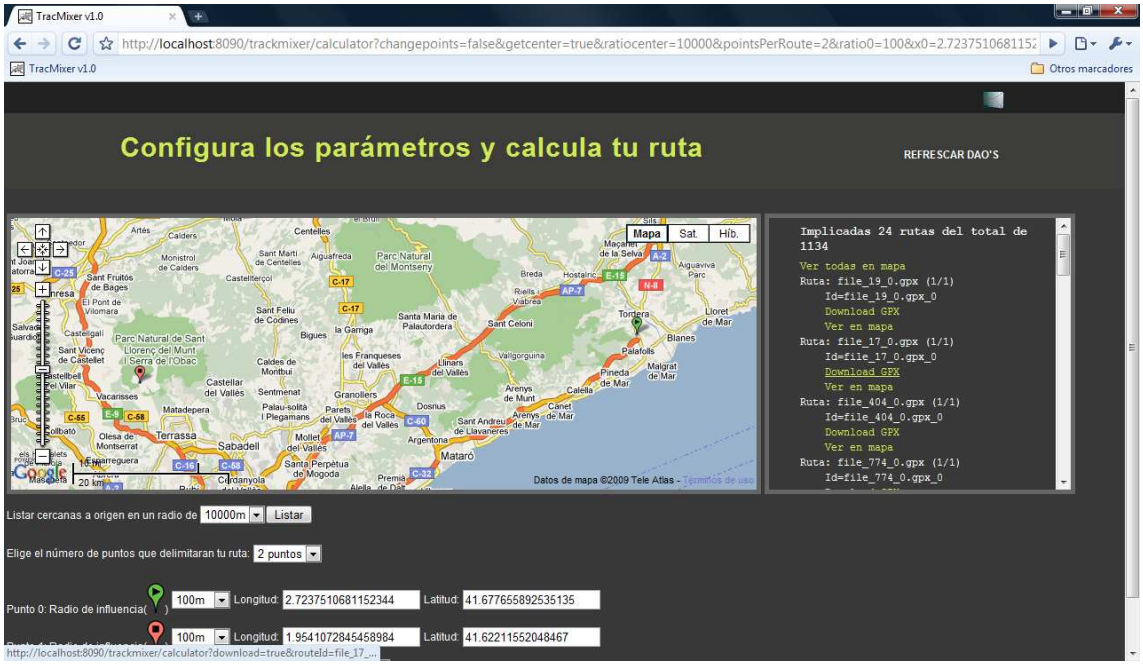


Vemos que de toda la base de conocimiento, tenemos solamente implicadas 17 rutas, y como siempre, podemos verlas todas en el mapa, una a una, o descargarlas a disco en GPX.

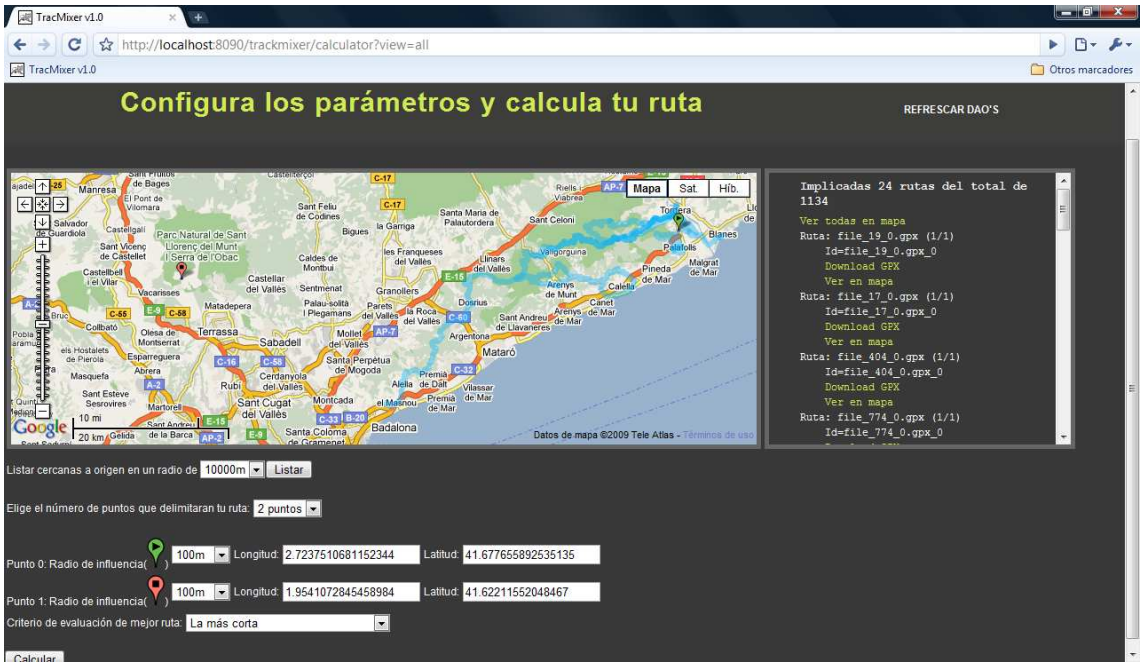
5.5. Caso de ruta no encontrada.

Para mostrar este ejemplo nos hemos de posicionar en un punto en el que tengamos poca densidad de rutas. Esto provocara que aumente la posibilidad de no encontrar rutas con los parámetros que seleccionemos ya que las rutas implicadas en el calculo son menores.

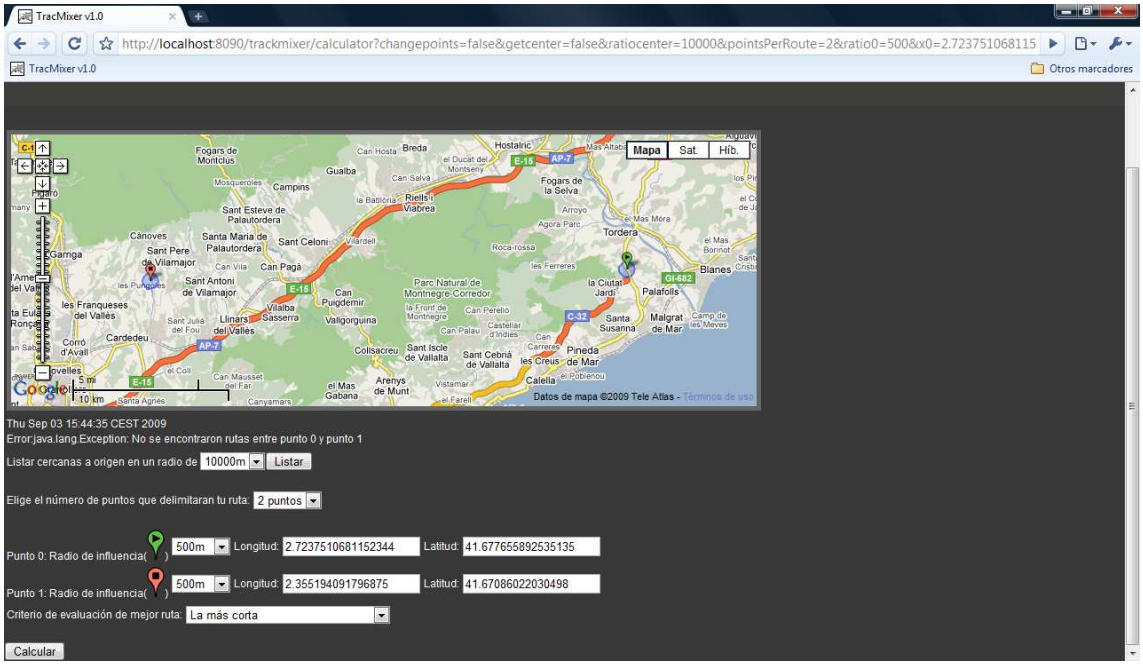
Comenzamos listando las rutas cercanas a un radio de 10 kilómetros de un punto en concreto:



Ahora, mostramos todas las rutas en la zona, para ver que, efectivamente, hay muy poca densidad, y ayudarnos en el posicionamiento del segundo punto para provocar el error:



Podemos ver que si calculamos una ruta entre dos puntos sobre esta zona, nos produce el siguiente resultado, ya que no ha sido capaz de encontrar una ruta que una los puntos origen y destino con los parámetros establecidos.

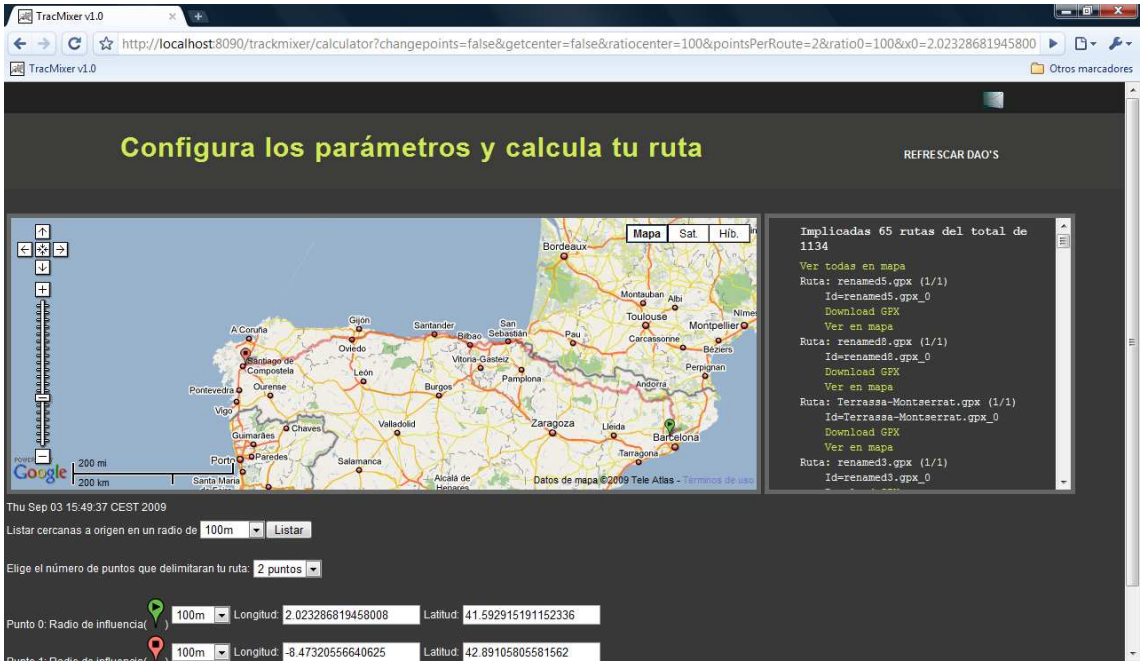


5. 6. Ejemplo de ruta que pasa por distintas zonas UTM

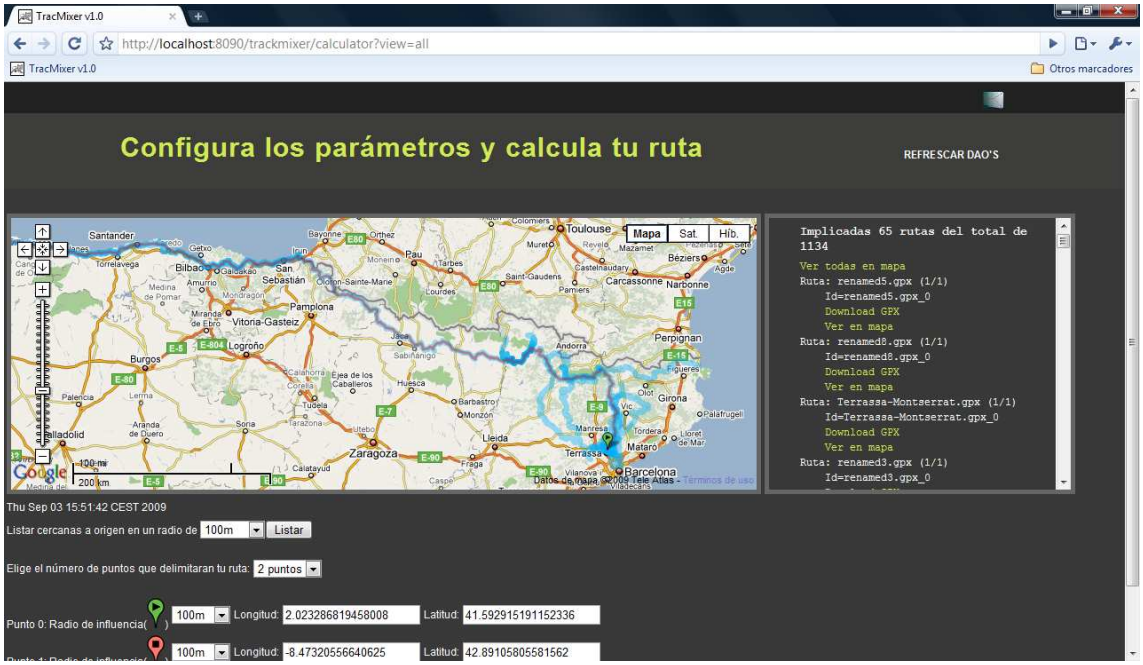
En este ejemplo trataremos el caso de una ruta que comprenda dos puntos que están separados por una distancia considerablemente grande, y que además, están separados incluso de zona UTM, ya que a cada punto le corresponde una zona diferente.

Buscaremos, por ejemplo, una ruta entre Santiago de Compostela (Huso 29) y Matadepera (Huso 31)

Seleccionamos los parámetros y calculamos:



Vemos que, según se muestra en el listado de la derecha, para confeccionar la ruta calculada han sido necesarias la utilización de segmentos de ruta de 65 rutas diferentes de las que componen nuestra base de conocimiento. Las mostramos también en el mapa:



6. Conclusiones:

Para concluir, podemos decir que se ha proporcionado una herramienta que añade unas capacidades que muchos sistemas GIS de estas características no contemplan, y que son, la capacidad de generar rutas en función de su base de conocimiento y la posibilidad de ver rutas en el entorno de un punto de origen.

6.1. Objetivos cubiertos.

Los objetivos planteados en un principio se han cubierto. Se ha desarrollado un aplicativo independiente de la plataforma que posibilita la creación de nuevas rutas a partir de unas que ya se tienen, y se proporciona un entorno visual mediante el cual poder manipular el sistema, mediante la conjunción de html y el api de Google Maps. Se proporciona la funcionalidad de generar estas rutas basándonos en unos criterios determinados, y, además, podemos descargárnoslas en un fichero con un formato estándar reconocible por la mayoría de los dispositivos GPS y aplicaciones, como por ejemplo Google Earth, cosa que posibilita la visualización del resultado de múltiples maneras.

TrackMixer es una herramienta muy fácil de manejar, cumple una función específicamente definida, y está pensada en última instancia para ser usada en un futuro como un módulo a añadir en otros sistemas GIS, añadiéndole una nueva capacidad. Su diseño ha sido pensado de esa forma, pudiéndose adaptar a cualquier tipo de entorno grafico y a cualquier fuente de datos

6.2. Líneas de continuidad.

TrackMixer es un aplicativo totalmente abierto. Al haberse gestado en un principio como un módulo en vistas a añadirlo a otro sistema, su diseño permite añadir múltiples mejoras manteniendo todas las funcionalidades ya existentes, y algunas en las que se tiene previsto dedicar más esfuerzos son nuevos criterios de búsqueda de rutas, la inclusión de un conversar de formatos de rutas para posibilitar la descarga de rutas en formatos más específicos de dispositivos concretos, e incluir en la ruta generada referencias a diferentes puntos de interés existentes. También se está estudiando el uso

de alguna base de datos espacial, que contribuirá a una mejora considerable en el tiempo de cálculo del grafo de intersecciones. Otra de las cosas en las que se hará hincapié en un futuro es utilizar la información referente a las alturas en las rutas contenidas en la base de conocimiento para poder ver en la ruta calculada información con respecto al desnivel de la misma.

7. Bibliografía.

- BASART, J. M.: Grafs: fonaments i algorismes. Manuals de la UAB, 13. 1994. ISBN 84-7929-982-7.
- de Berg, Mark. Cheong. Otfried, van Kreveld, Marc. Overmars, Mark. Computational Geometry. Algorithms and applications. Thrid Edition. ISBN 978-3-540-77973-5
- [PRE01] Pressman, Roger S. "Ingeniería de Software. *Un enfoque práctico."* 5ta Edición (traducción de la edición original en Inglés *"Software Engineering. A practical approach"*), Editorial Mac Graw Hill, Madrid, España, 2001.
- Larman,Craig: UML y Patrones. Introducción al análisis y diseño orientado a objetos. Editorial Prentice Hall.

Recursos Web:

- Google Maps API : http://code.google.com/intl/es/apis/maps/
- Casos de uso: http://ayfisw.files.wordpress.com/2008/08/tema3d1.pdf
- Información general: http://www.wikipedia.org
- Coordenadas UTM: http://maptools.com/UsingUTM/
- Coordenadas UTM: http://www.manifold.net/doc/universal_transverse_mercator_utm_.htm
- Sobre el formato GPX: http://www.topografix.com/GPX/1/1/
- http://www.jstott.me.uk/jcoord/
- Información general: http://www.nosolosig.com

8. Apéndices:

8.1. Instrucciones de instalación.

La aplicación se distribuye en un formato estándar ".war" (Web Application Resource) que se deberá instalar en el servidor siguiendo las instrucciones de carga del fichero war en el servidor web utilizado.

El fichero ".war" se encuentra en \build\release\trackmixer

Para instalarla en un servidor Jacarta Tomcat simplemente se deberá copiar el archivo en el directorio ***$TOMCAT_DIR/webapps*** y de esta manera se desplegará en el directorio.

A continuación será necesario configurar la aplicación
$TOMCAT_DIR/webapps/control/WEB-INF/classes/config.properties

8.2 Fichero de configuración:

Existe un fichero de configuración del aplicativo donde se especifican las variables de configuración dependientes del sistema:

El fichero se encuentra en la ruta \WEB-INF\classes\config.properties.

Dicho fichero contiene la variable de configuración que se encarga de definir la ruta donde está ubicada nuestra base de conocimiento.

9.1 Fichero de configuración:

Existe un fichero de configuración del aplicativo donde se especifican las variables de configuración dependientes del sistema:

El fichero se encuentra en la ruta \WEB-INF\classes\config.properties.

Dicho fichero contiene la variable de configuración que se encarga de definir el directorio de disco donde está ubicada nuestra base de conocimiento de rutas.

Variable	Descripción	Ejemplo
trackmixer.FileRouteDAOImpl.basedir	Directorio base donde se encuentra nuestra base de conocimiento.	**c:/trackMixer/misrutasbtt**