ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

“SISTEMA EJECUTOR DE UN NAVEGADOR PERSONALIZADO”

Nieves Gorriti Alastuey

Jesús Villadangos Alonso

Pamplona, 5 de Octubre de 2011

*I*NDICE DEL DOCUMENTO

1. RESUMEN

El proyecto fin de carrera titulado “Sistema ejecutor de un navegador personalizado” surgió con la idea de estudiar los sistemas de navegación existentes y poder añadirles nuevas funcionalidades de modo que se pueda saber en todo momento en dónde nos encontramos. Para ello hemos creado un sistema de navegación centrado en ciudades de modo que el usuario pueda configurar la ruta a realizar.

Básicamente, el usuario va a ser capaz de introducir el punto inicial y el final por el que desea pasar. A continuación, el sistema devuelve el mejor trayecto encontrado para la ruta indicada. Además, el usuario podrá ser capaz de introducir diferentes puntos de paso. Estos puntos de paso resultan útiles si el usuario desea ir a direcciones concretas o también si desea evitar direcciones dado que el tráfico es denso o la calle en ese momento está cortada por obras.

Otra funcionalidad del proyecto es ver en todo momento una simulación de la ruta a seguir que muestra la localización, en tiempo real, de las calles por las que se pasaría con esa ruta y las instrucciones a seguir a lo largo de la ruta.

Además, es importante poder calcular rutas alternativas de manera automática para que el usuario sea capaz de elegir la que más le guste o más se adecúe a sus necesidades en ese momento.

Por lo tanto, el navegador está personalizado para que el usuario pueda elegir distintos puntos por los que desee pasar. Además, y siguiendo con el tema de personalización del navegador, el proyecto ofrece la posibilidad de introducir la velocidad media a la que se desea ir y devuelve la nueva duración del trayecto a dicha velocidad. En caso de no introducir ninguna velocidad, la ruta se calculará con la velocidad que se utiliza por defecto.

Este proyecto está dividido en dos partes, una la parte gráfica de interacción con el usuario y otra la parte de ejecución de instrucciones. En este documento voy a profundizar en la parte de ejecución de instrucciones y explicaré la conexión que tiene con la parte gráfica que ha realizado mi compañero Andrés Goñi.

El objetivo de este documento es estudiar cómo funciona un sistema de navegación y ver qué funcionalidades nuevas se pueden añadir a los sistemas de navegación ya existentes. Para ello, he utilizado el sistema ya existente de Google Maps. Este sistema dispone de su propio API que permite a cualquier usuario con ciertos conocimientos de informática desarrollar su propio sistema de navegación personalizado.

2. INTRODUCCIÓN

En la actualidad, los sistemas de navegación son dispositivos integrados en la sociedad. La mayoría de los ciudadanos que utilizan ciertas tecnologías como Internet o teléfonos de última generación se han servido de sistemas de navegación en algún momento. Bien sea para localizar una calle concreta en una ciudad o para saber cómo llegar desde un punto a otro.

Hace ya varios años que surgieron los GPS (sistema global de posicionamiento) para los vehículos. Estos dispositivos son muy útiles para llegar a ciudades o calles que no conocíamos ya que son bastante precisos.

Al ver el éxito de los dispositivos GPS, varias empresas como Vía Michelin, Repsol o Google crearon sistemas de navegación en sus sitios Web para crear rutas y facilitar a los conductores moverse de un punto a otro. Más tarde, algunas empresas, como Google por ejemplo, ampliaron sus utilidades permitiendo también crear rutas dentro de ciudades y dando la opción de marcar etapas en su recorrido.

Hoy en día, los sistemas de navegación se utilizan cada vez más gracias a su incorporación en los teléfonos móviles de última generación. Por ejemplo, Nokia ha desarrollado su propio sistema de navegación para sus terminales y los teléfonos que utilizan el sistema operativo “Android” emplean “Google Maps”.

Los sistemas de navegación son de gran ayuda para cualquier tipo de usuario (bien sea transportistas, repartidores, empresarios que necesitan desplazarse a otras empresas o simplemente personas que buscan una dirección o comercio en concreto y que no saben muy bien dónde ubicarlo).

Muchos negocios disponen de sitios Web para promocionarse y muchos de estos sitios ofrecen un pequeño mapa en el que ubican su negocio para facilitar a los clientes la localización del negocio.

En este proyecto vamos a estudiar cómo funciona un sistema de navegación y ver qué funcionalidades se le pueden añadir para que el usuario pueda personalizar su ruta.

2.1. ESTADO ACTUAL

En la actualidad existen diferentes sistemas de navegación que facilitan a los usuarios trazar sus rutas, organizar sus vacaciones y planificar sus trayectos.

Antes de comenzar el proyecto necesitamos ver todo lo que nos ofrecen los sistema de navegación actuales, cuáles son las diferencias entre ellos, cuáles nos permiten añadir funcionalidades que queremos y cuáles nos permiten desarrollar nuestro propio sistema de navegación.

A continuación, vamos a estudiar los sistemas más utilizados en la actualidad y haremos una comparativa de ellos.

- VÍA MICHELÍN: www.**viamichelin**.es

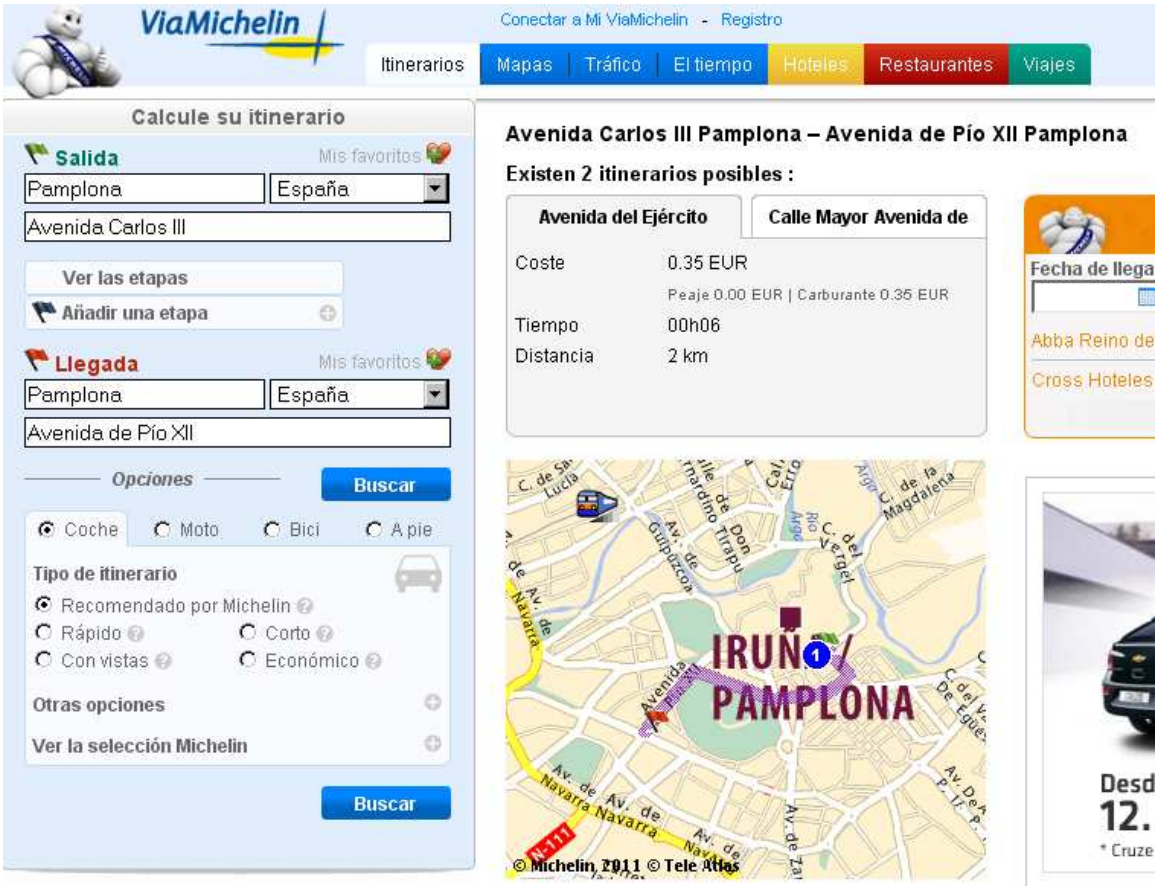


Imagen 1: pantallazo del sitio Web de la Vía Michelin

Vía Michelin nos ofrece la posibilidad de introducir una dirección concreta de una ciudad desde la que inicia la ruta y otra a la que se llega. Además, ofrece itinerarios para ir en coche, bici, en moto o a pie y la posibilidad de elegir el tipo de itinerario (económico, rápido, corto, etc.).

Muestra, como resultado el coste, el coste de los peajes, el tiempo, la distancia y la ruta representada en el plano.

Al hacer clic sobre el mapa, la Vía Michelin ofrece también la opción de añadir una etapa al trayecto y calcular el gasto de carburante en función del precio de la gasolina.

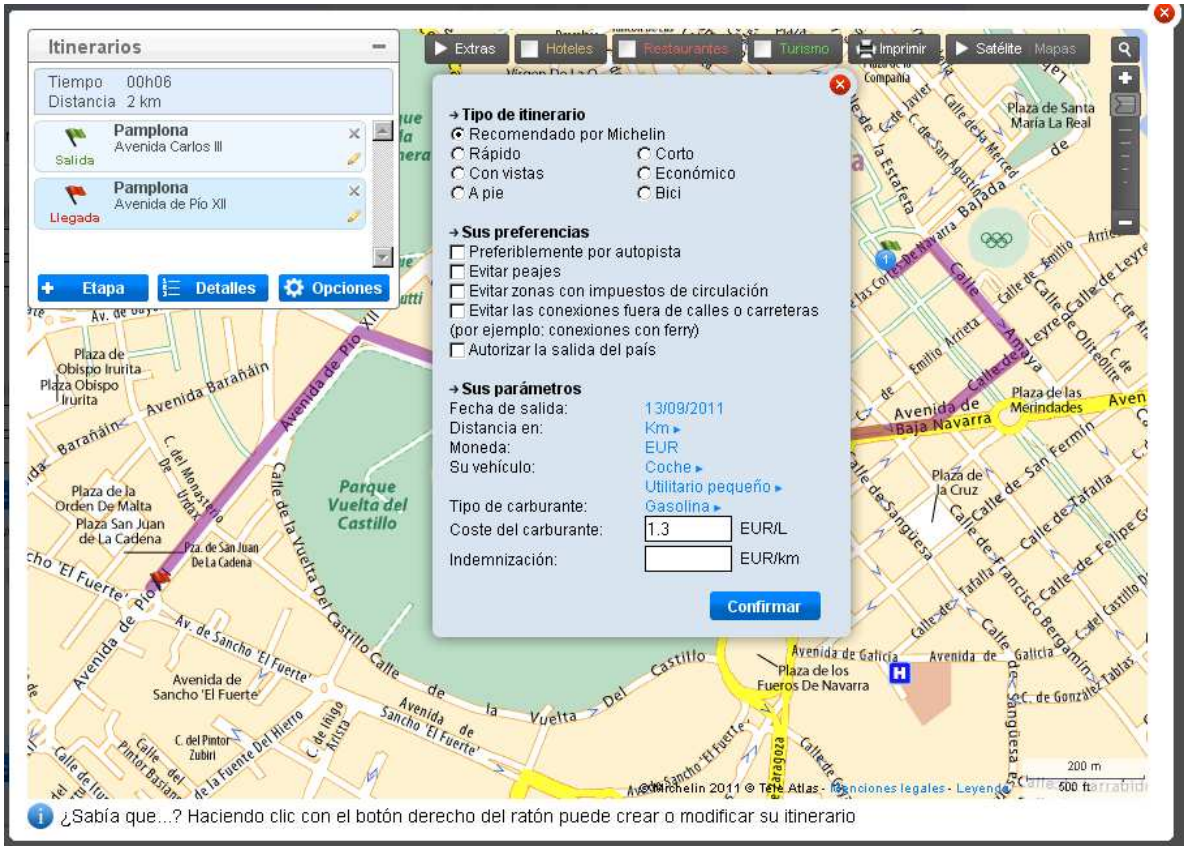


Imagen 2: Pantallazo de los resultados de la Vía Michelin

También ofrece la posibilidad de visualizar en el mapa radares, servicios de la ciudad (restaurantes, hospitales, bares, etc.) así como una explicación detallada de la ruta.

Lo que no ofrece la Vía Michelin es la posibilidad de ver rutas alternativas para realizar ese trayecto ni una simulación del trayecto para ver gráficamente un icono que siga la ruta.

- GUIA CEPSA: www.buenviajecepsa.com

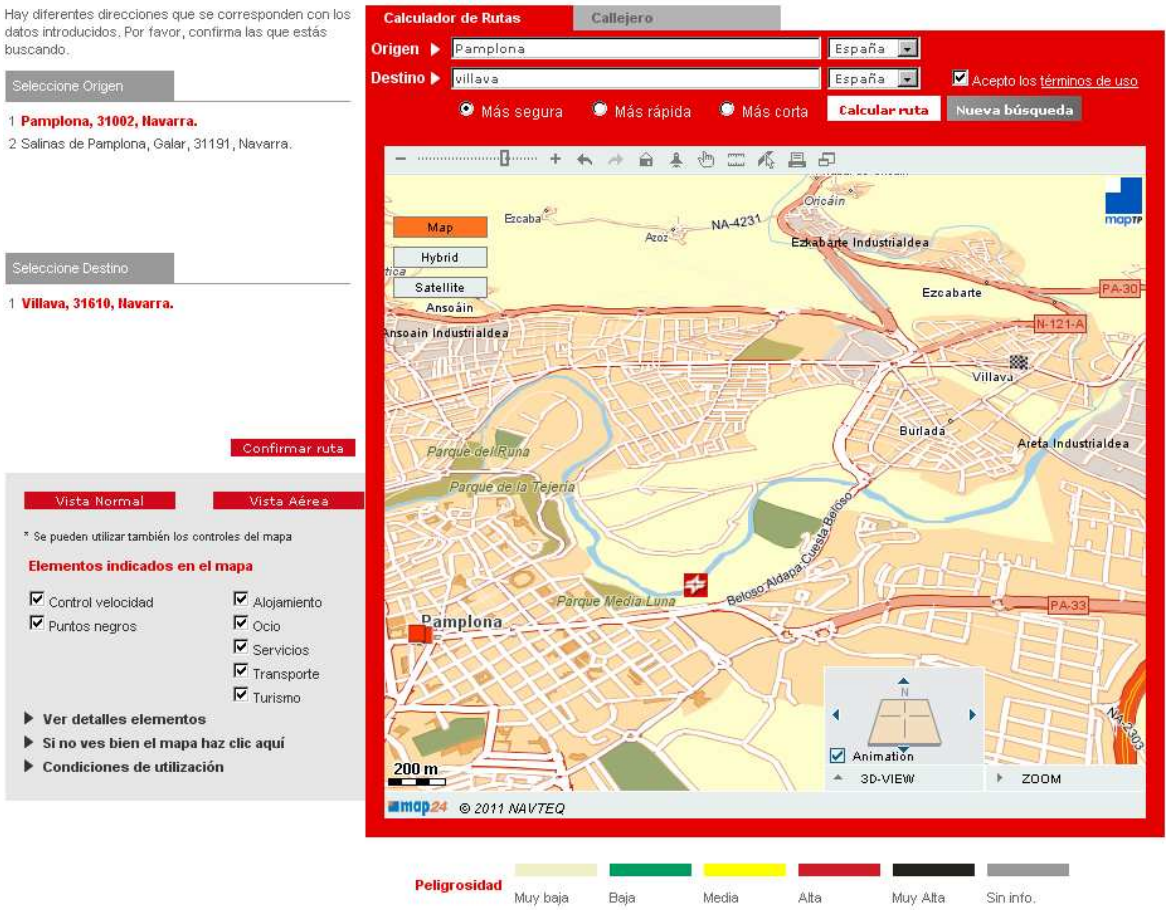


Imagen 3: Pantallazo del sitio Web de Cepsa

El sistema de navegación de Cepsa es bastante rudimentario ya que ni siquiera ofrece una representación gráfica de la ruta a seguir ni información de cómo alcanzar el punto de destino. Sólo muestra una marca en el punto de inicio y otro en el punto final.

Permite ver servicios de la ciudad y escoger entre el tipo de ruta (rápido, seguro o corto).

Lo que no permite es introducir etapas en el camino o ver el coste económico ni la duración. Tampoco ofrece la opción de ver rutas alternativas.

Lo que sí ofrece este sistema y no el de la Vía Michelin es la opción de ver la peligrosidad de las rutas (muy baja, baja, media, alta, muy alta o sin información).

- GUÍA REPSOL: http://www.guiarepsol.com/es_es/home/

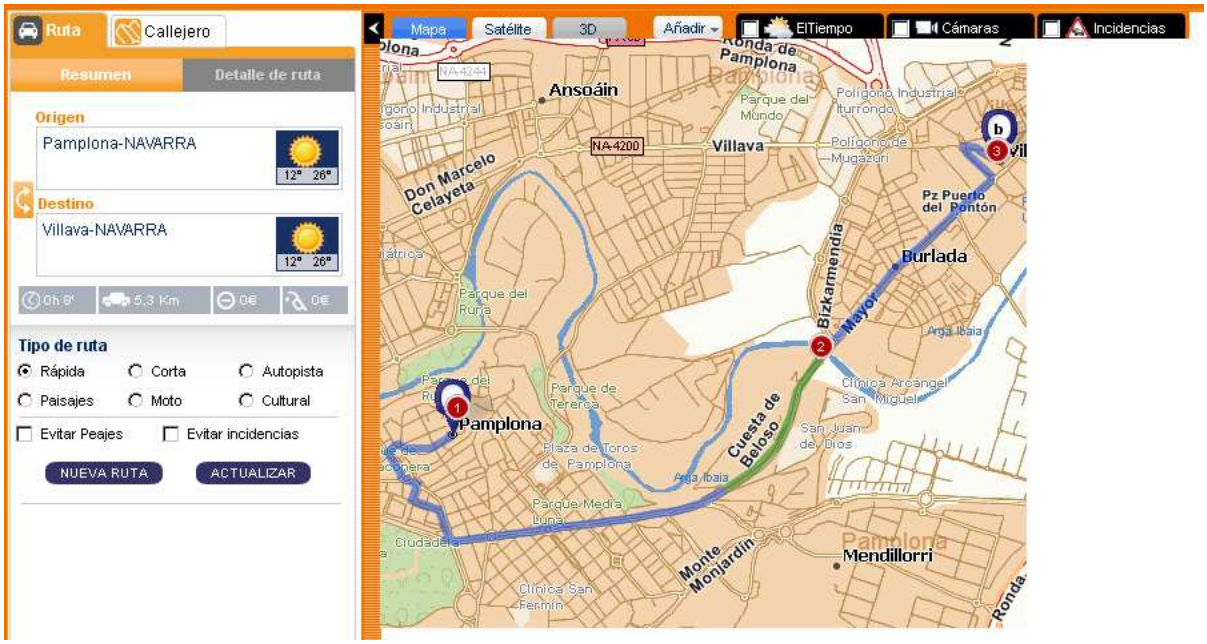


Imagen 4: Pantallazo de la Guía Repsol

La Guía Repsol es más completa y útil. Ofrece la posibilidad de introducir una dirección de origen y un destino y muestra los resultados en un mapa de manera gráfica y como texto mostrando indicaciones de la ruta a seguir.

Además, permite seleccionar tipos de ruta y muestra el tiempo, el coste de carburante y de peajes y la duración del trayecto.

La guía Repsol muestra los radares existentes y proporciona la opción de evitar peajes e incidencias.

Lo que no ofrece es la posibilidad de hacer una simulación gráfica del trayecto ni muestra los servicios disponibles en la ruta. Tampoco permite añadir puntos de paso en el trayecto.

A diferencia de los sistemas de navegación anteriores, la Guía Repsol sí ofrece la posibilidad de ver rutas alternativas y muestra su nueva duración.

Sistema ejecutor de un navegador personalizado

- GOOGLE MAPS: http://maps.google.es/maps

Google Maps es el sistema de navegación más utilizado en la actualidad.

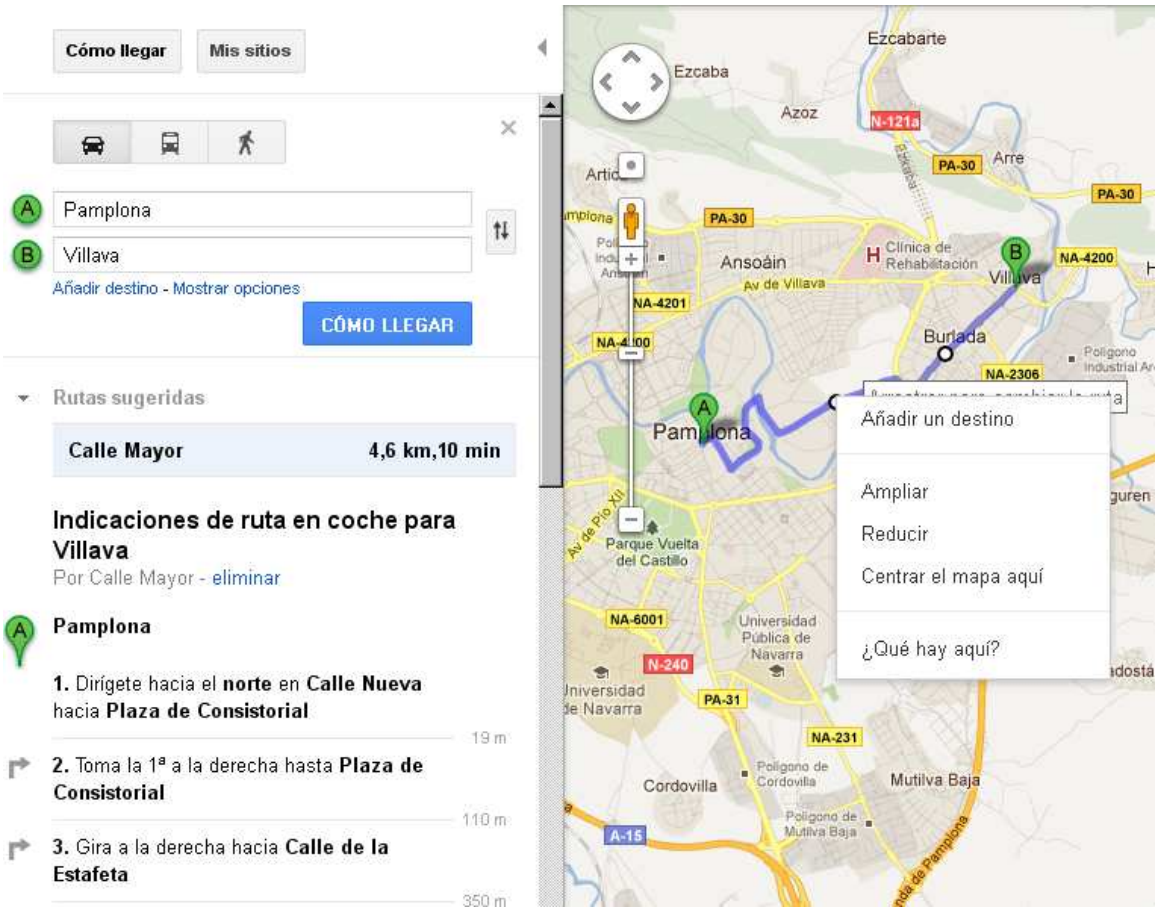


Imagen 5: Pantallazo de Google Maps

Google Maps muestra en un mapa el resultado de manera gráfica. Además, ofrece la posibilidad de arrastrar tanto la ruta como los marcadores para recalcular la ruta de manera más rápida.

Sí ofrece la posibilidad de añadir etapas en el trayecto y además, también muestra la información de la ruta con texto en la ventana de la izquierda explicando las indicaciones que deben seguirse y los metros de cada explicación. Muestra también el tiempo y la distancia total para completar el trayecto.

Lo que no ofrece es la posibilidad de elegir entre rutas alternativas ni el coste económico de la ruta. Así como tampoco muestra servicios de la ruta, ni radares ni el gasto del carburante. Tampoco se puede realizar una simulación de la ruta en tiempo real.

Lo que si permite, a diferencia de todos los sistemas de navegación existentes, es desarrollar nuestro propio sistema de navegación gracias al API de Google Maps que ha desarrollado Google y el cual ofrece muchas funcionalidades para añadir.

A continuación mostramos una tabla comparativa de todos los sistemas de navegación analizados:

Propiedades	Michelin	Google Maps	Cepsa	Repsol
Punto de salida y de llegada	Sí	Sí	Sí	Sí
Diferentes modos de viaje	Sí	Sí	No	No
Introducir etapas	Sí	Sí	No	No
Elegir económico, corto, etc.	Sí	Sí	Sí	Sí
Simulación de la ruta	No	No	No	No
Gasto carburante	Sí	No	No	Sí
Radares	Sí	No	No	Sí
Mostrar servicios ciudad	Sí	No	Sí	No
Información detallada de la ruta	Sí	Sí	No	Sí
Coste económico	Sí	No	No	Sí
Tiempo	Sí	Sí	No	Sí
Distancia	Sí	Sí	No	Sí
Rutas alternativas	No	No	No	Sí
Peligrosidad de la ruta	No	No	Sí	No
Programable	No	Sí	No	No
Ruta “draggable”	No	Sí	No	No

Imagen 6: Tabla comparativa de los sistemas de navegación más populares

Ahora nosotros tenemos que ver cuál de todos ellos es el que mejor se adapta a nuestras necesidades en función de las características que cumplen o no.

2.2. OBJETIVOS

La idea por la que surgió este proyecto es la de conocer cómo funciona un sistema de navegación para después poder darle distintos usos. La clave es estudiar lo que ya existe para poder incorporarle más funcionalidades, no crear desde cero un dispositivo de navegación.

El principal objetivo del proyecto es crear un sistema de navegación centrado en la ciudad, que permite simular la ruta entre dos puntos obteniendo toda la información posible de los puntos de paso (si son rectas, rotondas, etc.) Además, es importante que podamos introducir la velocidad para poder hacer una simulación real. Por ejemplo, si estamos en una recta la velocidad será mayor que en una rotonda. Para conseguir todo esto, el usuario debe ser capaz de introducir un punto de inicio y de fin para que se calcule una ruta.

No nos importa tanto el coste económico ni los servicios existentes en la ciudad. También es interesante encontrar un modo de calcular rutas alternativas por si se da el caso de que haya atascos y también la opción de añadir etapas por si necesitamos desviarnos de la ruta o en caso de que no queramos pasar por un punto en concreto.

Según los sistemas de navegación vistos en el apartado anterior, existen muchas opciones que podemos obtener pero ninguno de ellos nos ofrece la posibilidad de realizar una simulación gráfica del trayecto.

Google Maps sin embargo ofrece un API para permitir a los desarrolladores crear sistemas de navegación personalizados. Por lo tanto, hemos elegido el sistema de navegación de “Google Maps”. Google ha creado su propio API para flash y javaScript que permite añadir muchas funcionalidades a un sistema de navegación personalizado.

Necesitamos estudiar el API para saber cómo se pasan los parámetros a los servidores de Google y ver qué nos devuelven estos servidores para poder usar cierta información de toda la que devuelve.

Gracias a “Google Maps” seremos capaces de acceder a prácticamente todos los lugares del mundo y con diferentes tipos de mapas. Además, tenemos la opción de desarrollar nuestro propio sistema de navegación gracias al API existente tanto para JavaScript como para Flash.

No vamos a calcular la ruta entre dos puntos, ya que de eso se encargan los servidores de Google que son un Sistema de Información Geográfico, sino que vamos a saber cómo conectar con los servidores y cómo recoger los parámetros para obtener las salidas que deseamos. Lo más importante es conocer cómo seleccionar los puntos de paso y saber cómo pasárselos a los servidores. Ver si es necesario distinguir entre punto de inicio, de paso o punto final, así como ver qué más parámetros son necesarios para iniciar el cálculo. Además, también queremos conocer qué devuelven los servidores de mapas, y ver cómo lo devuelven. Saber cómo recoger esos parámetros para luego jugar con ellos y mostrar aquellos que nos interesan o no.

3. ARQUITECTURA DEL SISTEMA

Un sistema de navegación está formado por una interfaz en la cual el usuario introduce los puntos origen y destino y en algunos casos también los puntos de paso y se muestra la ruta calculada sobre un mapa en la misma interfaz.

Para conseguir una ruta, se pasan los puntos origen y destino y opcionalmente también se pasan los puntos intermedios a los servidores de "Google Maps" donde calcula la ruta óptima y la devuelve en forma de ruta y/o direcciones escritas.

Por lo tanto, existen dos partes diferenciadas en el proyecto. Por un lado está la interfaz, en la cual se recogen los parámetros a calcular y se muestra el resultado. Por el otro lado está el paso de los parámetros a los servidores y la obtención del resultado para poder ser mostrado.

En resumen, estas dos partes son muy diferentes y las hemos dividido entre dos personas. Es por ello que mi proyecto se complementa con el proyecto titulado "Captura y visualización de datos en un navegador GPS". El proyecto nombrado se encarga de desarrollar una interfaz que recoja la información necesaria para que los servidores de Google Maps realicen la consulta. Además, muestra los resultados en la interfaz y permite realizar nuevas consultas y funciones sobre los resultados.

Mi proyecto se encarga de recibir esos datos y estructurarlos para poder realizar la consulta a los servidores y una vez realizada la consulta recoger la información que devuelve y pasarle al otro proyecto los datos que necesite para mostrar los resultados.

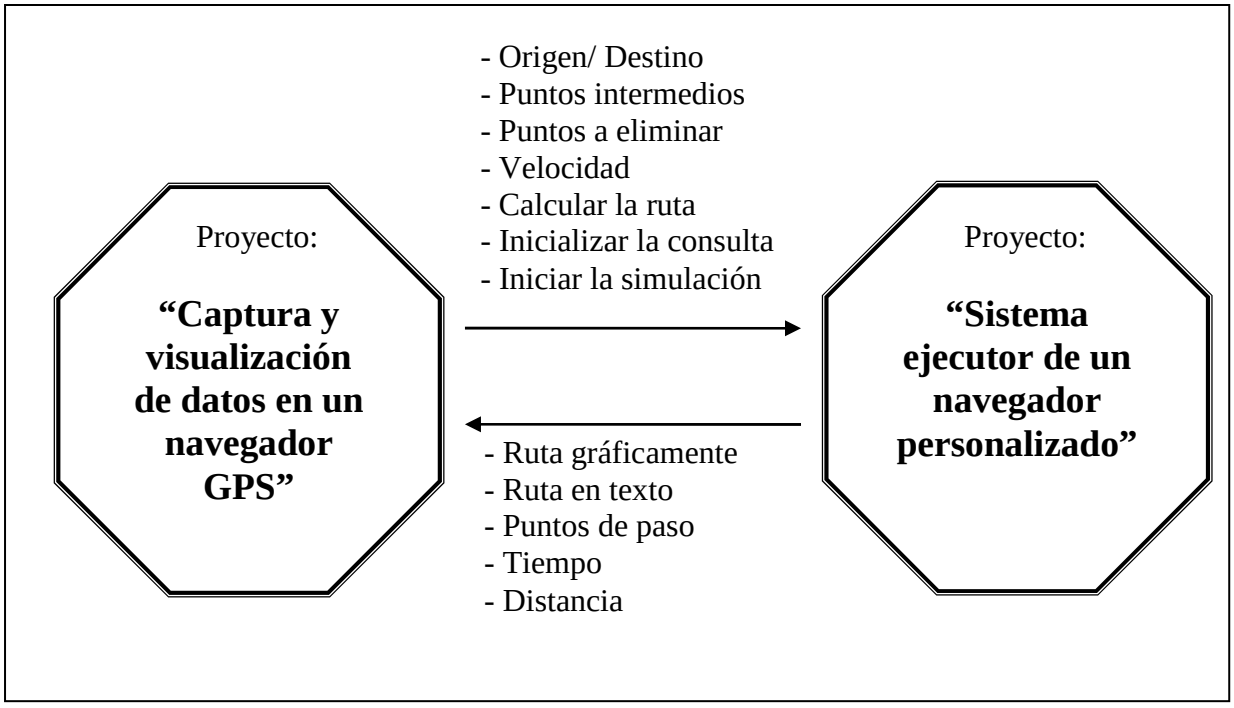


Imagen 7: Representación gráfica de la división del trabajo y los parámetros que intercambian.

Como vemos en la Imagen 7, ambos proyectos se necesitan para realizar su trabajo. En primer lugar, se recogen los puntos origen y destino que ha introducido el usuario. Sólo con eso, el sistema ejecutor ya es capaz de calcular la ruta optimizada para llegar desde el origen hasta el destino. Además, el usuario puede añadir puntos de paso que el sistema ejecutor incorporará a sus parámetros para modificar la obtención de la ruta.

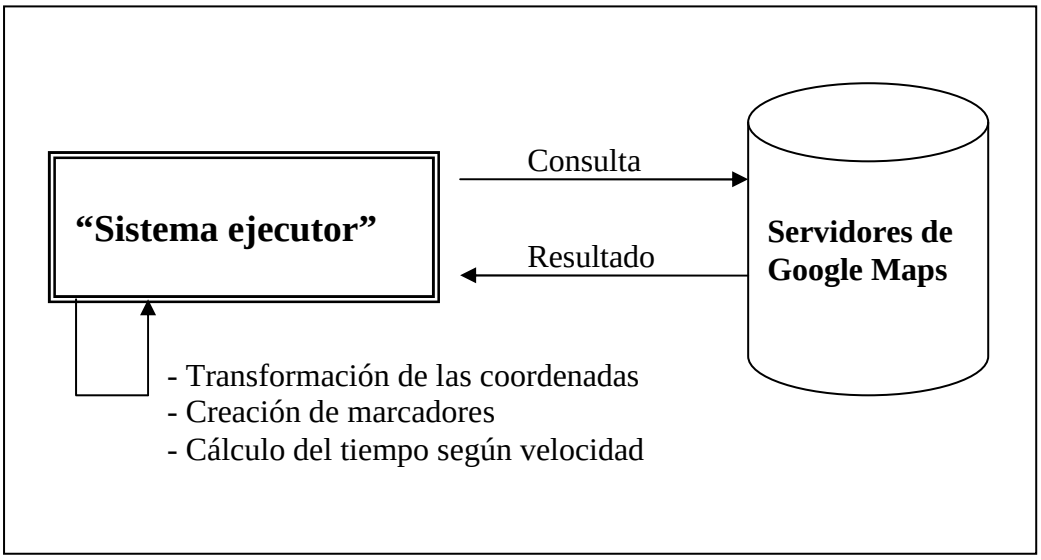


Imagen 8: Representación gráfica del paso de parámetros a los servidores de Google.

En la imagen 8 he representado el trabajo correspondiente a mi proyecto, ejecutar la orden y obtener resultados. El propio sistema ejecutor se encarga de recibir la dirección que puede ser en coordenadas (latitud, longitud) o como cadena de texto. En este último caso será necesario transformarlo a coordenadas o al revés en función del uso que se le vaya a dar. Así por ejemplo, será necesario convertir las coordenadas que devuelve el servidor a cadena de texto para que sea comprensible por el usuario.

Otra de las funciones del sistema ejecutor será el de pasar a la interfaz el punto por el que realmente pasa la ruta. Esto se debe a que el usuario puede seleccionar un punto que no existe realmente por lo que se debe representar en la ruta real lo más cerca posible de donde ha seleccionado el usuario.

Además, el servidor de Google Maps devuelve la distancia de la ruta y el tiempo necesario en recorrerla, por lo que si el usuario introduce una velocidad determinada es necesario recalcular el tiempo a dicha velocidad.

Por otra parte, la consulta se envía a los servidores de Google Maps que son los encargados de calcular la ruta óptima.

3.1- INTERACCIÓN DE LAS PARTES

Teniendo en cuenta que para la realización de este proyecto hemos tenido que dividirlo entre dos personas, va a ser imprescindible que la interacción entre las dos partes sea buena. Vamos a tener que mantener una consistencia en los datos y elementos que sean comunes para ambas partes para así poder trabajar sin errores.

La primera parte, encargada de la captación de los datos que aporte el usuario y de la representación de los posteriores resultados, deberá iniciar el almacenamiento de algunos datos que después utilizará el sistema ejecutor. Para ello, se establecerán una serie de estructuras de almacenamiento que sean adecuadas para el posterior uso en las funcionalidades que desarrolle el segundo proyecto.

Principalmente vamos a utilizar puntos de localización para la mayoría de actuaciones, ya sea de forma individual o un grupo de los mismos. Por ello, tendremos que tener un criterio común, que cree el sistema de captación y que utilice el sistema ejecutor, y que además le permita a este último trabajar sin problemas con las diferentes funcionalidades que proporciona Google Maps en su API.

Los puntos de localización se representan con dos valores, la latitud y la longitud. También es posible obtener de esos dos valores la dirección en una cadena de texto, pero esa representación es únicamente para el usuario. Internamente, trabajaremos siempre con la latitud y la longitud. Para conservar la consistencia limitaremos el número de decimales para tener la misma estructura en todos ellos y así poder hacer coincidir los puntos recogidos en la interfaz y los recogidos en los servidores de Google.

Para conservar la coherencia es necesario redondear todos los valores que sean de la forma Latitud/Longitud. El problema que nos encontramos era que, de un mismo punto, los servidores de Google podían devolverlo con distinto número de decimales cada vez. Por ello, se nos hacía muy complicado saber con qué punto queríamos trabajar. Así por ejemplo, si un usuario quería eliminar un punto y teníamos que ver en cuál había hecho clic, podía darse el caso de que no lo tuviéramos almacenado con ese mismo número de decimales y por eso nos daba fallo.

La manera de resolver este conflicto fue el de separar por un lado la latitud y por otro la longitud y redondearlos a 12 decimales. Después, convertíamos otra vez los dos valores en uno solo de la clase Latitud/Longitud. Creamos la siguiente función:

```
////////////////////////////////////////////////////////////////////////////////
//Funcion que limita el numero de decimales de una coordenada y crea un nuevo latlng//
////////////////////////////////////////////////////////////////////////////////
function pintarOrigenDestino(a){
    var latitud = a.lat().toFixed(12);
    var longitud = a.lng().toFixed(12);
    return new google.maps.LatLng(latitud,longitud,{noWrap:true});
}
```

Además, cuando necesitemos guardar más de un punto, lo haremos en un Array. Diferenciaremos dos puntos que emplearemos de diferente forma. Estos puntos son el inicio y el final de la ruta, que, aunque tendrán la misma estructura, tendrán usos algo diferentes del resto. Es por ello, que los tendremos almacenados en variables independientes.

El sistema de captación de datos pasará los puntos al sistema ejecutor cuando el usuario decida calcular la ruta. Una vez ya esté calculada, el sistema ejecutor seleccionará los datos importantes y se los pasará al sistema de captación para que los muestre. También podrá ordenarle que realice diversas acciones sobre la misma, como simular el recorrido o calcular rutas alternativas.

Si se decide borrar un punto de paso, o todos, las estructuras mencionadas se deberán actualizar para que en futuras solicitudes se realicen de acuerdo a lo especificado por el usuario.

Para la mayoría de las acciones que realice el usuario, se va a necesitar la interactuación de la interfaz con el sistema ejecutor, por lo que es imprescindible que la comunicación entre ambas partes sea adecuada.

4. HERRAMIENTAS: API DE GOOGLE MAPS [7, 8, 9]

Como ya he explicado anteriormente, Google ha desarrollado su propio API para que personas con cierto grado de nivel informático puedan ser capaces de configurar sus propios mapas y darles usos específicos según necesidades.

El API está disponible tanto para JavaScript como para Flash. Nosotros hemos utilizado el API de JavaScript para desarrollar, desde cero, una aplicación centrada en ciudades y personalizada por los usuarios. Así, pudimos comprobar cómo funciona la consulta de rutas y la devolución de las mismas.

En el 2010, Google lanzó la versión 3 del API de JavaScript. Esta versión era muy diferente de la anterior ya que ampliaba funcionalidades pero cambiaba la mayoría de comandos y la manera de trabajar con los datos. Todavía son muchos los desarrolladores que siguen usando la versión 2 ya que existen multitud de ejemplos en Internet al contrario que de la 3 que cada vez es más usada pero los ejemplos tampoco son abundantes.

Nosotros comenzamos trabajando con la versión 2 ya que fue la que nos resultó más sencilla para comenzar, pues los ejemplos son múltiples. Cuando llegamos al punto en el que queríamos trabajar sobre los marcadores que Google devuelve automáticamente con la consulta nos encontramos con que en esta versión era imposible, así que nos vimos obligados a estudiar también la versión 3 y comprobamos que en esa versión íbamos a ser capaces de trabajar con los marcadores.

En este proyecto he investigado cómo poder recibir los datos de la interfaz, saber cómo tengo que estructurar estos datos para poder enviarlos al sistema de información geográfica y cómo recibirlos para poder pasárselos de nuevo a la interfaz y que el usuario pueda ser capaz de obtener el resultado que buscaba.

A continuación explico las clases y métodos disponibles en el API que he utilizado para poder realizar mi proyecto.

4.1 PARTE GRÁFICA

En este apartado voy a hacer una breve explicación de la parte gráfica del proyecto. Esta parte no corresponde a mi proyecto pero es necesaria para poder entender las funcionalidades de mi proyecto por lo que pienso que es necesaria una explicación.

Imagen 9: Pantallazo de nuestra interfaz tras obtener resultados

4.1.1 EL MAPA

La idea del proyecto era centrarnos en una ciudad para que el usuario sea capaz de optimizar las rutas por las que pasa diariamente. Por ello, la parte gráfica se centra en la ciudad de Pamplona. Básicamente porque es en la que nos encontramos pero se puede modificar por el usuario desde la interfaz.

Para crear el mapa con el que el usuario va a interactuar Google creó la clase “Map”:

```
Map(mapDiv:Node, opts?:MapOptions)
```

La clase “Map” crea un mapa nuevo dentro del contenedor HTML en cuestión, que generalmente suele ser un elemento `DIV`. En las opciones del mapa podemos especificar elementos como el punto inicial en el que se va a centrar, el nivel de zoom o el tipo de mapa que queremos utilizar (en coche, a pie o en bicicleta).

4.1.2 PUNTOS INICIAL Y FINAL

Una vez que el usuario carga la aplicación debe introducir de qué punto a qué punto desea ir. Lo normal es que el usuario no sepa la latitud y longitud del punto por lo que introduce una cadena de texto con la dirección. Estos dos valores se almacenan en dos variables para que posteriormente pueda trabajar con ellos:

```
puntoInicial = document.form_ruta.desde.value;
puntoFinal = document.form_ruta.hasta.value;
```

Con estos dos puntos ya puedo crear una consulta y obtener resultados de cómo llegar, pero también existe la opción de introducir puntos intermedios en la ruta.

4.1.3 PUNTOS DE PASO

Es posible que el usuario quiera introducir puntos de paso externos a la ruta que se crea por defecto. Con estos puntos de paso se puede evitar y lograr pasar por un punto. En este caso, los puntos de paso no los vamos a introducir como cadena de texto, si no que para facilitar el trabajo bastará con pinchar sobre el mapa para desviar la ruta. Otra opción para añadir puntos de paso es arrastrar los ya existentes a otro punto del mapa y así ya hemos conseguido pasar por el punto deseado. Para poder trabajar con estos puntos hemos creado un Array (“listaPuntos”) en el que los almacenamos.

4.1.4 MARCADORES

Los marcadores se crean con la clase “Marker” del API de Google:

```
Marker(opts?:MarkerOptions)
```

Son iconos gráficos que muestran el punto por el que inicia, pasa o finaliza la ruta. El servidor de Google Maps devuelve por defecto unos marcadores, pero creemos que es imposible trabajar con ellos (por ejemplo en el caso de querer arrastrarlos). Por eso, nosotros creamos nuestros propios marcadores personalizados.

4.1.5 RUTA

Tras procesar la información y calcular la ruta, los servidores de Google Maps devuelven la ruta en forma de línea que une los puntos seleccionados. Esta línea supone el recorrido a seguir por el usuario.

4.2 EJECUCIÓN DE INSTRUCCIONES

Los servidores de Google son los que trazan la ruta en función de los parámetros que elige el usuario. Pueden recibir un único punto, dos o varios.

En el caso de que reciba uno, interpreta como punto de inicio el punto desde el que se encuentra el usuario y el introducido será el punto de fin.

En el caso de introducir dos puntos, consideramos que uno es el punto de inicio y otro el punto final. Si se introducen varios puntos, es necesario indicar cuáles son el punto inicial y el punto final y el resto son considerados como puntos de paso por los que debe pasar la ruta.

Lo importante en mi proyecto es saber cómo pasar todos los parámetros a los servidores para que interpreten bien cada punto. Los parámetros que debo pasar son:

- Punto inicial
- Punto final
- Puntos de paso (opcionales)
- Modo de viaje: a pie, en coche o en bicicleta
- Opción de optimizar la ruta o no

Tras a consulta, la otra función básica de mi proyecto es saber cómo recoger los datos para poder pasárselos a la parte de la interfaz para que puedan ser visibles por el usuario. Los datos que debo recoger son:

- La representación de la ruta
- Información de cada punto de paso
    - Tiempo
    - Distancia
    - Dirección
- Distancia total
- Tiempo total
- Todos los puntos de paso para realizar la representación gráfica
- Rutas alternativas

Es por este motivo que voy a dividir mi explicación en dos. En una parte explicaré la parte de envío de datos a los servidores y en la otra la parte la recepción de los datos para pasarlos a la parte gráfica.

4.2.1 ENVÍO DE DATOS A LOS SERVIDORES

En esta sección explico cómo poder conectarse con los servidores enviándole todos los parámetros que necesitan.

La conexión con los servidores de Google Maps se realiza a partir de la clase "*DirectionsService*".

La clase "*DirectionsService*" es un servicio que permite calcular indicaciones entre dos o más lugares.

Constructor:

Constructor	Descripción
`DirectionsService()`	Crea una instancia nueva de un `DirectionsService` que envía consultas de indicaciones a los servidores de Google.

Por lo tanto, hemos creado una variable y la hemos creado con el constructor de esta clase de la siguiente manera:

```
var directionsService = new google.maps.DirectionsService();
```

Método:

Métodos	Valor de retorno	Descripción
`route(request:DirectionsRequest, callback:function(DirectionsResult, DirectionsStatus))`	`None`	Emite una solicitud de búsqueda de indicaciones.

Este método es el más importante para poder desarrollar un sistema de navegación personalizado. Es el conecta directamente con los servidores y les pasa la ruta. En el parámetro "*request*" se le pasa un "*DirectionsRequest*" con todos los parámetros que ha introducido el usuario y tras ser enviado a los servidores devuelve toda la información en "*DirectionsResult*" y el estado de la consulta en "*DirecitonsStatus*" el cual nos indica si ha habido éxito o no.

Así es como nosotros hemos implorado este método:

```
directionsService.route(request, function(response, status) {
```

Como en esta parte me he centrado en el envío de parámetros a los servidores, voy a explicar el "*DirectionsRequest*" para poder comprender y realizar una consulta con todos los parámetros escogidos por el usuario.

"*DirectionsRequest*" es una consulta de indicaciones que se enviará a DirectionsService. Permite configurar la consulta y la respuesta gracias a las propiedades que tiene como por ejemplo optimizar la ruta, evita autopistas, ofrecer rutas alternativas o el modo de viaje entre otros. Estas son todas las opciones que ofrece:

Propiedades	Tipo	Descripción
avoidHighways	boolean	Si está establecido en "true", indica al servicio Directions que evite las carreteras principales donde sea posible. Opcional
avoidTolls	boolean	Si está establecido en "true", indica al servicio Directions que evite las autopistas de peaje donde sea posible. Opcional
destination	LatLng|string	Es la ubicación de destino. Se puede especificar como una cadena a la que se asignará un código geográfico o como un objeto LatLng. Obligatorio
optimizeWaypoints	boolean	Si está establecido en "true", DirectionService intentará volver a ordenar los hitos intermedios proporcionados con el fin de minimizar el coste total de la ruta. Si se optimizan los hitos, inspecciona DirectionsRoute.waypoint_order en la respuesta para determinar el nuevo orden.
origin	LatLng|string	Es la ubicación de origen. Se puede especificar como una cadena a la que se asignará un código geográfico o como un objeto LatLng. Obligatorio
provideRouteAlternatives	boolean	Indica si se deben proporcionar alternativas a la ruta. Opcional
region	string	Es el código de región que se utiliza como especificación de las solicitudes de codificación geográfica. Opcional
travelMode	DirectionsTravelMode	Es el tipo de ruta solicitada. Obligatorio
unitSystem	DirectionsUnitSystem	Sistema de unidades preferido para mostrar la distancia. De forma predeterminada, es el sistema de unidades utilizado en el país de origen.
waypoints	Array.<DirectionsWaypoint>	Es el conjunto de hitos intermedios. Las indicaciones se calculan desde el origen hasta el destino pasando por los hitos incluidos en este conjunto. Opcional

Los únicos puntos obligatorios en toda consulta son el origen, el destino y el modo de viaje. Nosotros además, hemos utilizado la opción de "*waypoints*" para poder hacer paradas en el trayecto; hemos utilizado la propiedad de "*optimizeWaypoints*" para que siempre nos calcule la ruta óptima entre varios puntos y por último hemos puesto el valor "*provideRouteAlternatives*" a true para poder obtener rutas alternativas y no sólo la ruta óptima que devuelve por defecto el servidor.

El "*DirectionsRequest*" que he creado tiene la siguiente estructura:

```
//Creamos la consulta que queremos calcular
var request = {
    origin: puntoInicial,
    destination: puntoFinal,
    waypoints: listaPuntos,
    optimizeWaypoints: true,
    provideRouteAlternatives: true,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
};
```

Pasamos los puntos inicial y final en "*origin*" y "*destination*". Estos dos valores nos los pasa la parte gráfica cuando el usuario ha introducido los valores en los dos inputs destinados para ello:

```
puntoInicial = document.form_ruta.desde.value;
puntoFinal = document.form_ruta.hasta.value;
```

Estos valores se actualizarán siempre que el usuario decida cambiarlos desde los inputs y por tanto se volverá a recalcular la ruta con el nuevo punto inicial o punto final.

"*Waypoints*" es nuestra lista de puntos intermedios por los que desea pasar el usuario. En la primera consulta la lista está vacía ya que por defecto, se calcula la ruta con el punto inicial y final y después el usuario va seleccionando sobre el mapa por qué lugares desea pasar.

Estos puntos intermedios también se pueden eliminar o arrastrar sobre el mapa y acto seguido se actualizan los valores del "*DirectionsRequest*" y se vuelve a calcular la ruta.

Gracias a "provideRouteAlternatives" tenemos la opción de mostrar diferentes rutas para un punto de inicio y de fin, por lo que es una opción muy útil para evitar posibles atascos o comparar tiempos y distancias entre dos puntos.

"*listaPuntos*" debe ser un array de tipo "*DirectionsWaypoints*", que representa una ubicación entre el origen y el destino a través de la cual debe pasar la ruta de viaje. A continuación explico las propiedades de "*DirectionsWaypoints*":

Propiedades	Tipo	Descripción
`location`	`LatLng|string`	Es la ubicación del hito. Puede ser una cadena de dirección o un objeto `LatLng`. Opcional
`stopover`	`boolean`	Si está establecido en `true`, indica que este hito constituye una parada entre el origen y el destino. Esto provoca que la ruta se divida en dos. El valor predeterminado es `true`. Opcional

Trabajar con estos arrays en JavaScript es un poco especial. Debe hacerse de la siguiente manera:

Si se desea añadir un elemento realizo la siguiente instrucción:

```
//Anado los puntos al array de puntos intermedios
listaPuntos.push({
    location:event.latLng,
    stopover:true
});
```

Para eliminar un elemento cualquiera e incluso su posición, por ejemplo en el caso de que se arrastre el marcador a otra posición es necesario realizar la siguiente instrucción:

```
for (i=0;i<listaPuntos.length;i++){
    numMarcador=i;
    //Borro el contenido de coorde de la posicion numMarcador para borrar ese punto
    var aux =listaPuntos.slice(numMarcador+3);
    coorde = listaPuntos.slice(0,numMarcador+2);
    coorde = listaPuntos.concat(aux);
    break;
}
```

Lo que hacemos es recorrer todos los puntos guardados en "*listaPuntos*" y almacenamos dos elementos del array para después concatenarlos y eliminando el no deseado.

Por último, si se desea borrar el último punto introducido porque el usuario se ha equivocado realizaríamos lo siguiente:

```
listaPuntos.pop();
```

La siguiente propiedad que utilizo es "*optimizeWaypoints*". Considero que optimizar los puntos de paso es una opción muy importante ya que permite reordenar los puntos para hacer el mejor camino entre todos los puntos. Es una manera de saber cómo mejorar el tiempo para hacer todas las paradas deseadas. Es muy útil tanto para ahorrar tiempo como para ahorrar en combustible en empresas que se dedican al transporte.

Por último indicio el modo de viaje "*travelMode*" el cual puede ser a pie, en coche o en bicicleta y la ruta cambia en función del parámetro seleccionado:

Constante	Descripción
BICYCLING	Especifica una solicitud de indicaciones para llegar en bici.
DRIVING	Especifica una solicitud de indicaciones para llegar en coche.
WALKING	Especifica una solicitud de indicaciones para llegar a pie.

4.2.2 DEVOLUCIÓN DE LA RUTA

Una vez realizada la consulta y enviada ahora debo recoger la respuesta para poder pasársela a la parte gráfica.

En primer lugar debo controlar si la consulta ha tenido éxito o no. El parámetro "*status*" es de tipo "*DirectionsStatus*" y es quien me va a dar esa información y puede tener los siguientes valores:

Constante	Descripción
`INVALID_REQUEST`	La solicitud `DirectionsRequest` proporcionada no es válida.
`MAX_WAYPOINTS_EXCEEDED`	Indica que se proporcionaron demasiados hitos (`DirectionsWaypoint`) en `DirectionsRequest`. El número máximo de hitos permitido es ocho, además del origen y del destino.
`NOT_FOUND`	No se ha podido codificar geográficamente el origen, el destino o los hitos.
`OK`	Indica que la respuesta contiene un valor `DirectionsResult` válido.
`OVER_QUERY_LIMIT`	La página web ha superado el límite de solicitudes en un período de tiempo demasiado breve.
`REQUEST_DENIED`	Indica que no se permite el uso del servicio de indicaciones en la página web.
`UNKNOWN_ERROR`	No se pudo procesar una solicitud de indicaciones debido a un error del servidor. Puede que la solicitud se realice correctamente si lo intentas de nuevo.
`ZERO_RESULTS`	Indica que no se encontró ninguna ruta entre el origen y el destino.

Es a la hora de lanzar la consulta cuando recojo el valor del "*DirectionsStatus*":

```
directionsService.route(request, function(response, status) {
    if (status == google.maps.DirectionsStatus.OK) {
```

Si todo ha ido bien comienzo a trabajar con los datos. Lo primero que necesito es una variable para poder trabajar con la clase "*DirectionsRenderer*". Esta clase es la encargada de procesar la información de la ruta y mostrarla por pantalla.

Constructor	Descripción
`DirectionsRenderer(` `opts?:`*`DirectionsRendererOptions`*`)`	Crea el procesador con las opciones proporcionadas. Las indicaciones se pueden procesar en un mapa (como superposiciones visuales) o también en un panel `<div>` (como instrucciones en forma de texto).

Esta clase es imprescindible, pues sin ella sería imposible ver resultados.

Métodos de la clase "*DirectionsRenderer*":

Métodos	Valor de retorno	Descripción
`getDirections()`	*DirectionsResult*	Devuelve el conjunto de indicaciones actual del procesador.
`getMap()`	Map	Devuelve el mapa en el que se procesa `DirectionsResult`.
`getPanel()`	Node	Devuelve el panel `<div>` en el que se procesa `DirectionsResult`.
`getRouteIndex()`	number	Devuelve el índice de ruta (basado en cero) actual que el objeto `DirectionsRenderer` utiliza en ese momento.
`setDirections(directions:`*DirectionsResult*`)`	None	Establece el procesador para que utilice el resultado de `DirectionsService`. Si se establece un conjunto de indicaciones válido de esta manera, se mostrarán las indicaciones en el mapa y en el panel especificados en el procesador.
`setMap(map:Map)`	None	Este método indica el mapa en el que se procesarán las indicaciones. Transmite `null` para eliminar las indicaciones del mapa.
`setPanel(panel:Node)`	None	Este método procesa las indicaciones en un objeto `<div>`. Transmite `null` para eliminar el contenido del panel.
`setRouteIndex(routeIndex:number)`	None	Establece el índice (basado en cero) de la ruta del objeto `DirectionsResult` que se va a procesar. De forma predeterminada, se procesa la primera ruta del conjunto.

Es imprescindible utilizar el método "*setMap*" para indicar en qué mapa tiene que mostrar los resultados:

```
directionsDisplay.setMap(map);
```

Así como el método "*setDirections*" ya que nos permite obtener información de la ruta como el destino, la duración o las indicaciones:

```
directionsDisplay.setDirections(response);
```

En el caso de mostrar rutas alternativas a la óptima, también es necesario establecer cuál es el índice de la ruta con la que se está tratando y que se quiere dibujar. Esto lo conseguimos gracias a "*setRouteIndex()*":

```
directionsDisplay2.setRouteIndex(f);
```

Propiedades de la clase "*DirectionsRenderer*":

Propiedades	Tipo	Descripción
directions	*DirectionsResult*	Las direcciones que se mostrarán en un mapa o en un panel , recuperadas como un objeto DirectionsResult de DirectionsService
hideRouteList	boolean	Esta propiedad indica si el procesador debe proporcionar una interfaz de usuario que permita seleccionar diferentes rutas alternativas. De forma predeterminada, este indicador está establecido en "false" y se muestra una lista de rutas que puede seleccionar el usuario en el panel asociado de indicaciones. Para ocultar esta lista, establece hideRouteList en true.
map	Map	Es el mapa en el que se mostrarán las indicaciones.
markerOptions	*MarkerOptions*	Opciones de los marcadores. Todos los marcadores que procesa DirectionsRenderer utilizarán estas opciones.
panel	Node	Elemento en el que se mostrarán los pasos de las indicaciones
polylineOptions	*PolylineOptions*	Opciones de las polilíneas. Todas las polilíneas que procesa DirectionsRenderer utilizarán estas opciones.
preserveViewport	boolean	De forma predeterminada, el mapa de entrada aparece centrado y con un nivel de zoom correspondiente al cuadro delimitador de este conjunto de indicaciones. Si esta opción se establece en true, la ventana gráfica no se modificará, a menos que no se haya establecido nunca ni el centro ni el nivel de zoom del mapa.
routeIndex	number	El índice de la ruta en el objeto DirectionsResult. El valor predeterminado es 0.
suppressBicyclingLayer	boolean	Suprime el procesamiento de BicyclingLayer cuando se solicitan indicaciones para llegar en bici.
suppressInfoWindows	boolean	Suprime el procesamiento de las ventanas de información.
suppressMarkers	boolean	Suprime el procesamiento de marcadores.
suppressPolylines	boolean	Suprime el procesamiento de polilíneas.

Son muchas opciones que podemos modificar para configurar la respuesta en función de lo que desee el usuario. Podemos modificar desde los marcadores, la línea que muestra la ruta, el tipo de capas que queremos que sean visibles así como centrar el mapa en función de los resultados o no.

De estas propiedades he utilizado únicamente dos. La primera es "*suppressMarkers*". Permite eliminar los marcadores que por defecto devuelve la consulta. Es imposible recuperar esos marcadores y trabajar con ellos, por ejemplo si se quieren arrastrar y recalcular la ruta. Por ello, es mejor suprimirlos y crear nuestros propios marcadores:

```
directionsDisplay = new google.maps.DirectionsRenderer({suppressMarkers: true});
```

La otra propiedad que he utilizado ha sido "*PolylineOptions*" que permite configurar valores como el color, el grosor o la opacidad de la línea que va a representar la ruta:

```
directionsDisplay.setOptions({
    polylineOptions: {
        strokeColor: '#00458E',
        strokeWeight: 4,
        strokeOpacity:1
    }
});
```

Estas son todas las opciones que ofrece "*PolylineOptions*":

Propiedades	Tipo	Descripción
clickable	boolean	Indica si este objeto Polyline controla eventos click. El valor predeterminado es true.
geodesic	boolean	Procesa cada extremo como una polilínea geodésica (un segmento de un "gran círculo"). Una polilínea geodésica es la ruta más corta entre dos puntos situados sobre la superficie de la Tierra.
map	Map	Es el mapa en el que se va a mostrar la polilínea.
path	MVCArray.<LatLng>|Array.<LatLng>	Es la secuencia ordenada de coordenadas de la polilínea. Esta ruta se puede especificar mediante un conjunto simple de LatLng o un conjunto MVCArray de LatLng. Recuerda que si transmites un conjunto simple, se convertirá en MVCArray. Si insertas o eliminas LatLng en MVCArray, se actualizará automáticamente la polilínea en el mapa.
strokeColor	string	El color del trazo en código hexadecimal HTML, por ejemplo, "#FFAA00"
strokeOpacity	number	La opacidad del trazo entre 0,0 y 1,0
strokeWeight	number	La anchura del trazo en píxeles
zIndex	number	zIndex comparado con otras polilíneas.

Además de mostrar el resultado gráficamente, también es importante destacar que podemos mostrar los resultados en texto. Podemos obtener todas las indicaciones de cómo llegar de un punto a otro, el tiempo, la distancia y otra información. Para ello necesitamos trabajar con la clase "*DirectionsResult*". Cuando realizamos la consulta a los servidores, la función nos devolvía un "*DirectionsResult*":

Métodos	Valor de retorno	Descripción
`route(request:DirectionsRequest, callback:function(DirectionsResult, DirectionsStatus))`	`None`	Emite una solicitud de búsqueda de indicaciones.

En mi código lo he hecho de la siguiente manera:

```
directionsService.route(request, function(response, status) {
```

Por lo que "*response*" es mi "*DirectionsResult*" del que voy a sacar la información que necesito de la ruta. Un "*DirectionsResult*" es la respuesta de indicaciones en formato JSON recuperada en el servidor de indicaciones. Puedes procesar este objeto mediante un objeto de la clase "*DirectionsRenderer*", o bien analizarlo y procesarlo tú mismo.

El formato JSON, acrónimo de "*JavaScript Object Notation*", es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

Propiedades	Tipo	Descripción
`routes`	`Array.<DirectionsRoute>`	Un conjunto de `DirectionsRoute` en el que cada una de estos elementos contiene información sobre los tramos y pasos que forman dicho conjunto. Solo habrá una ruta a menos que hayas creado la solicitud `DirectionsRequest` con `provideRouteAlternatives` establecido en `true` (esta propiedad antes se denominaba "trips").

Podemos obtener la información de cada ruta. Como en nuestro caso hemos utilizado la opción de ver rutas alternativas, el número de rutas va a varias en función de las rutas alternativas existentes para cada ruta que creamos. Tendremos que recorrer cada una de sus rutas, que a su vez están formadas por piernas, "*legs*", que contienen la información para cada paso de cada ruta.

Las propiedades de "*DirectionsRoute*" son:

Propiedades	Tipo	Descripción
`bounds`	`LatLngBounds`	Son los límites de esta ruta.
`copyrights`	`string`	El texto de los derechos de autor que se mostrará con la ruta.
`legs`	`Array.<DirectionsLeg>`	Un conjunto de `DirectionsLeg`, en el que cada uno de estos elementos contiene información sobre los pasos que forman dicho conjunto. Representa un tramo por cada hito o destino especificado. De este modo, una ruta sin hitos incluirá un `DirectionsLeg`, mientras que una ruta con un hito incluirá dos
`overview_path`	`Array.<LatLng>`	Un conjunto de `LatLng` que representa el recorrido completo de la ruta. La ruta se simplifica para que pueda adaptarse a contextos en los que se necesita un número reducido de vértices
`warnings`	`Array.<string>`	Representa las advertencias que se mostrarán al visualizar estas indicaciones.
`waypoint_order`	`Array.<number>`	Si `optimizeWaypoints` está establecido en `true`, este campo contendrá los cambios reordenados de los hitos de entrada. Por ejemplo, si la entrada es: Origen: Los Ángeles Hitos: Dallas, Bangor, Phoenix Destino: Nueva York y la salida optimizada se ordena de la siguiente forma: Origen: Los Ángeles Hitos: Phoenix, Dallas, Bangor Destino: Nueva York este campo será un `Array` con los valores [2, 0, 1]. Ten en cuenta que el número de hitos se basa en cero. Si cualquiera de los hitos de entrada tiene `stopover` establecido en `false`, este campo estará vacío, ya que el proceso de optimización de rutas no está disponible para ese tipo de consultas.

El campo "*legs*" es muy importante para poder trabajar con cada ruta y sacar información importante de ella. Un "*DirectionsLeg* " es un único tramo compuesto de un conjunto de pasos en formato JSON en un "*DirectionsResult*". Puede que no se devuelvan todos los campos del tramo en todas las solicitudes.

Propiedades del "*DirectionsLeg*":

Propiedades	Tipo	Descripción
distance	*DirectionsDistance*	Es la distancia total que cubre este tramo. Puede que esta propiedad quede sin definir si se desconoce la distancia.
duration	*DirectionsDuration*	Es la duración total de este tramo. Puede que esta propiedad quede sin definir si se desconoce la duración.
end_address	string	Es la dirección de destino de este tramo.
end_location	LatLng	DirectionsService calcula las indicaciones entre ubicaciones con la opción de transporte más cercana (normalmente una carretera) a las ubicaciones de salida y de llegada. end_location indica el destino real codificado geográficamente, que puede ser diferente al parámetro end_location del último paso si, por ejemplo, la carretera no está cerca del destino de este tramo.
start_address	string	Es la dirección de origen de este tramo.
start_location	LatLng	DirectionsService calcula las indicaciones entre ubicaciones con la opción de transporte más cercana (normalmente una carretera) a las ubicaciones de salida y de llegada. start_location indica el origen real codificado geográficamente, que puede ser diferente al parámetro start_location del primer paso si, por ejemplo, la carretera no está cerca del origen de este tramo.
steps	Array.<*DirectionsStep*>	Un conjunto de DirectionsStep, en el que cada uno de estos elementos incluye información sobre cada uno de los pasos de este tramo

La propiedad más importante de la clase "*DirectionsLeg*" son los pasos o "*steps*" ya que nos permiten obtener toda la información importante y necesaria de la ruta. Gracias a los pasos podemos obtener la siguiente información de la ruta:

Propiedades	Tipo	Descripción
distance	*DirectionsDistance*	Es la distancia que cubre este paso. Puede que esta propiedad quede sin definir si se desconoce la distancia.
duration	*DirectionsDuration*	El tiempo que habitualmente es necesario para realizar este paso, expresado en segundos y en formato de texto. Puede que esta propiedad quede sin definir si se desconoce la duración.
end_location	LatLng	La ubicación de llegada de este paso
instructions	string	Las instrucciones para este paso
path	Array.<LatLng>	Una secuencia de LatLng que describe todo el recorrido de este paso
start_location	LatLng	La ubicación de salida de este paso

Estas serían, aproximadamente, la mayoría de clases con las que he trabajado para poder desarrollar mi proyecto. He necesitado saber perfectamente cómo programar en JavaScript y después leerme el API para obtener las clases que necesitaba.

Existen muchas más propiedades y clases que he utilizado pero que no son tan relevantes como las que he explicado. Ahora que ya tengo las herramientas definidas, explicaré, a lo largo de este documento, algunas funciones que he implementado que me resultan más peculiares o curiosas y que las he realizado gracias a todas estas clases.

5. REQUISITOS

Fundamentalmente, el requisito principal del proyecto común es encontrar nuevos usos y funcionalidades para los sistemas de navegación. Estudiar lo existente para poder ofrecer mejoras útiles a los usuarios.

Es importante encontrar utilidades para los usuarios, que sean fáciles de lanzar y de entender su funcionamiento. Los resultados obtenidos podrán ser útiles tanto como para usuarios comunes como para desarrolladores de software que buscan mejoras en los sistemas de navegación.

En nuestra aplicación tenemos 3 tipos de actores:

- Analista de la aplicación
- Usuario
- Usuarios expertos

El rol del analista es el principal en este proyecto ya que a pesar de que vamos a desarrollar una aplicación, la actividad principal va a ser la investigación de su funcionamiento. Serán los encargados de tener una visión completa del sistema, conociendo todas sus funciones y sobre todo, cómo deben implementarse y qué es necesario para ello.

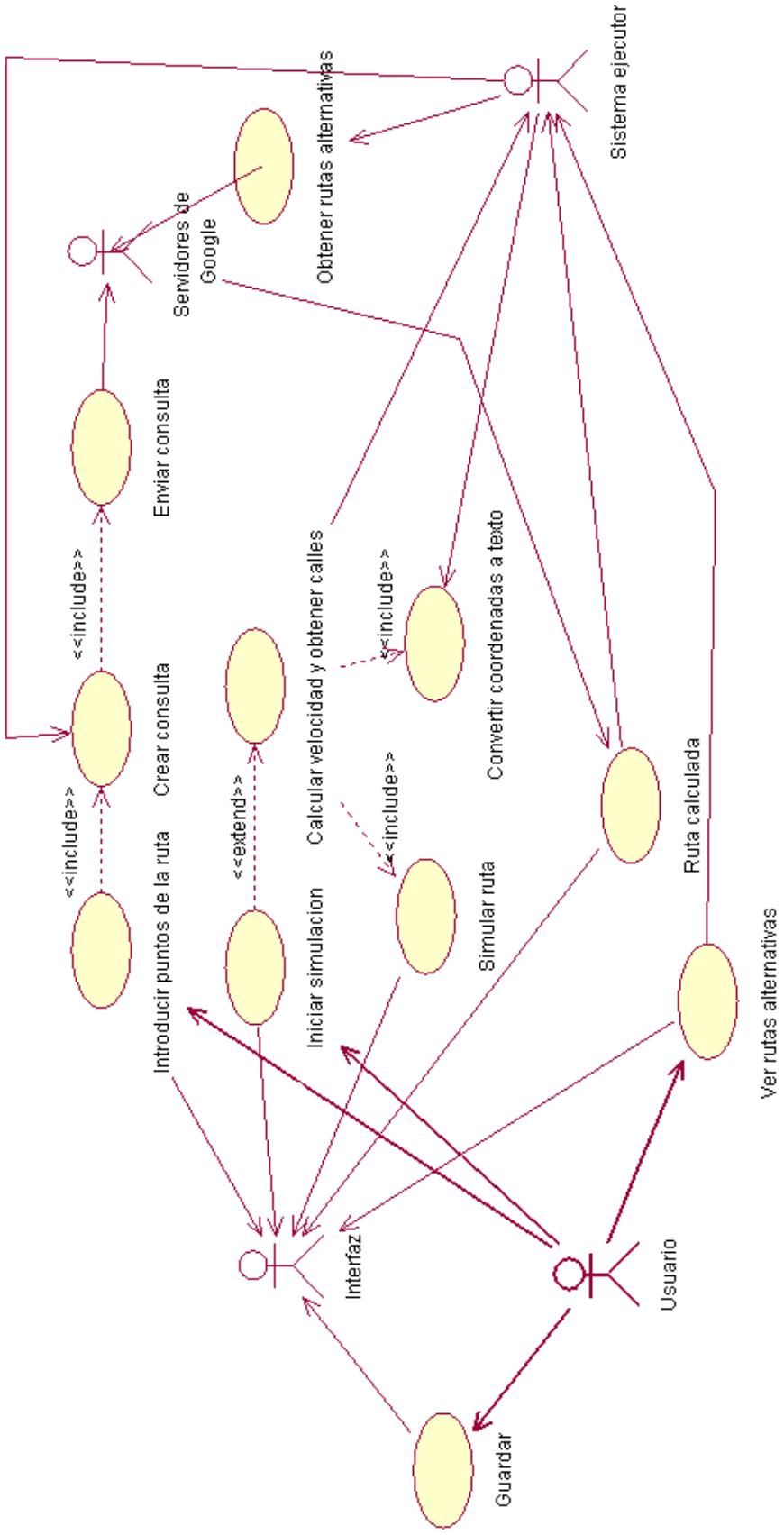
El rol de usuario puede ser desempeñado por cualquier persona dispuesta a probar la aplicación que se ha creado para comprobar el funcionamiento de toda la investigación realizada por el analista. No es necesario que tenga ningún tipo de conocimiento acerca de la aplicación ni qué se pretende conseguir con ella. De hecho, el propio analista puede desempeñar esta función ya que además, podrá comprobar con mayor detalle todo lo que ha investigado anteriormente.

Por último, el rol de usuarios expertos puede ser cualquier desarrollador que quiera introducir diferentes funcionalidades a un servicio de navegación que trabaje con el API de Google Maps y que crea que le puedan ser útiles las funcionalidades que nosotros hemos añadido.

Es un requisito el desarrollar una interfaz clara y sencilla desde la que cualquier usuario pueda realizar sus rutas de manera personalizada sin que se sienta confuso con las nuevas opciones que ofrecemos. Del mismo modo, el software que desarrollamos también debe ser claro para que cualquier desarrollador sepa rápidamente cuál es el apartado que le interesa reutilizar de todos los que creamos.

A continuación exponemos los diagramas de caso de uso y de secuencia para facilitar la comprensión de qué debemos hacer y quiénes van a interactuar en cada acción.

5.1 DIAGRAMA DE CASOS DE USO



En el diagrama de casos de uso vemos que el usuario puede realizar cuatro funciones básicas:

- Introducir puntos de la ruta
- Iniciar simulación
- Ver rutas alternativas
- Guardar

El usuario introduce los puntos de la ruta en la interfaz, la cual pasa los puntos al sistema ejecutor quien a su vez reorganiza estos puntos y crea la estructura adecuada para enviar estos puntos a los servidores de Google Maps. Los servidores realizan el cálculo de la ruta y obtienen un resultado que envían al sistema ejecutor. Éste recoge los puntos y se los envía, de manera ordenada, a la interfaz, quien realiza la organización de los puntos para que sean entendibles por el usuario.

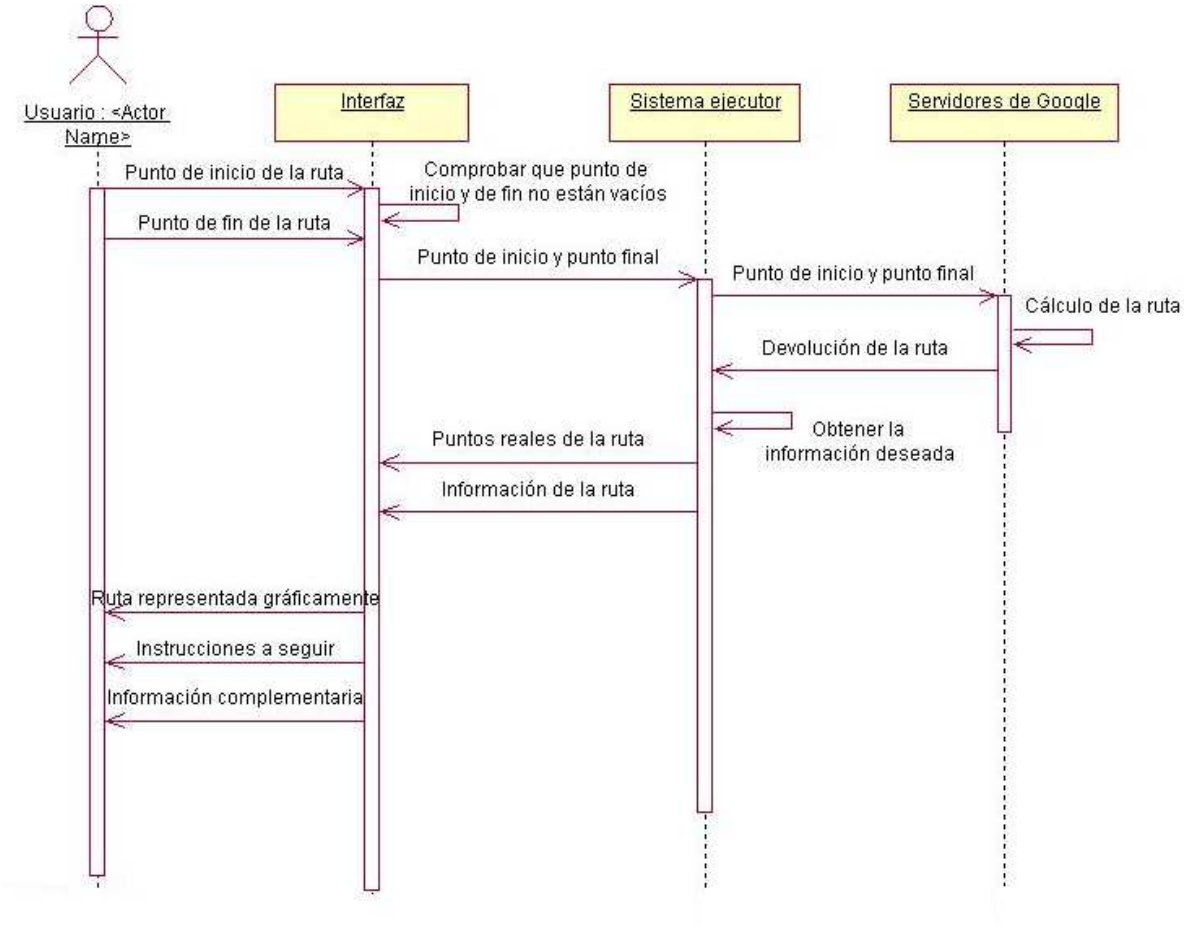
Cuando el usuario decide iniciar la simulación, la interfaz notifica la acción al sistema ejecutor quien calcula la velocidad en la que se va a ir mostrando la simulación y la información que se tiene que mostrar en cada punto. En este evento no actúan los servidores de Google Maps.

Si se selecciona ver rutas alternativas, la interfaz notifica al sistema ejecutor de esta acción y el sistema ejecutor ordena a los servidores que calculen las rutas alternativas para esos puntos. Cuando ya han calculado todas las rutas, el sistema ejecutor recoge las rutas alternativas y las pasa a la interfaz para que las muestre en el mapa y en el panel de informaciones de cada ruta.

Por último, si el usuario decide guardar la ruta, la interfaz creará una nueva ventana en la que mostrará el mapa con la ruta representada gráficamente y las instrucciones de cómo realizar el trayecto.

5.2 DIAGRAMAS DE SECUENCIA

- Diagrama de cómo calcular la ruta básica:



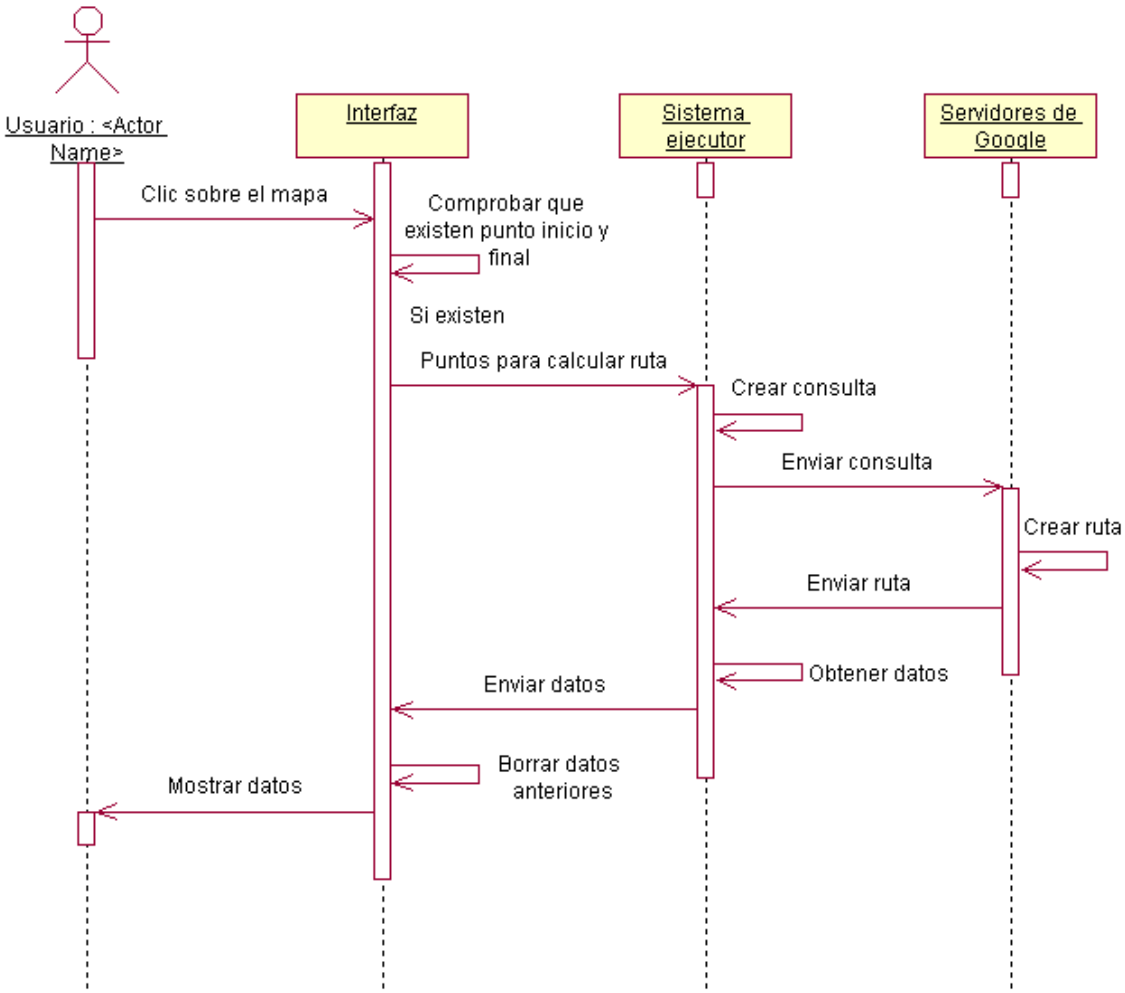
Cuando el usuario inicia nuestro sistema de navegación, el mapa se centra sobre la ciudad en la que se encuentra y el usuario debe introducir, en los dos espacios dedicados para ello, la cadena de texto con la dirección desde la que quiere salir y otra con la dirección a la que quiere llegar.

Es el usuario quien inicia la acción al introducir el punto inicial y final ya que la interfaz recibe la petición y la pasa al sistema ejecutor. El sistema ejecutor debe crear la consulta de acuerdo a las clases del API para que los servidores de Google puedan comprender la consulta y la envía a los servidores.

Cuando los servidores reciben la consulta, calculan la ruta óptima que une esos dos puntos y la devuelve al sistema ejecutor. Éste, se encarga de pasarle a la interfaz los datos que se quieren mostrar y la interfaz devuelve al usuario tanto una representación gráfica en el mapa como un texto explicativo con las indicaciones a seguir.

Sistema ejecutor de un navegador personalizado

- Diagrama de cómo introducir puntos de paso en la ruta:



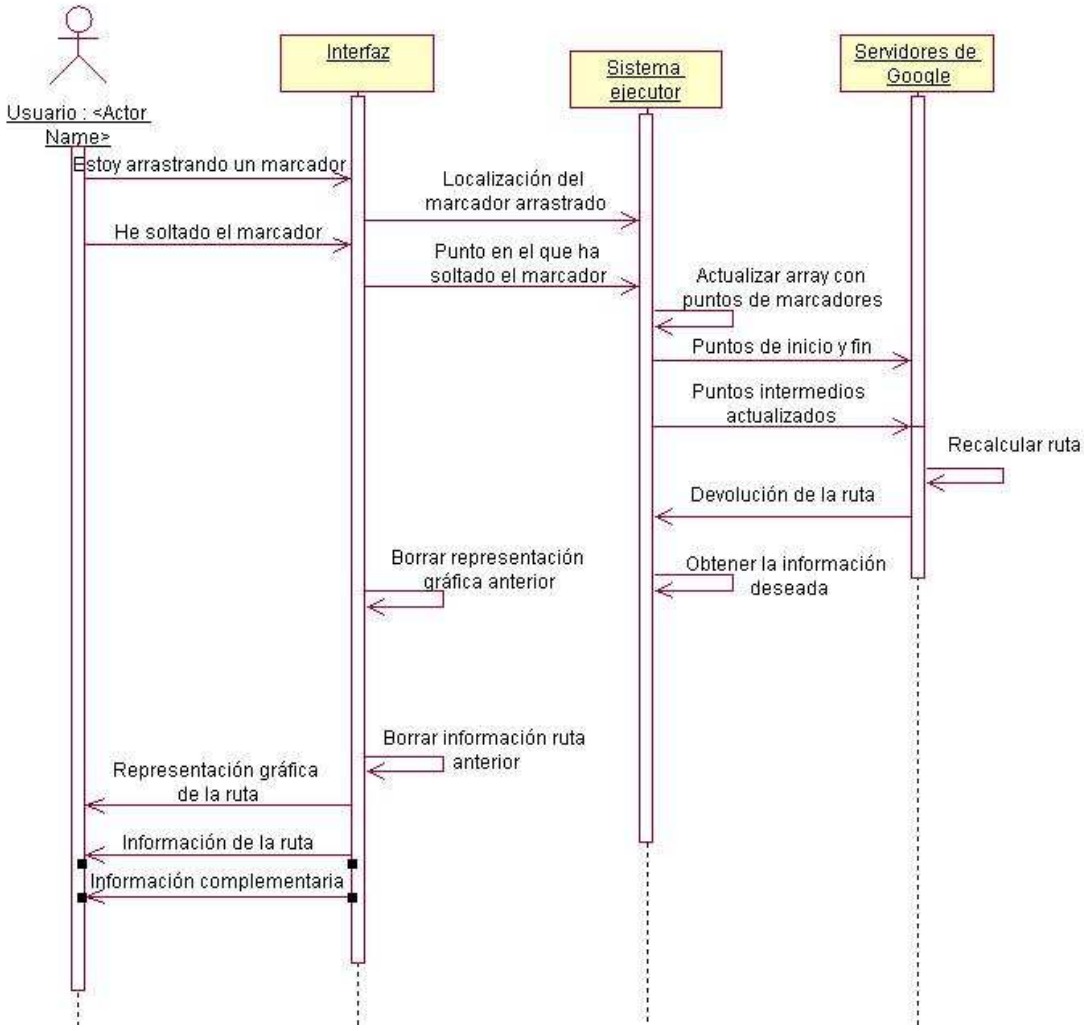
Cuando el usuario desea introducir etapas en el recorrido óptimo obtenido nada más introducir el punto de inicio y final, debe hacer clic en el mapa sobre el punto por el que desea pasar. Al centrar el sistema de navegación en ciudades, es fácil que el usuario quiera desviar su camino para hacer algún tipo de parada en el camino.

Cuando el usuario hace clic, la interfaz recoge el punto sobre el que ha clicado, y se lo envía al sistema ejecutor junto con los puntos iniciales y finales. El sistema ejecutor, a su vez, envía a los servidores de Google la nueva consulta que esta vez incorpora el punto de paso seleccionado.

Una vez que los servidores hayan recalculado la ruta, el sistema ejecutor recoge la ruta y envía la información a la interfaz para que la haga visible al usuario.

Sistema ejecutor de un navegador personalizado

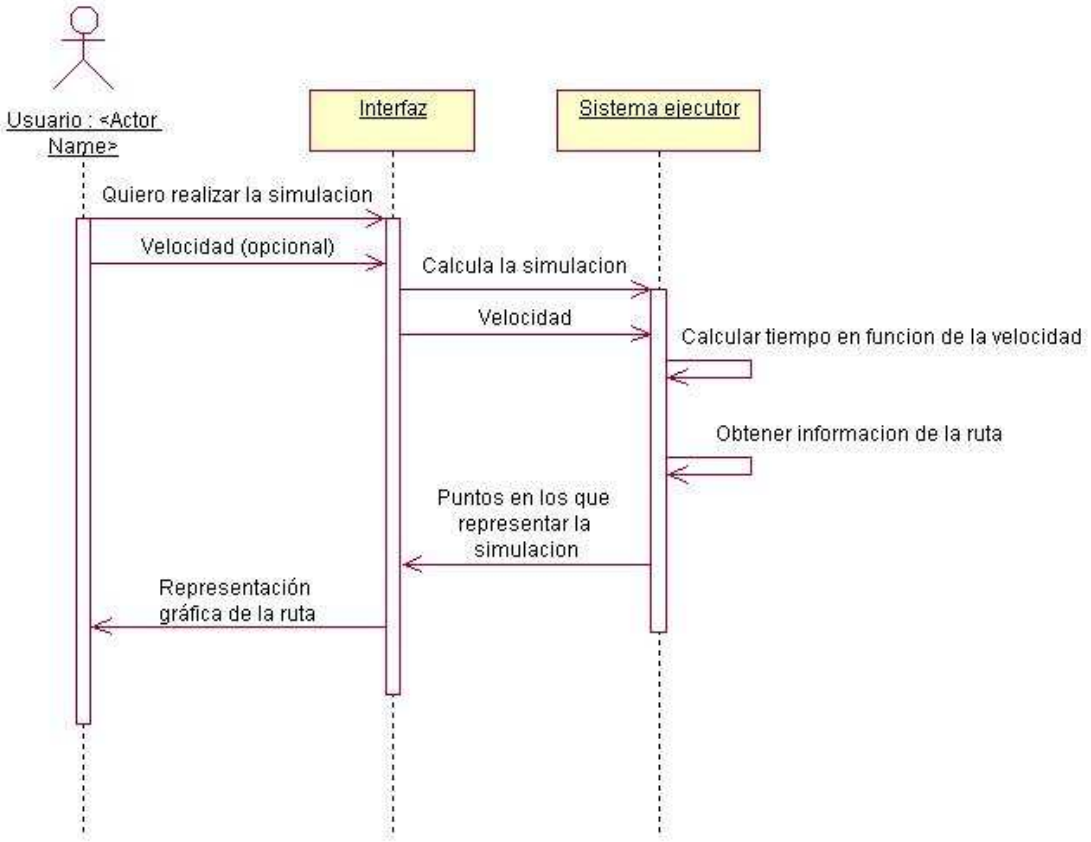
- Diagrama de cuando el usuario arrastra un marcador del mapa para recalcular la ruta:



Cuando el usuario hace clic sobre el mapa, está indicando que quiere añadir etapas, o puntos de paso, en la ruta que se ha creado. Esto lo puede hacer bien para evitar pasar por una calle que ha mostrado la ruta o también para indicar que quiere pasar por un punto concreto que no está en el resultado. Una vez creada la etapa, se señalizan con marcadores que se pueden arrastrar. Estos marcadores señalizan un punto que se ha pasado a los servidores para que recalculen la ruta. Si el usuario se ha equivocado o simplemente no quiere pasar por ese punto sino por otro, puede arrastrar el marcador y la ruta ya no pasará por el punto anterior sino que ahora pasará por el punto en el que se ha soltado el marcador que se ha arrastrado.

Este nuevo punto es recogido por la interfaz mediante "*listeners*" y lo pasa al sistema ejecutor. Éste debe ser capaz de eliminar el punto en el que se encontraba inicialmente el marcador e introducir el nuevo en la consulta que envía a los servidores de Google. Cuando se crea la nueva consulta, la interfaz borra el marcador anterior y crea uno nuevo en el nuevo punto. También borra la ruta anterior pintada sobre el mapa y pinta la nueva.

- Diagrama de cómo simular la ruta:

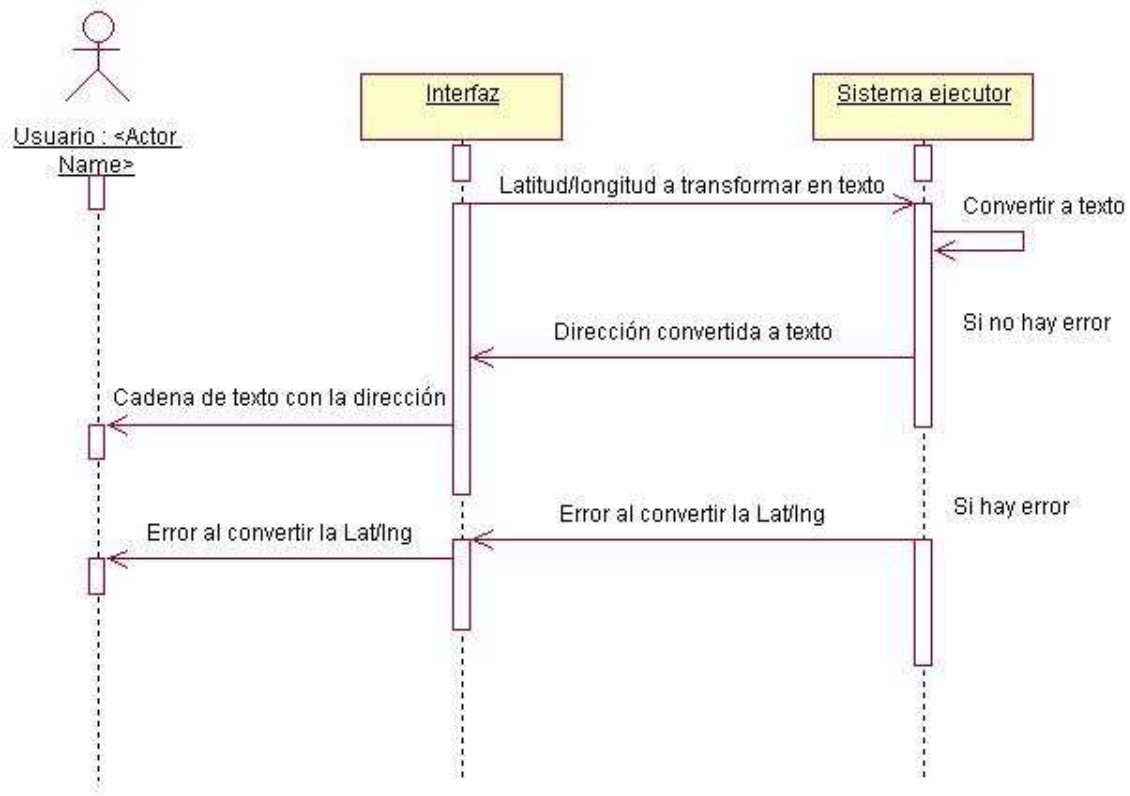


Hemos introducido la posibilidad de ver una representación gráfica del trayecto a seguir en tiempo real. La interfaz notifica al sistema ejecutor el deseo de llevar a cabo una representación y, si el usuario ha introducido una velocidad a la que realizarla, también le pasa la velocidad.

El sistema ejecutor analiza todos los puntos que tiene para esa ruta, son puntos internos para los cuales existe una posición, en formato Latitud/Longitud, y para algunos de ellos también existen unas instrucciones (por ejemplo gire a la derecha, o salga por la tercera salida). Con toda esta información, se crea una serie de tablas para que la interfaz pueda mostrar una imagen en el punto real con su indicación concreta para ese punto.

Con respecto a la velocidad, si el usuario introduce una velocidad en km/h el sistema ejecutor divide la distancia total entre el número de puntos y lo relaciona con la velocidad introducida, para así calcular el nuevo tiempo necesario para realizar la ruta. En caso de no introducir ninguna velocidad, el sistema ejecutor realizará las mismas acciones pero con la velocidad utilizada por defecto por los servidores.

- Diagrama de cómo convertir una dirección en formato Latitud/Longitud a cadena de texto para que sea comprensible por los usuarios:

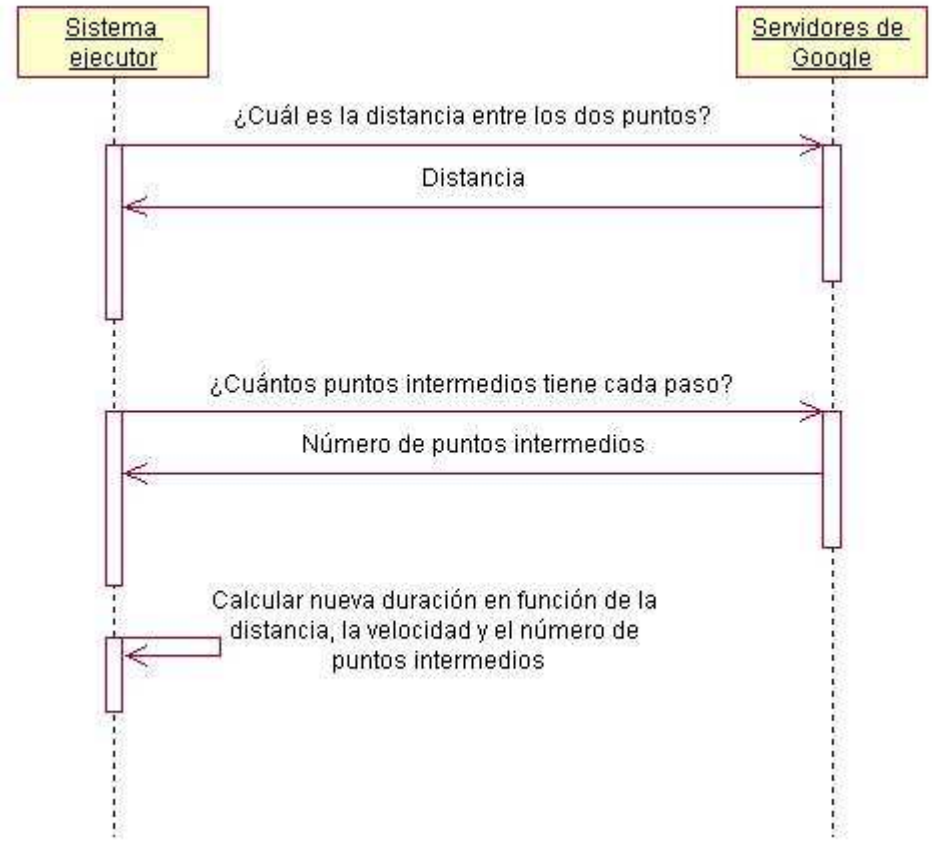


Hay ocasiones en las que el usuario le pasa a la interfaz una posición en formato Latitud/Longitud. Esto es por ejemplo cuando hace clic sobre el mapa y se crea un marcador. Después, si el usuario quiere ver cuál es la dirección sobre la que ha hecho clic, es necesario que el sistema convierta la dirección a formato cadena de texto.

Para realizar esta acción no es necesaria la ayuda de los servidores sino que la realizamos gracias a una función que se encarga de convertir a cadenas de texto posiciones geodésicas.

Es posible que no existe una dirección exacta para el punto sobre el que se ha hecho clic así que o bien muestra un mensaje de error o bien muestra una aproximación mostrando únicamente el código postal de la zona y la ciudad.

- Diagrama que muestra cómo obtener la duración en función de la velocidad

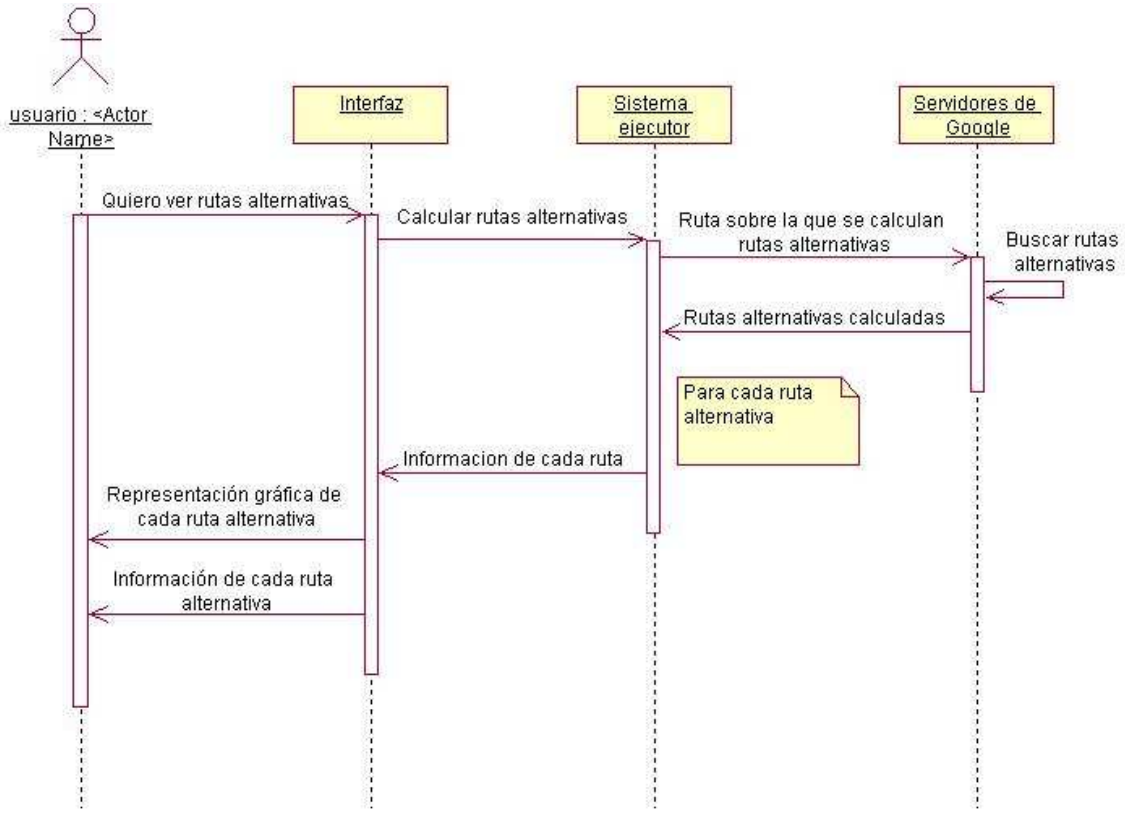


Cuando el sistema ejecutor recibe de la interfaz la petición de realizar una simulación, el sistema ejecutor debe pedir a los servidores de Google la distancia de la ruta. También es necesario que le solicite una serie de puntos intermedios, que los servidores de Google tienen almacenados, para poder ir dibujando un icono gráfico en estos puntos.

Los puntos intermedios pueden ser simples puntos o también hemos conseguido poner localización a los puntos de información (por ejemplo “en la rotonda salga por la 1º salida” o “gire a la derecha…”, etc.).

Una vez que tiene todos los datos, debe ver si el usuario ha introducido o no velocidad. En caso de haberla introducido debe realizar una regla de tres en función a la velocidad. En caso de no haber introducido velocidad, lo calculará automáticamente con el tiempo que por defecto tiene la ruta.

Diagrama de alternativas



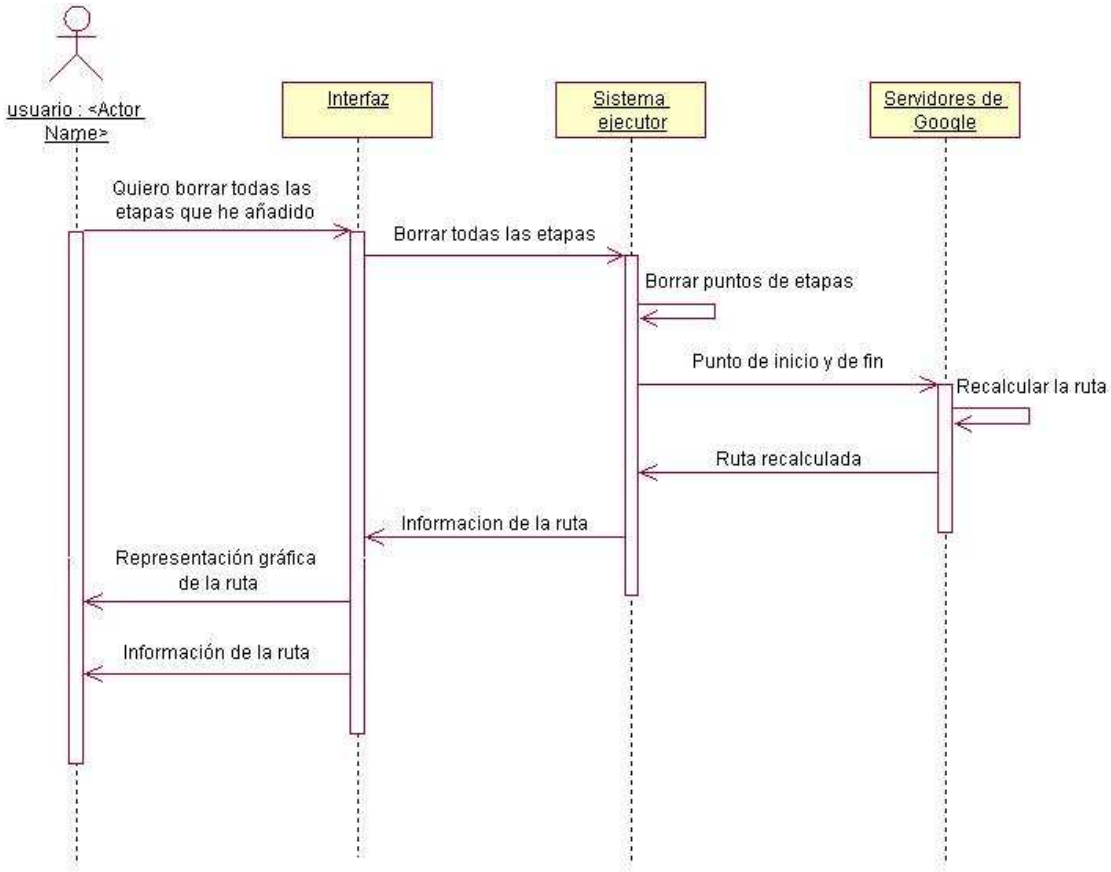
Es posible que el usuario quiera ver rutas alternativas a la ruta óptima que se ha obtenido. Bien porque sabe que esa ruta a la hora que va a pasar hay mucho tráfico o bien porque simplemente quiere ver nuevas rutas.

La interfaz notifica al sistema ejecutor que se quieren ver las alternativas a la ruta y el sistema ejecutor solicita a los servidores de Google que calculen estas rutas.

Cuando el sistema ejecutor recibe las rutas alternativas debe recorrer una a una cada ruta para sacar la información de cada una e ir pasándoselas a la interfaz.

La interfaz a su vez, mostrará de manera gráfica en el mapa las nuevas rutas. Lo hemos creado para que las muestre en un color diferente a la óptima para que el usuario pueda diferenciarlas de manera sencilla. Además, creará una nueva ventana con información sobre las rutas alternativas para indicar cómo y por dónde se debe ir así como el tiempo y la distancia de las nuevas rutas.

Diagrama de cómo limpiar todos los puntos intermedios de la ruta:

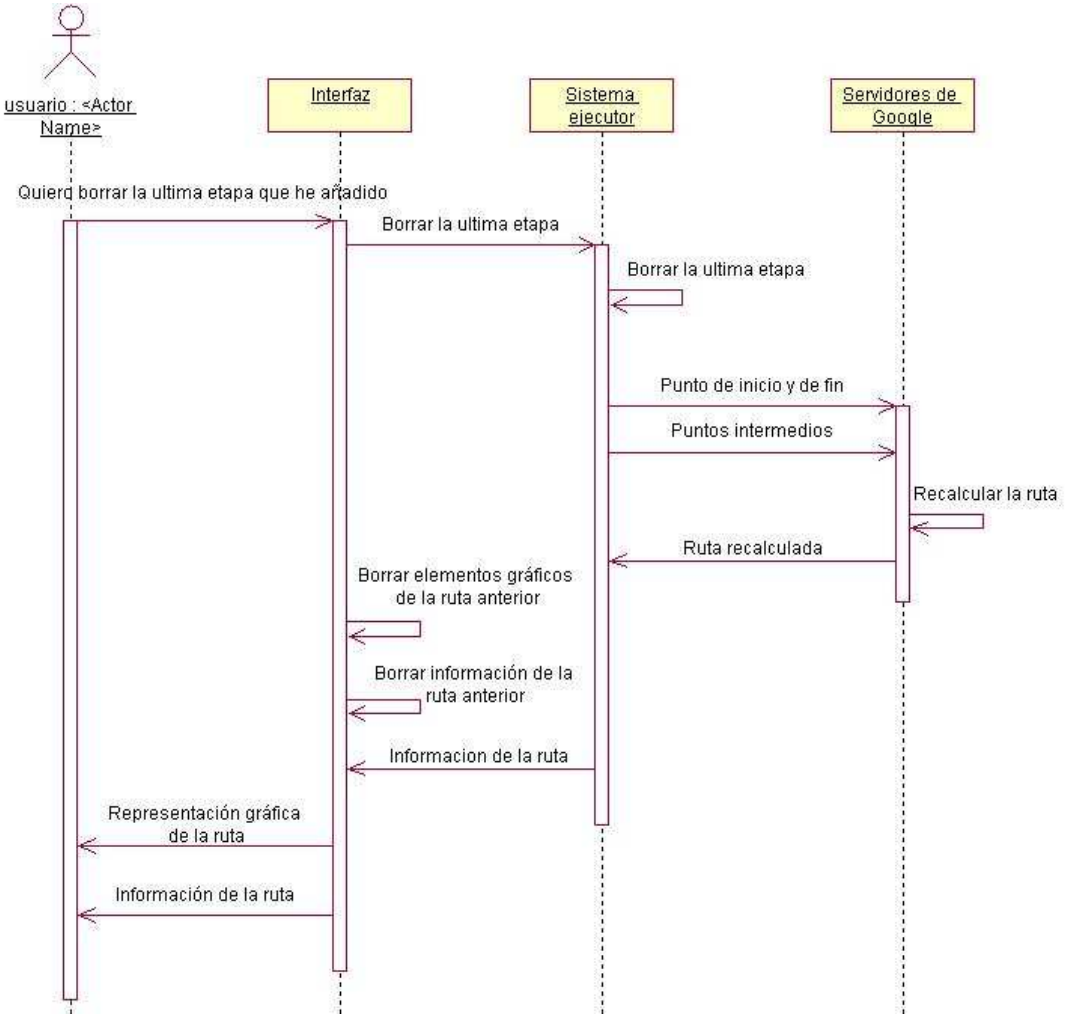


Es posible que el usuario haya introducido muchos puntos intermedios entre el origen y destino de una ruta. La introducción de los puntos intermedios la hemos explicado en uno de los diagramas anteriores, por lo que ya sabemos que a medida que ha hecho clic sobre el mapa se han ido añadiendo marcadores y con ello la ruta se ha ido modificando.

Puede darse el caso en el que el usuario haya introducido demasiados puntos o que al final decida pasar únicamente por la ruta óptima, por lo que quiere borrar todos los puntos que ha introducido. Esto se puede hacer de uno en uno o bien con el botón "*Limpiar ruta*" que borra todos los marcadores de vez y vuelve a calcular la ruta óptima.

Para ello, el sistema ejecutor vuelve a solicitar a los servidores de Google la ruta óptima pasándole únicamente el punto de inicio y de fin. Los servidores recalculan la ruta y la interfaz borra todos los resultados anteriores para mostrar los nuevos.

- Diagrama explicativo de cómo eliminar un único punto intermedio



A medida que el usuario ha ido haciendo clic sobre el mapa, se han creado marcadores y se ha recalculado la ruta. Estos puntos intermedios los hemos tenido que almacenar en un array que el sistema ejecutor le pasa a los servidores junto con el punto de inicio y de fin para que calculen la nueva ruta con las etapas.

Si el usuario se ha equivocado y ha introducido un punto que no quería introducir, puede eliminarlo pulsando el botón "*Eliminar punto*". La interfaz se encarga de eliminar el punto del mapa y del array para enviárselo de nuevo al sistema ejecutor para que éste ordene a los servidores que recalculen la ruta.

Una vez recalculada la ruta, el sistema ejecutor envía la información a la interfaz. La interfaz se encarga de borrar los datos anteriores relativos al punto que se ha eliminado y repinta la ruta otra vez sobre el mapa.

6. ANALISIS Y DISEÑO

En este apartado explicaré el proceso necesitado para averiguar cómo funciona un sistema de navegación y el proceso seguido para conseguir realizar mi proyecto.

En primer lugar comencé analizando qué sistemas existían ya. Como he explicado anteriormente, al final decidí trabajar con Google Maps ya que es el único que me ofrecía un API para poder desarrollar mi propio sistema de navegación personalizado. La siguiente cuestión que me planteé era averiguar cómo funcionaba realmente Google Maps.

Gracias al API yo puedo conectarme a los servidores de Google y obtener toda la información deseada para mostrarla al usuario. Ahora bien, mi pregunta inicial fue si yo tenía que saber calcular la ruta o cómo se calculaba realmente. Entonces descubrí que los "servidores de Google" de los que he hablado en todo el proyecto no es más que un Sistema de Información Geográfico (SIG), que se encarga de calcular las rutas. Yo no me encargo de desarrollar un SIG sino de enviarle la información y después recogerla.

Además, al obtener la información, he descubierto que la clave es saber trabajar con las piernas de la ruta y de cada pierna, con sus pasos ya que son quienes contienen la información necesaria como son las indicaciones, duración, tiempo y localización en el mapa.

Otro punto que trato en este apartado es la interacción entre lo que debo hacer y las clases con las que lo he hecho. Para que sea más comprensible me he ayudado de los diagramas de bloque para facilitar mi explicación.

6.1 CÓMO FUNCIONA GOOGLE MAPS

Al ingresar en Google Maps (http://maps.google.es/maps), nos encontramos con un globo terráqueo en forma de mapa bidimensional**,** donde se utilizan proyecciones para encontrar la información que se desea encontrar. Así pues encontraremos las rutas y direcciones de cómo llegar a lugares específicos.

Es un servicio gratuito disponible en Internet y al que se puede acceder no sólo desde ordenadores sino también desde teléfonos móviles.

Lo más resaltante del Google Maps, es que da la oportunidad de proporcionarnos diferentes vistas y proyecciones de los mapas**,** por ejemplo uno de los más destacables es el de Proyección de Mercator, un sistema de coordenadas internacional.

La proyección de Mercator es un tipo de proyección cartográfica cilíndrica, ideada por Gerardus Mercator en 1569, para elaborar planos terrestres. Es muy utilizada en planos de navegación por la facilidad de trazar rutas de rumbo constante.

Mercator, mediante proyección, pretende representar la superficie esférica terrestre sobre una superficie cilíndrica, tangente al ecuador, que al desplegarse genera un mapa terrestre plano. Es un modelo idealizado que trata a la Tierra como un globo hinchable que se introduce en un cilindro y que empieza a «inflarse» ocupando el volumen del cilindro, imprimiendo el mapa en su cara exterior. Este cilindro cortado longitudinalmente y desplegado sería parecido al mapa con la proyección de Mercator.

Conseguir las imágenes de Google Maps es un procedimiento muy sencillo. Las imágenes se toman por el satélite de teledetección QuickBird, ubicado a 465 kilómetros aproximadamente sobre la superficie terrestre, en el espacio. Este satélite viene ya funcionando desde el año 2001. Vale la pena destacar que no todas las imágenes que podemos observar provienen de un satélite, muchas de ellas fueran capturadas gracias a aviones que sobrevuelan a más de 10.000 metros de altura**.**

Pero no todos son halagos pues existen zonas del mundo que no son accesibles, debido a temas de protección de algunos países que suele oscurecer las áreas en que se ubican los más importantes edificios políticos, áreas militares y gubernamentales.

6.2 GOOGLE MAPS: FUNCIONAMIENTO INTERNO [1, 3, 4, 5, 6]

Detrás de las imágenes de Google Maps existe una tecnología que se describe como un servidor de mapas. El servidor de mapas genera un único mapa de la ubicación solicitada a partir de un gran conjunto de imágenes generadas como baldosas de 256x256 que cubren todo el planeta.

Estas baldosas de mapas se obtienen tras la obtención de la ruta en un Sistema de Información Geográfico (SIG).

Un Sistema de Información Geográfica es una integración de hardware, software y datos geográficos diseñada para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión. También puede definirse como un modelo de una parte de la realidad referido a un sistema de coordenadas terrestre y construido para satisfacer unas necesidades concretas de información. En el sentido más estricto, es cualquier sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográficamente referenciada. En un sentido más genérico, los SIG son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones.

El SIG funciona como una base de datos con información geográfica (datos alfanuméricos) que se encuentra asociada por un identificador común a los objetos gráficos de un mapa digital. De esta forma, señalando un objeto se conocen sus atributos e, inversamente, preguntando por un registro de la base de datos se puede saber su localización en la cartografía.

La razón fundamental para utilizar un SIG es la gestión de información espacial. El sistema permite separar la información en diferentes capas temáticas y las almacena independientemente, permitiendo trabajar con ellas de manera rápida y sencilla, facilitando al profesional la posibilidad de relacionar la información existente a través de la topología de los objetos, con el fin de generar otra nueva que no podríamos obtener de otra forma.

Las principales cuestiones que puede resolver un Sistema de Información Geográfica, ordenadas de menor a mayor complejidad, son:

1. **Localización**: preguntar por las características de un lugar concreto.
2. **Condición**: el cumplimiento o no de unas condiciones impuestas al sistema.
3. **Tendencia**: comparación entre situaciones temporales o espaciales distintas de alguna característica.
4. **Rutas**: cálculo de rutas óptimas entre dos o más puntos.
5. **Pautas**: detección de pautas espaciales.
6. **Modelos**: generación de modelos a partir de fenómenos o actuaciones simuladas.

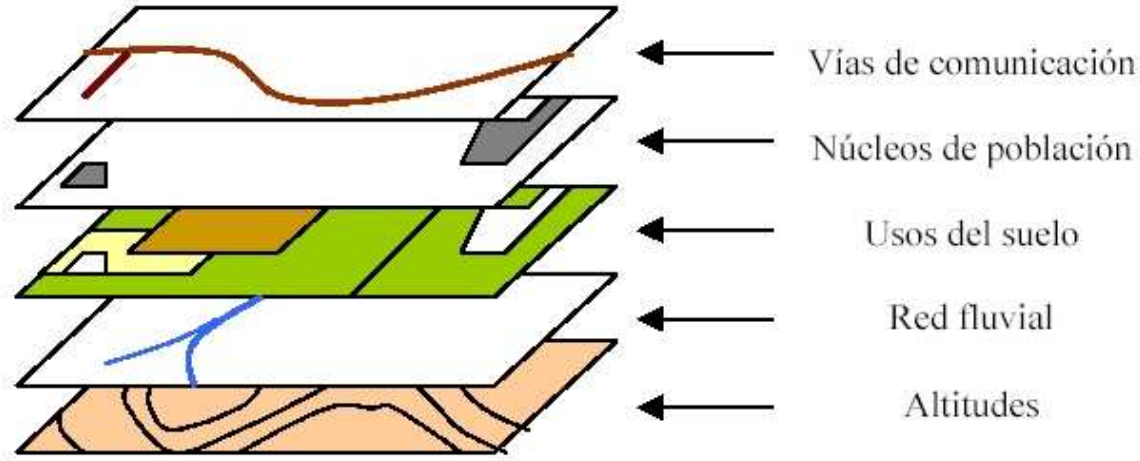


Imagen 10: Capas en las que divide un SIG el mundo real.

Un sistema de información geográfica, es una herramienta de análisis de información. La información debe tener una referencia espacial y debe conservar una inteligencia propia sobre la topología y representación.

En general un SIG debe tener la capacidad de dar respuesta a las siguientes preguntas:

- ¿Dónde está el objeto A?
- ¿Dónde está A con relación a B?
- ¿Cuantas ocurrencias del tipo A hay en una distancia D de B?
- ¿Cuál es el valor que toma la función Z en la posición X?
- ¿Cuál es la dimensión de B (Frecuencia, perímetro, área, volumen)?
- ¿Cuál es el resultado de la intersección de diferentes tipos de información?
- ¿Cuál es el camino más corto (menor resistencia o menor costo) sobre el terreno desde un punto (X1, Y1) a lo largo de un corredor P hasta un punto (X2, Y2)?
- ¿Qué hay en el punto (X, Y)?
- ¿Qué objetos están próximos a aquellos objetos que tienen una combinación de características?
- ¿Cuál es el resultado de clasificar los siguientes conjuntos de información espacial?
- Utilizando el modelo definido del mundo real, simule el efecto del proceso P en un tiempo T dado un escenario S.

Nosotros nos tenemos que estudiar cómo funciona un sistema de información geográfica, sino que tenemos que saber cómo pasarle parámetros y como los devuelve para poder conseguir las rutas. Cuando a lo largo del proyecto hago referencia a los “servidores de Google Maps”, me refiero al sistema de información geográfica que emplea Google.

6.3 CÓMO OBTENER INFORMACIÓN DE LA RUTA

En este apartado voy a explicar lo más importante para que mi proyecto pueda funcionar, cómo obtener la información de la ruta. Explicaré cómo está estructurada una ruta y qué pasos tengo que seguir para trabajar con ella.

6.3.1 QUÉ ES UNA RUTA

Una ruta es el trayecto completo para llegar de un punto de inicio al punto final. Es independiente del número de hitos o etapas porque una ruta las engloba.
Por lo tanto, una ruta podría ser una ruta sin hitos:

Imagen 11: Imagen que muestra la ruta, en color azul, sobre el mapa

O también una ruta con hitos o etapas que ha introducido el usuario:

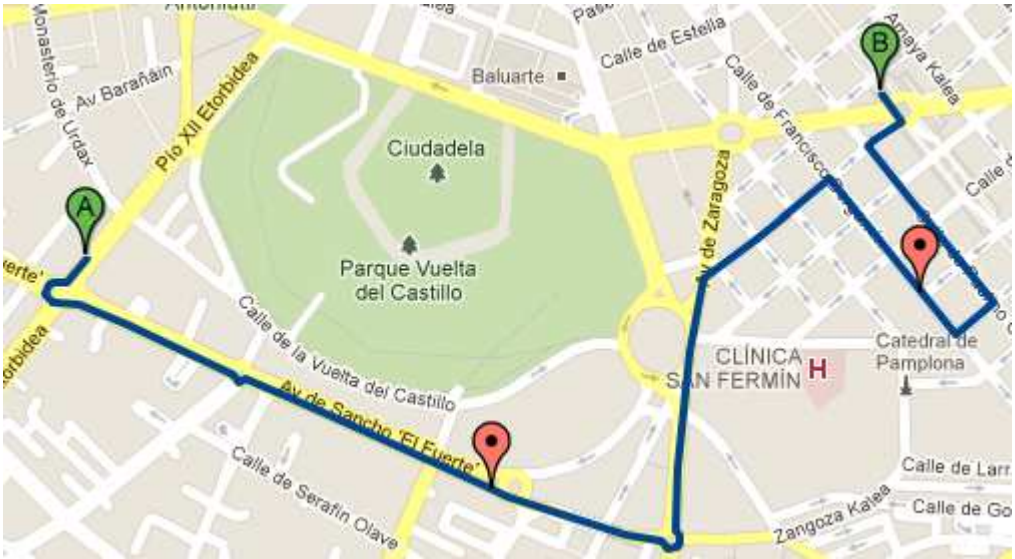


Imagen 12: Imagen que muestra una ruta con dos etapas

6.3.2 QUÉ ES UNA PIERNA

Una pierna, o "*leg*", es un conjunto de tramos de una ruta. Si por ejemplo tenemos una etapa en la ruta, el número de piernas será dos. Una desde el inicio hasta la etapa marcada y la otra pierna desde la etapa hasta el punto final de la ruta.

Imagen 13: Imagen que señaliza cada pierna de una ruta

En caso de que la ruta no tenga ninguna etapa, el número de piernas será 1 e irá desde el punto de inicio hasta el punto final.

El número de piernas siempre es igual al número de etapas +1.

Las piernas son de la clase "*DirectionsLeg*" y de cada una de ellas se puede obtener:

- Distancia
- Duración
- Dirección en la que inicia la pierna (cadena de texto)
- Localización en la que inicia la pierna (tipo Latitud/Longitud)
- Dirección en la que termina la pierna (cadena de texto)
- Localización en la que termina la pierna (tipo Latitud/Longitud)
- Pasos

En el proyecto recorro todas las piernas de la siguiente manera:

```
for (var i = 0; i < route.legs.length; i++) {
```

Es imprescindible que recorra cada pierna para poder obtener el punto en el que la interfaz debe pintar los marcadores intermedios así como para obtener la información de cada pierna.

6.3.3 QUÉ SON LOS PASOS

Cada pierna de cada ruta está formada por pasos. Incluyen la información sobre cada uno de los pasos del tramo. Para trabajar con los pasos tenemos que trabajar con la clase "*DirectionStep*" y de cada paso podemos obtener la siguiente información:

- Distancia: es la distancia que cubre este paso
- Duración: es el tiempo necesario para realizar este paso
- Localización de inicio: punto de inicio del paso en cuestión
- Localización de fin: punto de fin del paso
- Instrucciones: Indicaciones para los usuarios de qué hacer en ese paso.
- Path: secuencia de Latitud/Longitud que describe todo el recorrido de ese paso.

Cuando el usuario decide ir de un punto a otro mostramos las indicaciones de cómo realizar el trayecto. Estas indicaciones las obtenemos del campo "*instrucciones*" que acabo de comentar. Por lo tanto, lo que mostramos es cómo realizar un paso de una pierna determinada contenida en la ruta final.

Además, cuando el usuario decide realizar la simulación de la ruta lo que se hace es recorrer todos los paso de cada pierna e ir almacenando la instrucción y el punto en el que se ha ocasionado para que luego el icono gráfico pueda ir posicionándose ne un punto y con la indicación correspondiente.

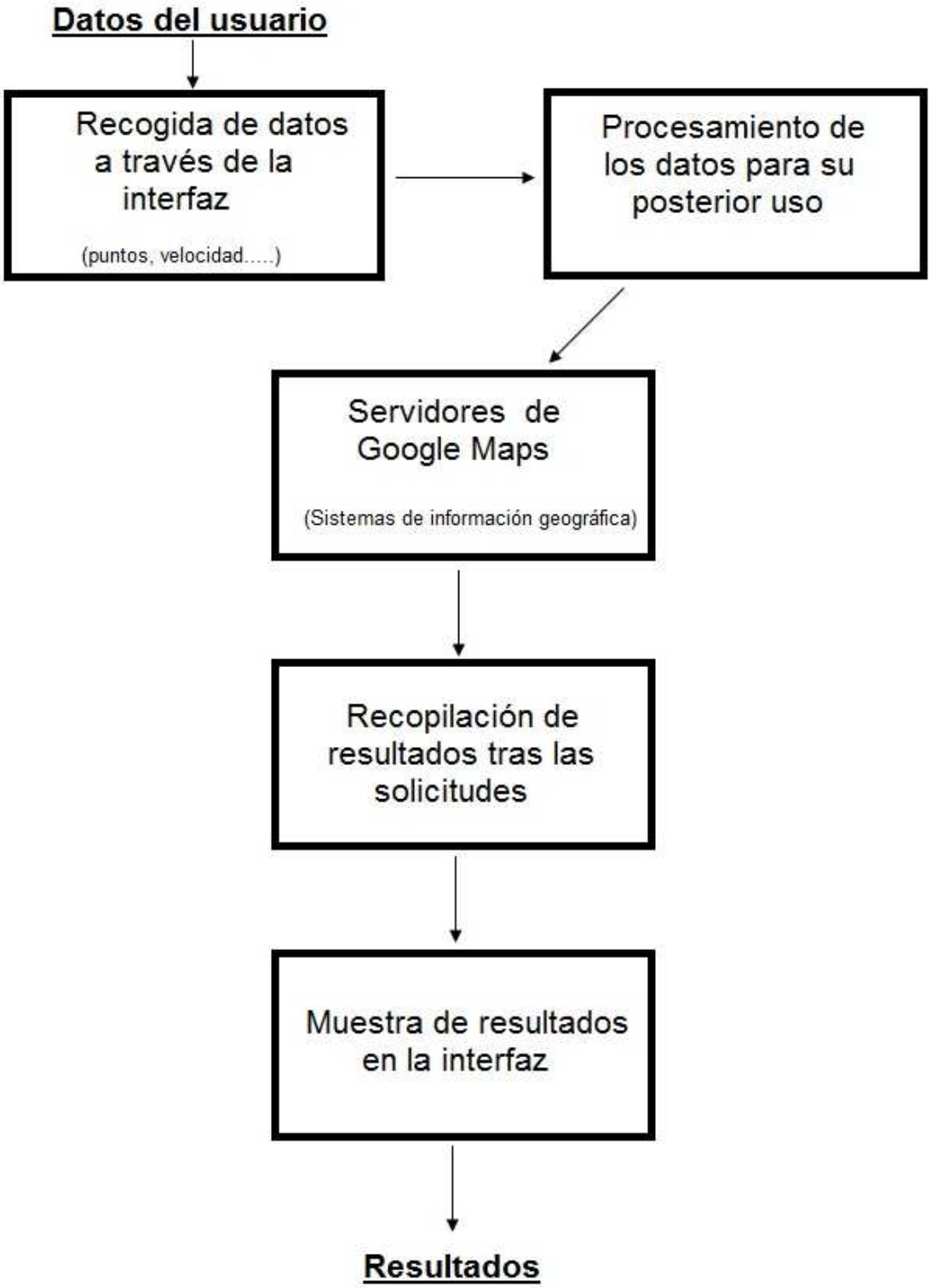
A continuación explico cómo se obtiene la instrucción de cada paso:

```
for (var i = 0; i < route.legs.length; i++) {
    for(var n=0;n<route.legs[i].steps.length;n++){
        texto += route.legs[i].steps[n].instructions + "<br />";
```

En resumen, los pasos son el campo más importante a la hora de obtener información de la ruta. De todas formas, es posible saber las posiciones de los marcadores únicamente con las piernas, ya que son las que nos indican dónde comienza y termina cada tramo.

6.4 DIAGRAMAS DE BLOQUES

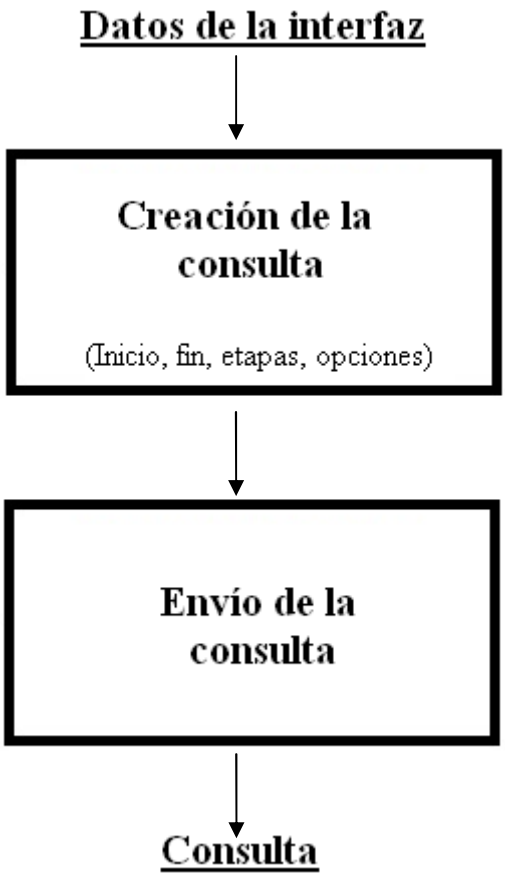
-Diagrama de bloques del sistema:



Este diagrama de bloques muestra, de manera general, el funcionamiento del proyecto, desde la interacción del usuario, pasando por la interfaz, el sistema ejecutor hasta los servidores de Google para realizar la consulta y el proceso inverso para devolver resultados al usuario.

Sistema ejecutor de un navegador personalizado

-Diagrama de bloques del procesamiento de los datos para su posterior uso:

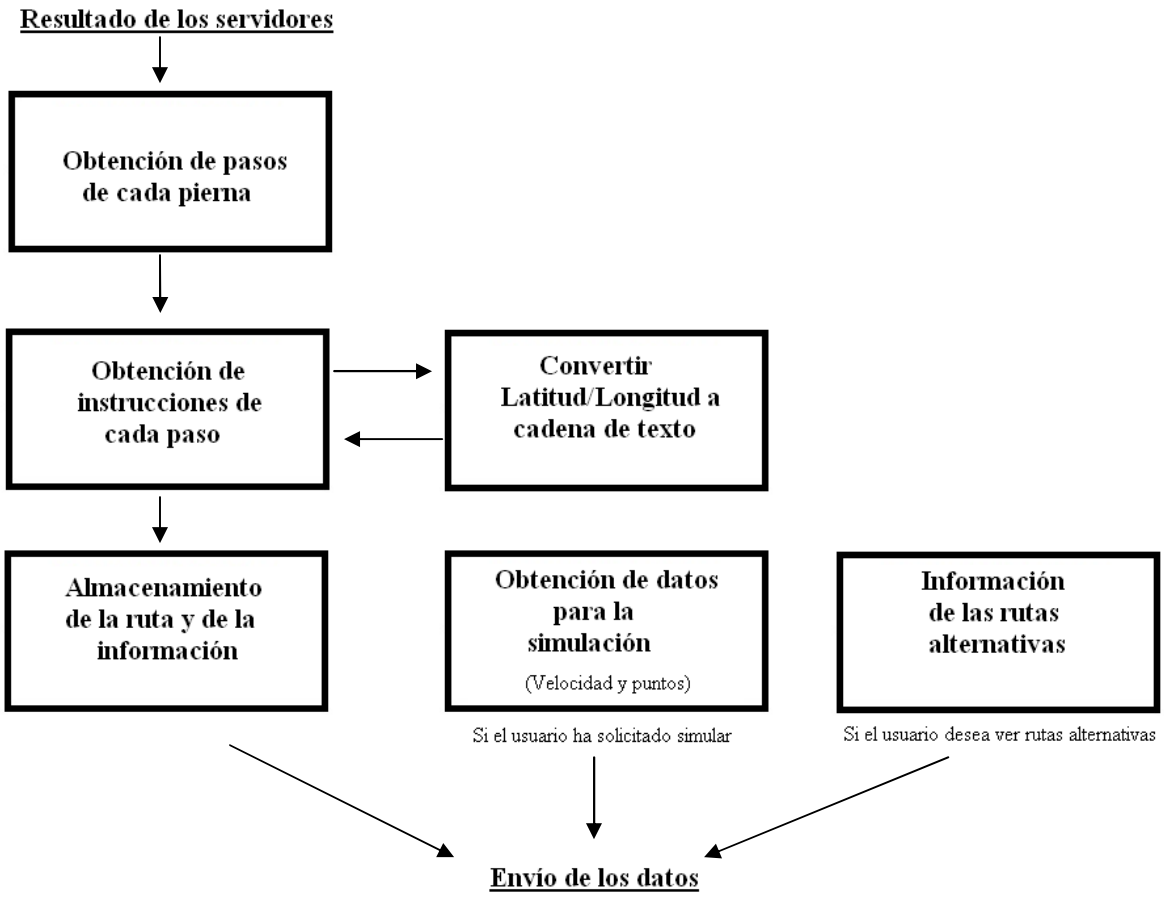


En este diagrama de bloque represento el momento en el que la interfaz me envía los datos para poder realizar la consulta.

Cuando la interfaz me envía los parámetros que necesito, ella ya ha calculado si los datos son correctos o no. Por lo tanto, únicamente tengo que recopilar estos datos y enviar la consulta a los servidores de Google Maps. Es decir, lo único que hago es completar la siguiente estructura y después pasársela a los servidores.

```
//Creamos la consulta que queremos calcular
var request = {
    origin: puntoInicial,
    destination: puntoFinal,
    waypoints: listaPuntos,
    optimizeWaypoints: true,
    provideRouteAlternatives: true,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
};
```

- Diagrama de bloques de la recopilación de los resultados tras las solicitudes:



El diagrama muestra el proceso que se realiza tras recibir el resultado de la consulta que los servidores han creado para ese trayecto.

En primer lugar tengo que conocer el número de piernas de la ruta. Recordemos que una ruta sin etapas tiene una pierna y una ruta con una etapa tendrá dos y así sucesivamente.

Para cada pierna tengo que obtener todos sus pasos porque son los que contienen las indicaciones a seguir en el trayecto además de su localización. Si es necesario, convertirá la localización a cadena de texto para que sea comprensible por el usuario.

Una vez obtenida la ruta, almaceno todos los datos para pasárselos a la interfaz y que ésta sea capaz de interpretarlos para mostrar el resultado al usuario.

Todo esto lo hago en la siguiente instrucción:

```
for (var i = 0; i < route.legs.length; i++) {
    for(var n=0;n<route.legs[i].steps.length;n++){
        texto += route.legs[i].steps[n].instructions + "<br />";
```

Donde "*route*" recoge el resultado obtenido por los servidores de Google. Así, para cada pierna de la ruta recorro cada uno de sus pasos y almaceno en "*texto*" las instrucciones de ese paso. Hago lo mismo para almacenar la duración, el tiempo y la localización, con la estructura "*Latitud/Longitud*" del paso en el mapa.

Si el usuario ha seleccionado ver una simulación deberé obtener la duración total en función de la velocidad y los puntos por los que va a pasar la simulación.

```
for (var d = 0; d < response.routes[0].overview_path.length; d++) {
    if(simulaCoorde[b].location.equals(response.routes[0].overview_path[d])){
        instruccionesOK[d]=simulaPuntos[b];
        b++;
    }
    else{
        instruccionesOK[d]= " ";
    }
    direccionesOK[d]=response.routes[0].overview_path[d];


    pintar(instruccionesOK[d].location, direccionesOK[d],d, tiempo);
}
```

Para recoger los puntos por los que pasa la simulación, en primer lugar almaceno todos los puntos en los que hay indicaciones del tipo "Gire a la derecha, en la rotonda…". Como estos puntos no son muy abundantes y la simulación quedaría muy escalonada, también almaceno aquellos puntos internos que tiene Google para dibujar la línea sobre el mapa. Estos puntos intermedios se almacenan en "*overview_path*" y pertenecen a cada ruta en vez de a cada pierna.

Si el usuario ha solicitado ver las rutas alternativas, debe obtener todas las rutas posibles y almacenarlas para pasárselas a la interfaz.

```
for (var f = 1; f<respuesta.routes.length; f++){
    var route = respuesta.routes[f];
```

Lo que hago es recorrer todas las rutas que nos devuelven los servidores en la variable "*route*" excepto la primera que esa es la óptima y ya la tengo almacenada. Por eso que el bucle comienza en la ruta 1. Después lo que hago es almacenar cada ruta en "*route*" para obtener después toda la información de esa ruta como ya se ha explicado.

7. IMPLEMENTACIÓN

En este apartado se van a destacar las partes más relevantes de la programación realizada en la aplicación. A partir del API de Google Maps, se ha podido crear un sistema de localización del cual veremos qué es lo que se ha necesitado modificar para conseguir un funcionamiento más adecuado a las necesidades de la aplicación.

Han sido muchas las funciones y opciones implementadas pero hemos querido diferenciar nuestro sistema de navegación de los ya existentes. Para ello, lo que hemos hecho ha sido añadir nuevas funcionalidades que todavía no se utilizan en otros sistemas de navegación.

Vamos a explicar:

- El proceso de registro en Google Maps
- Los problemas que hemos tenido con los puntos intermedios o etapas
- Cómo hemos obtenido las rutas alternativas
- Cómo hemos llevado a cabo la simulación
- Qué es la codificación geográfica
- Cómo mostramos las indicaciones de la ruta

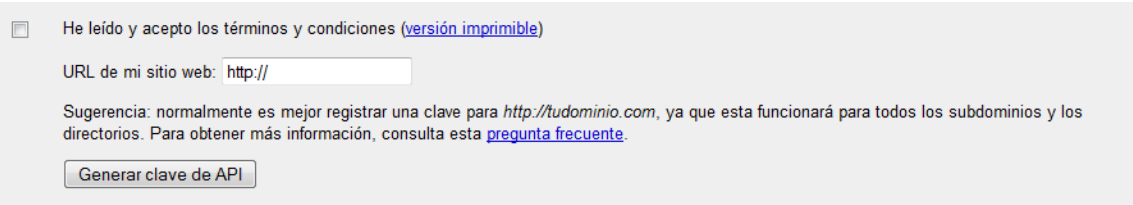
Obviamente no vamos a explicar cómo se obtiene una ruta u otras opciones básicas que ya hemos ido explicando en el proyecto en apartados anteriores.

7.1. REGISTRO EN GOOGLE MAPS

El API de Google Maps permite insertar Google Maps en páginas Web. La clave única de Google Maps API es válida para un "directorio" o para un dominio único. Para obtener una clave de API de Google Maps, es necesario disponer de una cuenta de Google, ya que la clave de API estará conectada con la cuenta de Google.

A continuación veremos algunas condiciones que impone Google Maps por utilizarlo.

- No existe limitación en el número de visitas diarias a la página que se pueden generar mediante Google Maps API.
- El número de solicitudes de codificación geográfica que se pueden enviar diariamente es limitado.
- El API de Google Maps no incluye publicidad.
- Si se utilizan otras API junto con el API de Google Maps, se debería consultar también sus condiciones del servicio. En concreto, tener en cuenta que el control GoogleBar del API de JavaScript de Google Maps utiliza el API AJAX para búsquedas y que esa API dispone de sus propias condiciones del servicio.
- Los usuarios finales deberán poder acceder de forma gratuita a tu servicio.
- No se puede modificar ni ocultar los logotipos ni la atribución del mapa.
- Se deberá indicar si la aplicación utiliza un sensor (por ejemplo, un localizador GPS) para identificar la ubicación del usuario.
- Se puede utilizar el API (salvo en el caso de Static Maps API) en sitios Web o en aplicaciones de software. En el caso de sitios Web, se debe con la URL en la que se encuentre tu implementación. En el caso de otras aplicaciones de software, regístrate con la dirección de la página en la que se puede descargar tu aplicación.
- Google actualizará las API periódicamente.
- Google se reserva el derecho de suspender o finalizar el uso de este servicio en cualquier momento.



7.2. PUNTOS INTERMEDIOS

Los puntos de paso son elementos importantes en esta aplicación y tienen un comportamiento diferente a los de inicio y fin. Ahora vamos a ver lo que los diferencia y que acciones se producen como resultado.

Como diferencia principal, se van a poder arrastrar en el mapa. Es decir, podremos modificar su localización en el mapa de una forma muy sencilla.

```
////////////////////////////////////
//Funcion que pinta los marcadores//
////////////////////////////////////
function makeMarker( position, icon) {
    marker=new google.maps.Marker({
            position: position,
            map: map,
            icon: icon,
            draggable: true
    });
    //Listener para guardar el numero de marcador que se comienza a mover
    google.maps.event.addListener(marker, "dragstart", function(event){
        coordenadas1 = pintarOrigenDestino(this.position);
        for (i=0;i<listaPuntos.length;i++){
            if (coordenadas1.equals(listaPuntos[i].location)){
                numMarcador=i;
                //Borro el contenido de coorde de la posicion numMarcador para borrar ese punto
                var aux = coorde.slice(numMarcador+3);
                coorde = coorde.slice(0,numMarcador+2);
                coorde = coorde.concat(aux);
                break;
            }
        }
        return numMarcador;
    });

    //Listener para cambiar las coordenadas del marcador que se ha movido
    google.maps.event.addListener(marker, "dragend", function(event){
        this.setVisible(false);
        listaPuntos[numMarcador].location=this.position;
        calcRoute();
    });
```

Aquí vemos cómo se crean y se les añaden dos "*Listeners*" para controlar su movimiento en el mapa. Lo que les diferencia de los otros puntos es que tienen la opción '*draggable'*, la cual permite que se arrastren.

En el primer "*Listener*", se controla que el marcador comience a arrastrarse. En primer lugar se almacenan las coordenadas actuales, y a partir de ellas se buscará la posición del Array que la contiene. Se deberá eliminar el contenido correspondiente a esta posición.

En el segundo "*Listener*", se controla el reposicionamiento del marcador. Por lo tanto, se oculta en marcador actual y se almacenan las nuevas coordenadas obtenidas al colocarlo en una nueva posición del mapa. Después se manda calcular la ruta para representarlo en ese nuevo cálculo.

Un tema muy importante a tener en cuenta con estos nuevos puntos es que, aunque ya se tengan unas nuevas coordenadas para insertar el marcador, no tienen por qué ser las definitivas. Es decir, puede ser que el marcador se sitúe en un lugar imposible de transitar y que cuando se calcule la nueva ruta no se puedan utilizar las coordenadas almacenadas.

Por ello, cuando se recalcule la ruta con ese nuevo punto, se tendrá que comprobar si es posible utilizarlo, o si no, almacenar el punto que la aplicación haya utilizado para el cálculo. Este nuevo punto estará localizado lo más cerca posible de la situación que el usuario haya seleccionado. Para entenderlo mejor, veremos el código encargado de ello.

```
//Recorremos nuestro array de puntos para ver si ya tenemos esa coordenada
//y asi no volvemos a pintar el marcador
//Primero comparamos la coordenada inicial
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].start_location).equals(coorde[cont])){
        existeInicio=true;
        break;
    }
}
if (!existeInicio){
    makeMarker(pintarOrigenDestino(route.legs[i].start_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].start_location);
}
else if (existeInicio){
    existeInicio=false;
}
//Ahora comparamos la coordenada final
for (var cont=0; cont<coorde.length; cont++){
    if (pintarOrigenDestino(route.legs[i].end_location).equals(coorde[cont])){
        existeFin=true;
        break;
    }
}
if (!existeFin){
    makeMarker(pintarOrigenDestino(route.legs[i].end_location), icons.marcador);
    coorde[coorde.length]=pintarOrigenDestino(route.legs[i].end_location);
}
else if (existeFin){
    existeFin=false;
}
```

En el código anterior se compraban las coordenadas existentes y en caso de que no coincida con ninguna de las existentes se dibuja el marcador. Y en la función siguiente, se almacenan las coordenadas correctas en un array que se emplea para el cálculo de las rutas.

```
//Sobreescribimos listaPuntos para poder
//tener almacenados en ese arrray los puntos
//reales por los que pasa la ruta

listaPuntos=null;
listaPuntos=new Array();
for (var j=2;j<coorde.length;j++){
    listaPuntos.push({
        location:coorde[j],
        stopover:true
    })
}
```

7.3. RUTAS ALTERNATIVAS

En el momento de calcular la ruta solicitada por el usuario, además de la óptima que será la que se represente de forma más destacada, también se van a calcular una serie de rutas alternativas para otorgar varias posibilidades. Así, el usuario podrá ver cual prefiere, o seleccionarlas según sea la densidad de tráfico de cada una de ellas, posibles obras en el recorrido, etc.…

Para obtener estas rutas alternativas se debe añadir una opción en la solicitud de la ruta al servidor de Google Maps, para que también las calcule. Aquí vemos cómo se añade la opción de '*provideRouteAlternatives*'.

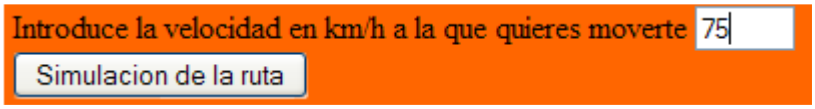
```
//Creamos la consulta que queremos calcular
var request = {
    origin: puntoInicial,
    destination: puntoFinal,
    waypoints: listaPuntos,
    optimizeWaypoints: true,
    provideRouteAlternatives: true,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
};
```

En el código anterior se compraban las coordenadas existentes y en caso de que no coincida con ninguna de las existentes se dibuja el marcador. Y en la función siguiente, se almacenan las coordenadas correctas en un array que se emplea para el cálculo de las rutas.

```
////////////////////////////////////////////////////////////////////////////////
// Funcion que muestra en una nueva ventana la informacion de las rutas alternativas //
////////////////////////////////////////////////////////////////////////////////
function prueba(){
   var testwindow2 = window.open("","Rutas alternativas","width=1000,height=1000,status=no,scrollbars=yes");
   testwindow2.moveTo(0, 0);
   var rutas=respuesta.routes.length-1;
   testwindow2.document.write("<body style='background-color:#ff6600;'><h1>Existen "+rutas+" rutas altenativas </h1>");
   for (var f = 1; f<respuesta.routes.length; f++){
       var route = respuesta.routes[f];
       testwindow2.document.write("<h3><u>Ruta alternativa "+f+" </u></h3>");
       var directionsDisplay2=new google.maps.DirectionsRenderer;
       directionsDisplay2.setOptions({
           suppressMarkers: true,
           polylineOptions: {
               strokeColor: '#ED1C24',
               strokeWeight: 3,
               strokeOpacity:0.6
           }
       });
       testwindow2.document.write("<b>Desde:</b>" +route.legs[0].start_address + "<br />");
       testwindow2.document.write("<b>Hasta:</b> " +route.legs[0].end_address + "<br />");
       testwindow2.document.write("<b>Distancia:</b> " +route.legs[0].distance.text + "<br />");
       testwindow2.document.write("<b>Duracion:</b> " +route.legs[0].duration.text + "<br />");
       testwindow2.document.write("<b>Instrucciones:</b> <br />");
       for(var n=0;n<route.legs[0].steps.length;n++){
           testwindow2.document.write(route.legs[0].steps[n].instructions+ "<br />");
       }
       directionsDisplay2.setMap(map);
       directionsDisplay2.setDirections(respuesta);
       directionsDisplay2.setRouteIndex(f);
   }
}
```

7.4 Simulación de la ruta

Cuando el usuario decide ver una representación de la ruta que ha solicitado, nuestro sistema le va a devolver un icono en el mapa, que se irá moviendo por la ruta. A medida que se mueve, mostrará indicaciones a seguir. Esta simulación la hacemos en tiempo real, es decir, que si la duración del trayecto son 6 minutos, la duración de la simulación será de seis minutos también.



El usuario puede introducir una velocidad a la que creo que, aproximadamente, irá por la ruta o bien no introducir ninguna y dejar que se calcule con la velocidad por defecto.

Google Maps determina por defecto una velocidad de:

- 26 km/h en las ciudades
- 93 km/h en trayectos que unen ciudades.

El sistema ejecutor, al recibir la orden de simular, lo primero que debe hacer es comprobar si el usuario ha introducido o no una velocidad. Con esta velocidad, calculará cada cuánto tiempo debe cambiar la posición del icono gráfico del mapa para representar su movimiento.

En caso de no haber introducido una ruta, simplemente calcula el tiempo de movimiento del gráfico dividiendo la distancia entre el número de puntos intermedios de la ruta.

Si ha introducido velocidad el proceso es más complejo. Solicitamos al usuario que introduzca la velocidad en km/h. En primer lugar hace una regla de tres de modo que:
Si por ejemplo la velocidad introducida son 70 km/h:

70_____60 minutos
Distancia_ruta_____ x minutos

De aquí sabemos cuál va a ser el nuevo tiempo para realizar la ruta a la velocidad indicada. Una vez que tenemos la duración total de la ruta a la nueva velocidad, podemos calcular el tiempo de simulación del mismo modo que cuando no introducíamos velocidad.

Una vez que sabemos el tiempo tenemos que centrarnos en los puntos en los que se va a ir dibujando el gráfico. Google Maps tiene almacenados puntos en todas las rutas. Puntos internos que, o bien no tienen información o bien son puntos en los que indican una serie de instrucciones para que el conductor no tenga problema en seguir el camino.

Resumen de la ruta con la nueva velocidad

Distancia total:2 km
Tiempo total:2 minutos

Lo que hemos hecho ha sido crear un único array uniendo estos dos tipos de puntos, de modo que tenemos muchos puntos intermedios de una ruta y además, de algunos de ellos, tenemos también indicaciones para facilitar la conducción al usuario.

Por lo tanto, a medida que nuestro icono vaya cambiando de posición según los puntos, también irá creando ventanas de información en las que muestre las instrucciones a realizar en ese punto.

Por último, para conseguir que el coche se vaya parando el tiempo calculado anteriormente en cada ruta y después cambie de posición, hemos utilizado un "*setTimeout*" para conseguir que para un tiempo determinado.

7.5 CODIFICACIÓN GEOGRÁFICA

La codificación geográfica es el proceso de transformar direcciones (como "Carlos III 3, Pamplona, España") en coordenadas geográficas, del tipo Latitud/Longitud (como 37.423021, -122.083739), que se pueden utilizar para colocar marcadores o situar el mapa.

El API de Google Maps proporciona una clase "*geocoder*" que permite codificar de forma geográfica las direcciones de forma dinámica a partir de los datos introducidos por el usuario.

Esto es útil cuando el usuario introduce una dirección y deseamos obtener más información de ese punto. En nuestro proyecto el usuario introduce dos direcciones, la de origen y la de destino. Al realizar la consulta a los servidores diferenciamos entre punto de origen, puntos de paso y punto final. Los puntos de origen y final pueden ser cadenas de texto, por lo que no necesitamos la codificación.

Sin embargo, cuando los servidores devuelven una Latitud/Longitud y queremos ver qué dirección es sí necesitamos realizar la codificación geográfica inversa. Cuando realizamos la simulación y queremos mostrar por pantalla en qué posición se encuentra el icono, necesitamos realizar la codificación geográfica inversa.

Para poder realizarlo, existe una clase en el API llamada "*Geocoder*" y que tiene el siguiente método:

Métodos	Valor de retorno	Descripción
geocode(request:*GeocoderRequest*, callback:function(Array<*GeocoderResult*>, GeocoderStatus))	None	Codifica de forma geográfica una solicitud.

En el campo de "*GeocoderRequest*" ponemos la dirección del tipo Latitud/Longitud y a continuación obtenemos la conversión.

7.6 INDICACIONES DE LA RUTA

Para mostrar al usuario todas las indicaciones que se disponen de una ruta lo primero que se debe hacer es configurar el idioma para que éstas se escriban en el idioma que el usuario va a utilizar, en nuestro caso el castellano. Para ello se debe introducir en el script en el que se invoca al API de Google Maps, al comienzo de nuestro código, dicha orden.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no"/>
<meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
<title>Sistema de Navegación Personalizado</title>
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?sensor=false&language=es"></script>
```

Una vez configurado el idioma, vamos a ver cómo se van a mostrar las indicaciones en la interfaz de la aplicación. Para no saturarla de información, se facilitará un enlace que despliegue dichas indicaciones en el momento en el que el usuario lo crea conveniente.

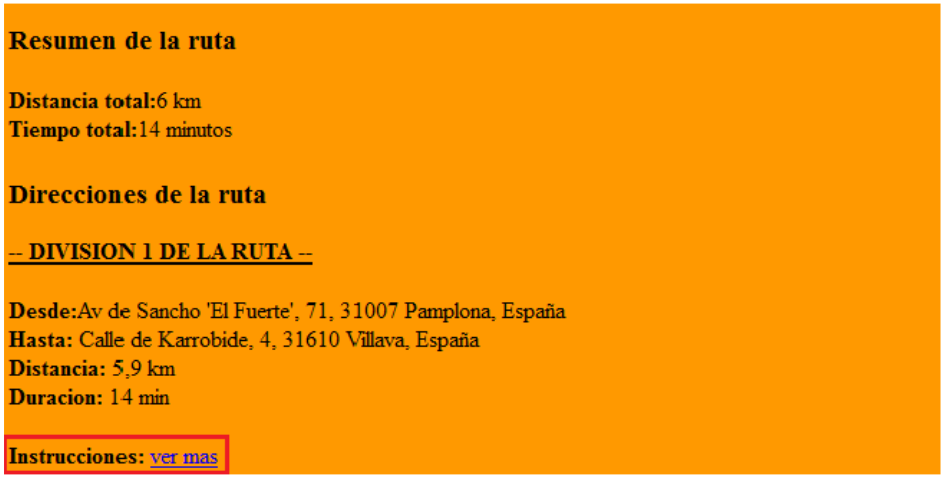
```
////////////////////////////////////////////////////////////////////////////
//Funcion que crea "ver mas" en las instrucciones completas de la ruta//
////////////////////////////////////////////////////////////////////////////
function vermas(o){
    if(o==1) panelIns.innerHTML="<b>Instrucciones: </b>"+mas;
    if(o==0) panelIns.innerHTML=texto+" "+men;
}
```

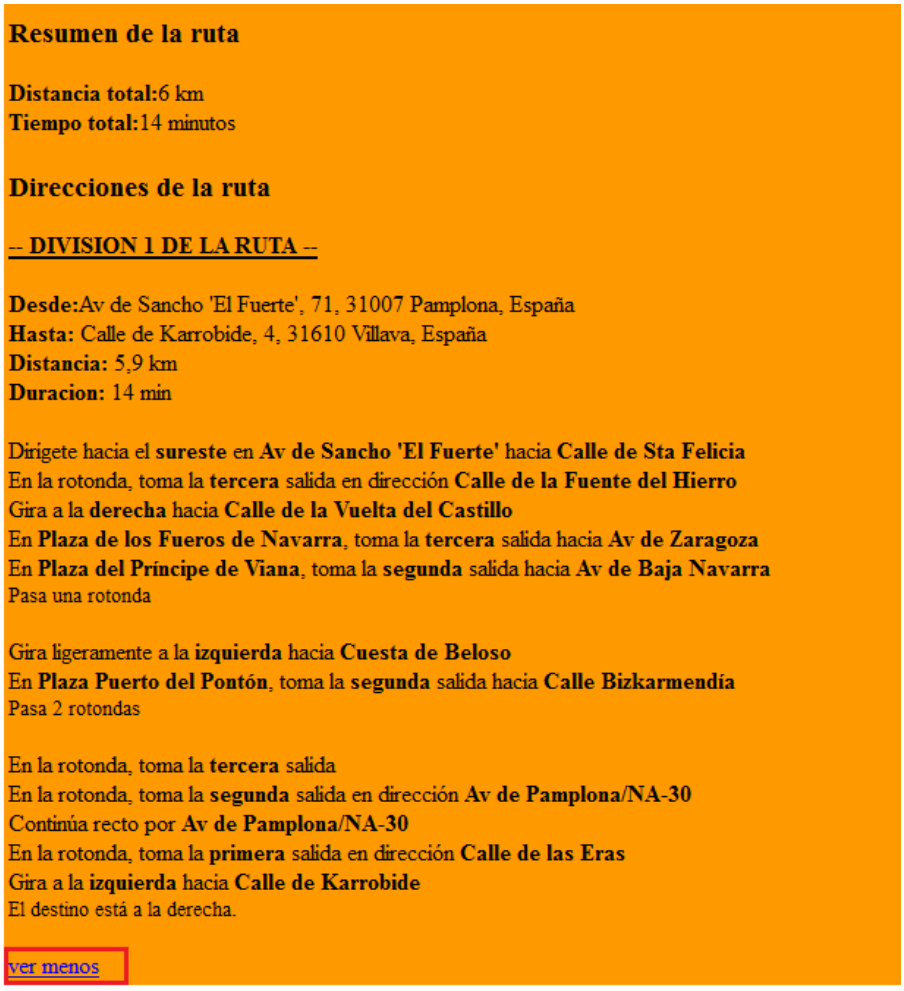
Con esta opción, el usuario podrá mostrar u ocultar la información en la ventana principal según le convenga.

```
for(var n=0;n<route.legs[i].steps.length;n++){
    texto += route.legs[i].steps[n].instructions + "<br />";
        simulaPuntos.push({
            location:route.legs[i].steps[n].instructions,
            stopover:true
        })
        var aux=pintarOrigenDestino(route.legs[i].steps[n].start_location);
        simulaCoorde.push({
            location:aux,
            stopover:true
        })
}
//Si es la primera vez que entramos a calcRoute() nos pinta el punto inicial y el final
if(primeravez){
    //Reducimos los decimales de las coordenadas que devuelve la ruta
    coorde[0]=pintarOrigenDestino(route.legs[i].start_location);
    coorde[1]=pintarOrigenDestino(route.legs[route.legs.length-1].end_location);
    //Alamaceno lo anterior por si se modifica el inicio o el fin y asi poder borrar sus marcadores
    anteriorinicio=coorde[0];
    anteriorfin=coorde[1];
    //Creamos los marcadores
    crearLimites(coorde[0],icons.inicio);
    crearLimites(coorde[1],icons.fin);
    primeravez=false;
    mas="<a href='javascript:vermas(0)'>ver mas</a>";
    men="<a href='javascript:vermas(1)'>ver menos</a>";
    panelIns.innerHTML = "<b>Instrucciones:</b> "+ mas +"<br /> ";
}
```

En el momento en el que se calcula por primera vez la ruta es donde se colocan los enlaces en la interfaz, ya que es el momento en el cual se ha generado la información para mostrar.

En esa misma función de calcular la ruta es donde se van a guardar todas las instrucciones obtenidas recorriendo todas las divisiones de la ruta.





8. CONCLUSIONES

A través de este proyecto he conseguido descubrir cómo funcionan los sistemas de navegación existentes. Además, he podido estudiar cómo acceder a los sistemas de información geográficos y cómo obtener la información deseada para poder ser mostrada a los usuarios.

Tras la realización del proyecto he llegado a la conclusión de que un sistema de navegación está dividido en tres partes. Por un lado está la parte gráfica, por otro la recepción y el envío de la información y por último los sistemas de información geográficos.

La parte gráfica se encarga de recoger los datos que el usuario introduce gracias a la interfaz. Estos datos van a ser obligatoriamente el punto de inicio y el final. Además de poder añadir etapas en la ruta.

Cuando la parte gráfica envía estos datos al sistema receptor, éste crea la consulta y la realiza al sistema de información, quien calcula runa ruta y la devuelve. En función de qué quiere visualizar el resultado, el sistema receptor determina qué información pasarle a la parte gráfica y cuál no.

Lo que hemos conseguido con este proyecto ha sido diferenciarlo de un sistema de navegación normal. Esto se debe a que hemos ofrecido al usuario funcionalidades inexistentes en otros sistemas de navegación. Estas funcionalidades son por ejemplo ver ruta alternativas, ver una simulación de por dónde pasa la ruta con las indicaciones a realizar en cada punto e incluso la opción de introducir una velocidad deseada y realizar la simulación conforme a esa velocidad.

En resumen, es un sistema de navegación más pero que permite al usuario nuevas funcionalidades hasta ahora inexistentes.

9. LINEAS FUTURAS

Tras finalizar el desarrollo del proyecto, podemos ver también hacia dónde se puede enfocar este trabajo. La investigación nos ha permitido conocer tanto el funcionamiento de los sistemas de localización, cómo ver qué más se les puede añadir atendiendo a las necesidades que se tengan.

Con ellos nos hemos dado cuenta de que podemos crear uno de estos sistemas y adaptarlo totalmente al tipo de usuario al que vaya destinado. Será cuestión de realizar un análisis de lo que el usuario desea y cuáles son los fines, ya que éstos pueden ser desde didácticos hasta lo que actualmente se conoce en el mercado de GPS.

Una nueva idea a partir de lo que nosotros hemos desarrollado, podría ser crear una aplicación similar para turistas que estén visitando una ciudad y deseen realizar un recorrido por la misma. De este modo, podrían realizar la visita de la forma óptima, minimizando sus recorridos y teniendo la posibilidad de ser ellos los que introduzcan los puntos de paso deseados.

Esta es una idea, pero su uso podría extenderse para aplicaciones de diversos tipos.

10. BIBLIOGRAFIA

[1] http://es.w3support.net/index.php?db=so&id=204644

[2] http://code.google.com/intl/es-ES/apis/maps/index.html

[3] http://www.vinagreasesino.com/articulos/google-maps-%C2%BFcomo-funciona.php

[4] http://es.wikipedia.org/wiki/Sistema_de_Informaci%C3%B3n_Geogr%C3%A1fica

[5] http://www2.uca.es/dept/filosofia/TEMA%201.pdf

[6] http://www.monografias.com/trabajos/gis/gis.shtml

[7] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/services.html

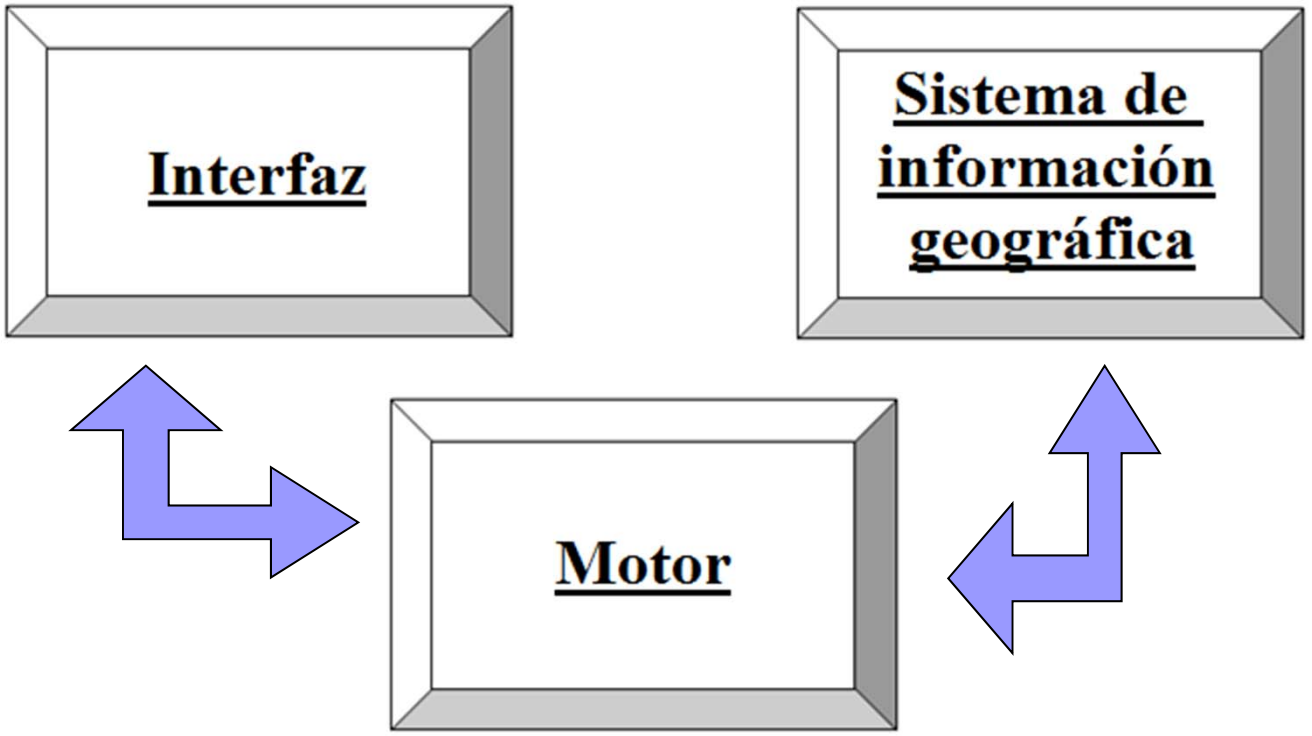
[8] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/reference.html#advertising

[9] http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/examples/index.html

Sistema ejecutor de un navegador personalizado

Nieves Gorriti Alastuey

Funcionamiento de los sistemas de navegación

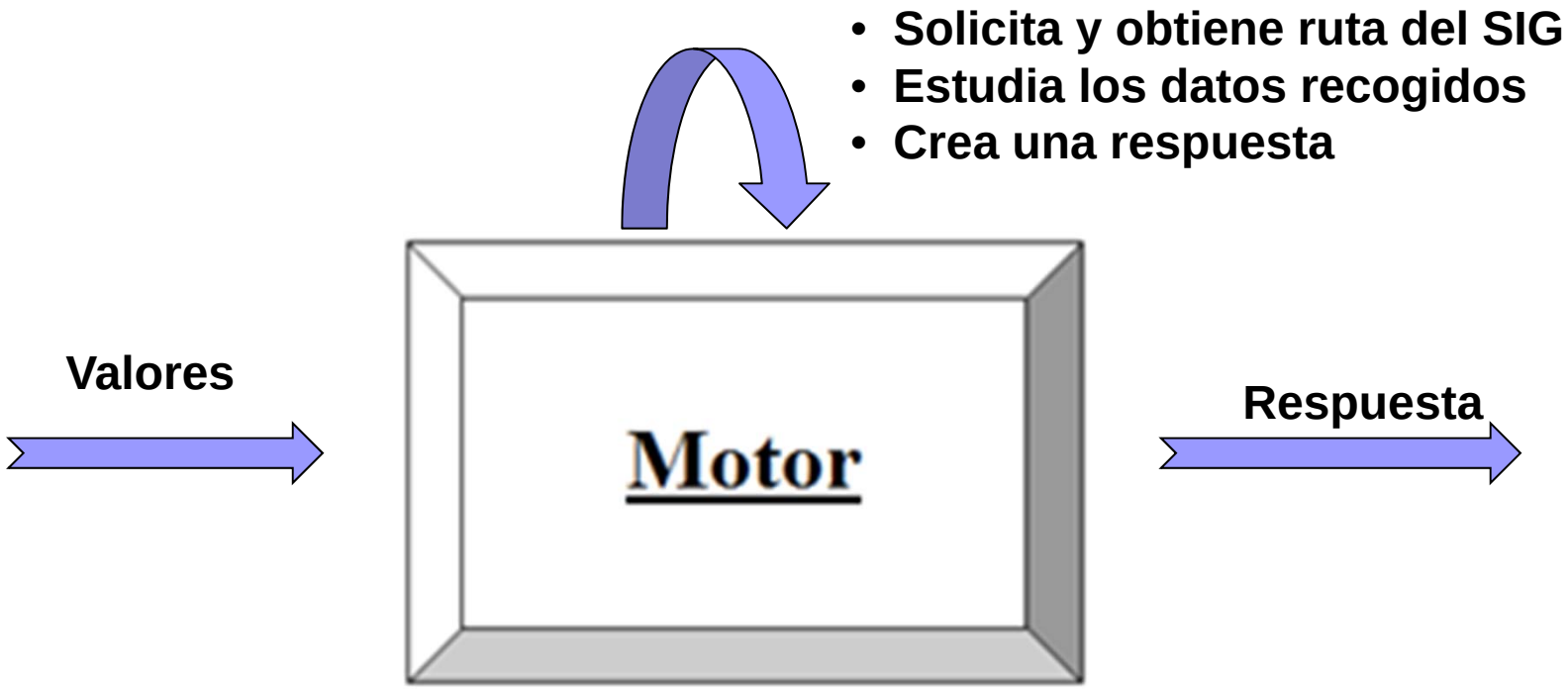


Para qué sirve el motor

- Conseguir independencia de Google
  - Crea marcadores propios
  - Crea la ruta gráficamente
  - Obtiene información del SIG
  - Adaptable a cada usuario

- API Google Maps JavaScript

Cómo funciona el motor



Recepción de datos: comunicación con la interfaz

- Coordenadas:
  - Punto de inicio
  - Punto de fin
  - Puntos intermedios
- Velocidad
- Ver rutas alternativas
- Realizar simulación

Creación de la consulta

- Clase "*DirectionsService*"

```
var directionsService = new google.maps.DirectionsService();
```

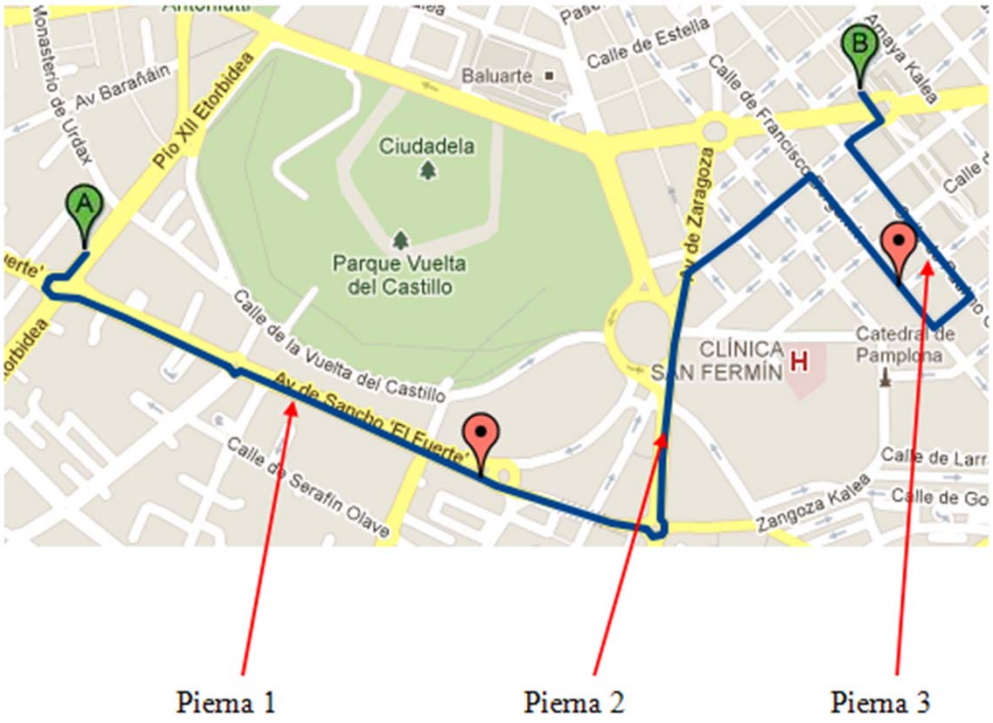
Métodos	Valor de retorno	Descripción
route(request:*DirectionsRequest*, callback:function(*DirectionsResult*, *DirectionsStatus*)))	None	Emite una solicitud de búsqueda de indicaciones.

- *Nuestra consulta*

```
//Creamos la consulta que queremos calcular
var request = {
    origin: puntoInicial,
    destination: puntoFinal,
    waypoints: listaPuntos,
    optimizeWaypoints: true,
    provideRouteAlternatives: true,
    travelMode: google.maps.DirectionsTravelMode.DRIVING
};
```

Creación de la respuesta

- Cómo se forma cada ruta



Ruta >= Pierna>Paso

Clase “DirectionsLeg”

- Distancia
- Duración
- Dirección de inicio (texto)
- Localización de inicio (Latitud/Longitud)
- Dirección de fin (texto)
- Localización de fin (Latitud/Longitud)
- Pasos: **“*DirectionStep*”**

Creación de la respuesta

- Codificación geográfica

"Carlos III 3, Pamplona, España“ ⟷ (37.423021, -122.083739)

```
geocode(request:GeocoderRequest, callback:function(Array
<GeocoderResult>, GeocoderStatus))
```

- Rutas alternativas
  - Recorrer cada ruta

Creación de la respuesta

- Simulación de la ruta:
  - Obtención de puntos intermedios:

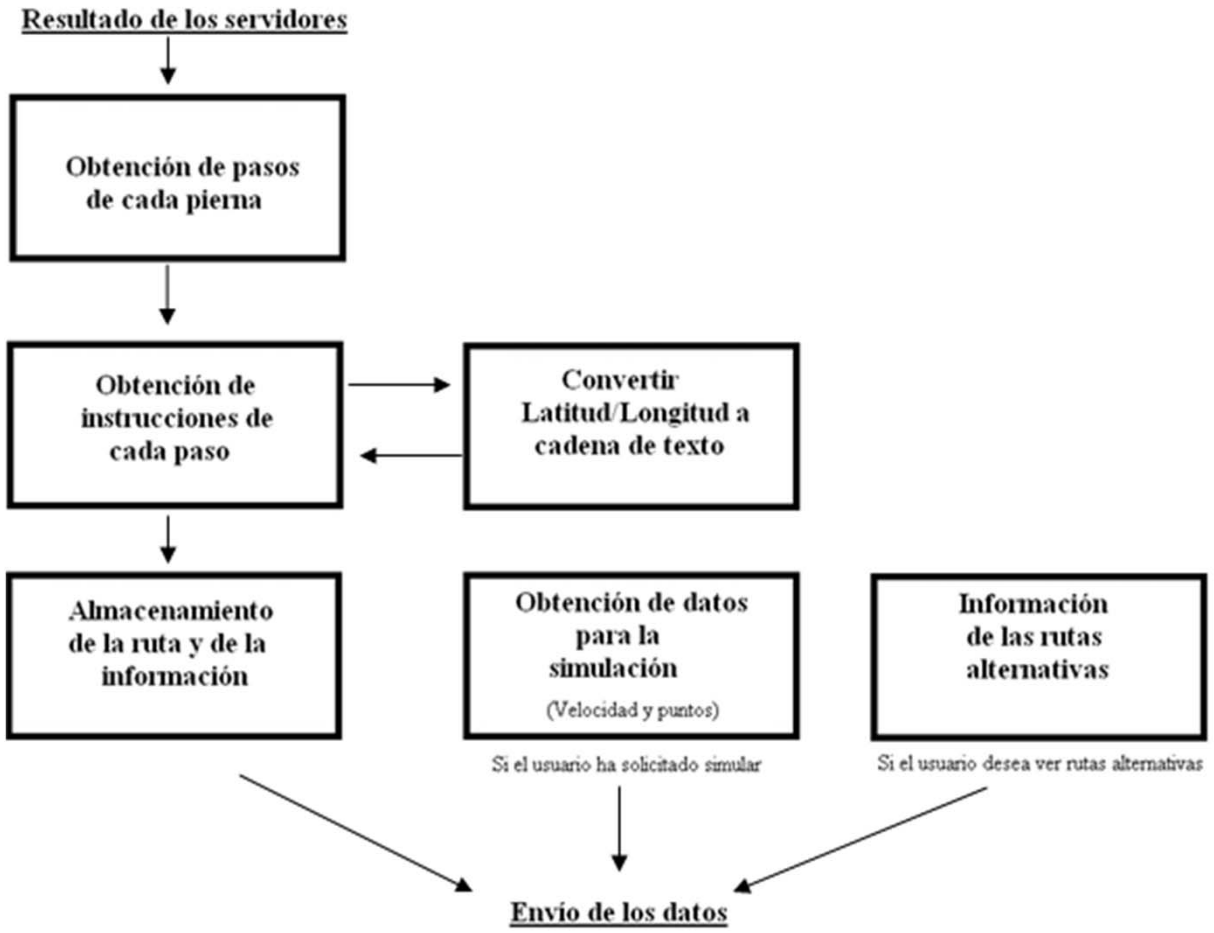
  Cada ruta tiene una serie de puntos intermedios:

`overview_path`	`Array.<LatLng>`	Un conjunto de `LatLng` que representa el recorrido completo de la ruta. La ruta se simplifica para que pueda adaptarse a contextos en los que se necesita un número reducido de vértices

  Más los puntos de las instrucciones de cada paso.

  - Cálculo de la velocidad

Creación de la respuesta



Conclusiones

- API de Google permite muchas funciones.
- El motor es independiente de Google.
- Se pueden conocer casi todos los puntos por los que pasa una ruta.
- Es necesario conectar con un SIG.
- A partir de la información del SIG se pueden obtener muchos datos.

Líneas Futuras

- Adaptar el motor a distintos usuarios
- Desarrollar un motor sin API de Google Maps
- Conseguir independencia de Internet para conectarse con el SIG