Proyecto Final de Carrera
- Ingeniería Informática -
Curso 2011/2012

Framework para la creación de visores de mapas compatibles con estándares internacionales sobre el iOS de iPhone.

Autor: Héctor Gracia Cabrera
Director: Javier Eced Cerdán
Ponente: F.Javier Zarazaga Soria

Dep. de Informática e Ingeniería de Sistemas (DIIS)
Centro Politécnico Superior
María de Luna, 1
50018 Zaragoza (España)

GeoSpatiumLab S.L.
Calle Carlos Marx, 4
Local Izdo
50015 Zaragoza (España)

Grupo de Sistemas de Información Avanzados (IAAA)
Centro Politécnico Superior
María de Luna, 1
50018 Zaragoza (España)

Agradecimientos:

A GSL, por proporcionarme en todo momento los recursos necesarios y ofrecerme el entorno y las condiciones prácticamente ideales para la realización de este proyecto. En particular a Javier, Silvia y F.Javier por su tiempo, dedicación y confianza.

A mi familia por darme plena libertad en mis decisiones y brindarme la oportunidad de realizar una carrera.

Por último, quiero agradecer también a la Universidad de Zaragoza, al Centro Politécnico Superior y al Departamento de Informática e Ingeniería de Sistemas las facilidades prestadas para la realización de este proyecto final de carrera.

Framework para la creación de visores de mapas compatibles con estándares internacionales sobre el iOS de iPhone.

RESUMEN

En los últimos años la creciente difusión de los dispositivos móviles ha venido acompañada de un creciente desarrollo de los sistemas operativos que los controlan. Estos sistemas son lo suficientemente complejos para soportar aplicaciones similares a las de un ordenador, o versiones más simplificadas de las mismas, con las limitaciones de interfaz impuestas por el reducido tamaño de la ventana de un dispositivo móvil y los escasos periféricos de entrada. Existe una fuerte competencia dentro del sector de los Sistemas Operativos (S.O.) para dispositivos móviles. Algunas de las compañías desarrolladoras de estos S.O. ofrecen kits de desarrollo de software (SDKs) para facilitar el desarrollo de aplicaciones para sus S.O.. Esto, y la gran difusión de los dispositivos móviles, hace que éste sea un sector muy atractivo para los desarrolladores de software.

GeoSpatiumLab S.L., empresa dedicada al desarrollo de Sistemas de Información Geográfica en la cual se desarrolló este PFC[1], tiene entre sus líneas de negocio el desarrollo de visores Web de mapas. El objetivo principal del PFC[1] es diseñar un nuevo modelo para la creación de visores de mapas en el S.O. iOS de iPhone, que esté basado en el desarrollado para las plataformas Web.

Para cumplir con el propósito de este PFC[1] se ha desarrollado un Framework[12] para la creación de visores de mapas que funcionan como aplicaciones nativas de iPhone y aprovechan todo su potencial tanto a nivel gráfico como de interfaz. Estos visores permiten mostrar gráficamente sobre un mapa capas de información base junto con información vectorial, proveniente de distintos formatos, en un área geográfica concreta.

Más detalladamente, los visores creados a partir de este Framework ofrecen las siguientes funcionalidades:

- Acceso a capas base proporcionadas por servicios Web compatibles con el estándar Web Map Service (WMS) de Open GeoSpatial Consortium (OGC).
- Acceso a capas base teseladas proporcionadas por servicios Web compatibles con la especificación Web Map Service Tile Caching (WMS-C) de OSGeo.
- Herramientas básicas de movimiento sobre el mapa (desplazamiento, cambio de nivel de zoom, etc.) que aprovechen todas las ventajas que ofrece la pantalla multitáctil del dispositivo.
- Representación de información vectorial sobre el mapa, mediante el dibujo de las geometrías asociadas a esa información (puntos, líneas, etc.) y posibilidad de elegir el estilo de representación de las mismas.

Adicionalmente, para mostrar todo el potencial del Framework[12], se ha desarrollado una aplicación nativa sobre iOS de iPhone para la gestión de rutas sobre el mapa. Una ruta está formada por un origen, un destino y una serie de puntos de paso y de información asociados. Esta aplicación incluye un visor creado haciendo uso de dicho Framework[12] y permite, a partir de un listado de rutas que puede ser modificado mediante la importación y exportación de datos, su representación sobre el mapa, así como obtener información de cada uno de sus elementos. Además, es capaz de geolocalizar la posición del usuario gracias al GPS del dispositivo móvil.

Índice de contenido

Capítulo 1. Introducción **6**
1.1. Contexto profesional 6
1.1.1. GeoSpatiumLab (GSL) 6
1.2. Contexto tecnológico 6
1.2.1. Sistemas de Información Geográfica (GIS) 6
1.2.2. Servicios Web y estándares para Información Geográfica 7
1.2.2.1. Servicio web de mapas (WMS-OGC) 7
1.2.2.2. WMS Tile Caching de OSGeo (WMS-C) 7
1.2.3. iOS iPhone 8
1.2.4. Objective-C 8
1.2.5. GPS 8
1.2.6. JSON 8
1.2.6.1. GeoJSON 9
1.3. Objetivos y alcance del proyecto 9
Capítulo 2. Trabajo realizado **11**
2.1. Situación previa 11
2.1.1 Estudio de mercado 11
2.2. Desarrollo del sistema 12
2.3. Arquitectura general del sistema 12
2.4. Framework gsl_mobileMap_iPhone 15
2.5. Aplicación iRoutes 22
Capítulo 3. Conclusiones **28**
3.1. Resultados obtenidos 28
3.2. Líneas futuras 34
Capítulo 4. Descripción de los anexos **36**
Anexo A. Análisis **37**
A.1. Requisitos 37
A.1.1. Requisitos funcionales 37
A.1.2. Requisitos no funcionales 38
A.2. Casos de uso 38
Anexo B. Diseño **48**
B.1. Framework gsl_mobileMap_iPhone 48
B.1.1. Modelo estático 48
B.1.2. Modelo dinámico 56
B.1.3. Configuración 61
B.2. Aplicación iRoutes 62
B.2.1. Modelo estático 62
B.2.2. Modelo dinámico 63
B.2.3. Interfaz 67
B.2.4. Configuración 69
Anexo C. Pruebas **71**
C.1. Pruebas de funcionalidad 71
C.2. Pruebas de eficiencia 73
Anexo D. Manuales **77**
D.1. Documentación técnica del Framework gsl_mobileMap_iPhone 77
D.1.1. Descripción del producto 77

D.1.2. Instalación..... 78
D.1.3. Acceso a la funcionalidad ofrecida por el Framework..... 78
D.2. Documentación pública iRoutes..... 82
D.2.1. Descripción del producto..... 82
D.2.2. Instalación y ejecución..... 82
D.2.3. Manual de usuario de la aplicación iRoutes 82
Anexo E. Servicios Web Estándar..... 89
E.1. Servicio web de mapas WMS-OGC..... 89
E.2. WMS Tile Caching de OSGeo..... 91
Glosario..... 93
Bibliografía..... 95

Capítulo 1. Introducción.

Este capítulo define el contexto en el que se desarrolla el proyecto, tanto a nivel profesional como tecnológico, y especifica los objetivos del mismo.

1.1. Contexto profesional

1.1.1. GeoSpatiumLab (GSL)

GeoSpatiumLab es una empresa de ingeniería informática que se dedica a la creación y puesta en servicio de tecnología para la gestión y difusión de información georreferenciada (también denominados generalmente datos geográficos y ahora más comúnmente datos espaciales) sobre un nuevo paradigma que se ha convenido en llamar Infraestructuras de Datos Espaciales (IDEs).

Nace como un *Spin-Off* de la Universidad de Zaragoza tomando como punto de partida la experiencia de más de 10 años del Grupo de Sistemas de Información Avanzados (IAAA) de la Universidad de Zaragoza. El trabajo de investigación del grupo trata de cubrir desde aspectos básicos de ingeniería de servicios hasta aspectos metodológicos de la puesta en funcionamiento de sistemas industriales que explotan la información geográfica. Concretamente, la labor de investigación del grupo aborda problemáticas vinculadas a las ontologías y meta-datos; interoperabilidad, composición y encadenamiento de servicios; visualización inteligente de información geográfica, metadatos, procesamiento semántico y recuperación inteligente de información (indexación, interrogación y recuperación); métodos y procesos para la creación de información y el modelado de contenidos heterogéneos; y, finalmente, un aspecto que se considera fundamental para la realimentación técnica y que es la definición, creación y puesta en funcionamiento real de nuevos servicios y aplicaciones.

1.2. Contexto tecnológico

1.2.1. Sistemas de Información Geográfica (*GIS*)

Los Sistemas de Información Geográfica (*Geographic Information Systems*, *GIS*) son sistemas de información capaces de 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.

Los *GIS* separan la información en diferentes capas temáticas y las almacenan

independientemente, permitiendo trabajar con ellas de manera rápida y sencilla, y facilitando al profesional la posibilidad de relacionar la información existente a través de la topología de los objetos, con el fin de generar nueva información que, de otra forma, no podríamos obtener.

Hay dos tipos fundamentales de datos dentro de los *GIS*: datos *raster* y datos vectoriales. Los datos *raster* son una matriz de celdas que representan un cierto espacio continuo cuyo contenido son valores numéricos que representan algún tipo de valor. En esta matriz cada elemento representa una zona geográfica (generalmente un cuadrado de algunos metros de lado). El tamaño de este cuadrado es la resolución de la cobertura *raster*; a mayor resolución (cuadrado más pequeño), más nivel de detalle. En los datos vectoriales la información geométrica se almacena en forma de elementos geométricos (básicamente puntos, líneas y polígonos, aunque existen tipos mucho más elaborados) definidos mediante sus coordenadas numéricas en un sistema de coordenadas concreto.

1.2.2. Servicios Web y estándares para Información Geográfica

En los últimos años, una de las tendencias en los *GIS* ha sido la de ser cada vez más accesibles a través de Internet. Una de las respuestas a esta tendencia es la especificación de servicios de visualización de mapas por Internet, mediante la persecución de acuerdos entre las empresas del sector que posibiliten la interoperación de sus sistemas de geoprocesamiento y faciliten el intercambio de información geográfica. Concretamente, las siguientes especificaciones de servicios de visualización han sido las utilizadas en este PFC [1]

1.2.2.1. Servicio web de mapas (WMS-OGC)

El servicio *Web Map Service* (*WMS*) definido por el consorcio *Open Geospatial Consortium* (OGC [2]), creado en 1994 con el fin de definir estándares abiertos e interoperables dentro de los GIS, produce mapas dinámicamente a partir de datos espaciales georeferenciados. Según este estándar internacional, un 'mapa' es una representación de la información geográfica en forma de un archivo de imagen digital para ser presentada en la pantalla de un ordenador (generalmente en formato de imagen como *PNG*, *GIF* o *JPEG*, y ocasionalmente como gráficos vectoriales en *SVG* o *WebCGM*). Un mapa no consiste en los propios datos, sino en su representación. Este estándar consolidado es uno de los que se han usado en este PFC [1] debido al gran número de servicios de visualización que se basan en él.

1.2.2.2. WMS Tile Caching de OSGeo (WMS-C)

La especificación *WMS Tile Caching* o *Tile Map Service* propuesta por *OSGeo* [3], comúnmente denominado WMS-C, es un estándar no oficial que describe el acceso a los *tiles* [4] que conforman un mapa cartográfico, indicando el modo en que los clientes deben requerir los *tiles* (o teselas, en español) que conforman dicho mapa y el modo en que el servicio los almacena. Un *tile* es una porción cuadrada de información que representa una parte de la extensión total de un área geográfica.

Normalmente, esta especificación es usada para el tratamiento de mapas con amplio ciclo de vida en los que las áreas mostradas cambian despacio y el acceso a estos cambios no es prioritario. Son además grandes en volumen y con un número potencial de *tiles* [4] muy elevado. Por todo ello es una especificación comúnmente utilizada a la hora de implementar herramientas de cacheado de

mapas y ha sido usada para la realización de este PFC [1].

1.2.3. iOS iPhone

iPhone es un teléfono inteligente multimedia con conexión a Internet, pantalla táctil capacitiva (con soporte multitáctil) y una interfaz de software minimalista de la compañía Apple Inc. Carece de un teclado físico, por lo que integra uno en la pantalla táctil con orientaciones tanto vertical como apaisado.

iOS (anteriormente denominado iPhone OS) es un sistema operativo móvil de Apple desarrollado originalmente para el iPhone, siendo después usado en el iPod Touch e iPad. Es un derivado del Mac OS X. El iOS tiene 4 capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios de comunicación" y la capa de "Cocoa Touch". Todo el sistema ocupa poco menos de 500 megabytes.

La ultima versión de iOS al comenzar este proyecto era la 4.3. Por ello sera la versión elegida para la que se desarrollaran los componentes creados a lo largo del PFC [1].

1.2.4. Objective-C

Objective-C es un lenguaje de programación orientado a objetos creado como un superconjunto de C pero que implementase un modelo de objetos parecido al de Smalltalk [5]. Originalmente fue creado por Brad Cox y la corporación StepStone. Posteriormente fue adoptado como lenguaje de programación de NEXTSTEP [6] y liberado bajo licencia GPL [7] para el compilador GCC [8]. Actualmente se usa como lenguaje principal de programación en Mac OS X y GNUstep [9] mediante el entorno de desarrollo XCode, que proporciona herramientas muy útiles a la hora de desarrollar aplicaciones con este lenguaje, como puede ser el simulador de iPhone. La versión de XCode utilizada en el desarrollo de este PFC [1] fue la 3.2.6, última versión gratuita de este entorno, que permite programar para el iOS 4.3.

1.2.5. GPS

El *GPS* (*Global Positioning System*: sistema de posicionamiento global) o *NAVOSTAR-GPS* es un sistema global de navegación por satélite (*GNSS*) que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de centímetros, aunque lo habitual son unos pocos metros de precisión. El sistema fue desarrollado, instalado y actualmente operado por el Departamento de Defensa de los Estados Unidos.

La capacidad del iPhone para usar este sistema se empleará para cumplir ciertos requisitos exigidos en este proyecto.

1.2.6. 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*.

La simplicidad de *JSON* ha dado lugar a la generalización de su uso, especialmente como alternativa a *XML* en *AJAX*. Una de las supuestas ventajas de *JSON* sobre *XML* como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador semántico de *JSON*. En *JavaScript*, *JSON* puede ser analizado trivialmente, lo cual ha sido fundamental para la aceptación de *JSON* por parte de la comunidad de desarrolladores *AJAX*, debido a la ubicuidad de *JavaScript* en casi cualquier navegador web.

En cualquier caso, los argumentos a favor de la facilidad de desarrollo de analizadores o del rendimiento de los mismos se ven ensombrecidos por la cuestión de la falta de seguridad de los mismos, por lo que este formato se emplea habitualmente en entornos donde el tamaño del flujo de datos entre el cliente y el servicio es muy importante y cuando la fuente de datos es explícitamente de fiar.

La importancia de este formato de archivo para este PFC [1] se basa en la decisión adoptada por la empresa GSL de utilizarlo para el intercambio de datos, a la hora de desarrollar visores Web, Android e iPhone, por ser ligero y fácilmente interpretable

1.2.6.1. GeoJSON

GeoJSON es un formato abierto para codificar una gran variedad de estructuras de datos geográficos. Se llama así porque se basa en *JSON*. De hecho, cada estructura de datos *GeoJSON* es también un objeto *JSON*, y por lo tanto las herramientas de *JSON* también pueden ser utilizadas para el procesamiento de datos en formato *GeoJSON*.

Este formato también permite especificar un sistema de coordenadas geográficas, con el *OGC* [2] CRS (sistema de coordenadas de referencia). Esta cualidad ya se había aprovechado en los visores desarrollados por la empresa GSL y se utilizara en este PFC [1] para los ficheros de rutas y POIs [10] (*Point Of Interes*) que será capaz de importar la aplicación iRoutes [11].

1.3. Objetivos y alcance del proyecto

En los últimos años la creciente difusión de los dispositivos móviles ha venido acompañada de un creciente desarrollo de los sistemas operativos que los controlan. Estos sistemas son lo suficientemente complejos para soportar aplicaciones similares a las de un ordenador, o versiones más simplificadas de las mismas, con las limitaciones de interfaz impuestas por el reducido tamaño de la ventana de un dispositivo móvil y los escasos periféricos de entrada.

GeoSpatiumLab S.L., empresa dedicada al desarrollo de Sistemas de Información Geográfica en la cual se desarrolla este PFC [1], tiene entre sus líneas de negocio el desarrollo de visores Web de mapas. La idea principal es trasladar el modelo que usan estos visores desarrollados en tecnologías Web al S.O. iOs de iPhone.

El propósito de este PFC [1] es, por tanto, desarrollar un Framework [12] para la creación de visores de mapas que funcionen como aplicaciones nativas de iPhone y se aprovechen de todo su potencial tanto a nivel gráfico como de interfaz. Estos visores permitirán mostrar gráficamente

sobre un mapa capas de información base junto con información vectorial, proveniente en distintos formatos, en un área geográfica concreta.

Más detalladamente, los visores creados a partir de este Framework ofrecerán la siguiente funcionalidad:

• Acceso a capas base proporcionadas por servicios Web compatibles con el estándar Web Map Service (WMS) de Open GeoSpatial Consortium (OGC [2]).

• Acceso a capas base teseladas proporcionadas por servicios Web compatibles con la especificación Web Map Service Tile Caching (WMS-C) de OSGeo [3].

• Herramientas básicas de movimiento sobre el mapa (desplazamiento, cambio de nivel de zoom, etc.) que aprovechen todas las ventajas que ofrece la pantalla táctil del dispositivo.

• Representación de información vectorial sobre el mapa, mediante el dibujo de las geometrías asociadas a esa información (puntos, líneas, etc.) y posibilidad de elegir el estilo de representación de las mismas.

Adicionalmente, para mostrar todo el potencial del Framework, se desarrollará una aplicación nativa sobre iOS de iPhone que permita la gestión de rutas sobre el mapa, que incluya un visor creado a partir del Framework desarrollado en el presente PFC [1]. Una ruta consiste en un itinerario con un origen y un destino (cuya representación gráfica será una o más líneas), una serie de puntos intermedios (en caso de haber varios tramos) y, opcionalmente, un listado de puntos de interés (POI [10]) asociados a la misma. Esta aplicación permitirá, a partir de un listado de rutas que puede ser modificado, mediante la importación y exportación de datos, su representación sobre el mapa. Además será capaz de geolocalizar la posición del usuario gracias al GPS del dispositivo móvil.

Capítulo 2. Trabajo realizado

Este capítulo detalla el trabajo realizado a lo largo de este proyecto. Parte de la situación previa para, posteriormente, una vez explicada la política de trabajo que se ha seguido, estudiar el sistema a nivel general y luego componente a componente, según se ha ido desarrollando.

2.1. Situación previa

En este apartado se intentara explicar la motivación de este proyecto.

En la empresa GSL, donde se ha desarrollado este PFC [1], ya contaban con Framework [12] y visores web con acceso a servicios WMS-OGC y WMS-C, basados en la librería OpenSource OpenLayers, y también Framework y visor de rutas para Android (sistema operativo para dispositivos móviles ideado por Google). Ambos Framework tienen un modelo de datos similar, que se ha usado como base en las decisiones de análisis y diseño tomadas en este PFC. La principal motivación de la realización de este PFC ha sido cumplir con uno de los objetivos de la empresa de introducirse en el mercado de las aplicaciones para iPhone, ya que en este ámbito existen pocas aplicaciones relacionadas con la visualización y gestión de datos geográficos que se basen en estándares web como los ofrecidos por OGC [2] u OSGeo [3]. Estas aplicaciones ya existentes se estudian más detenidamente en el siguiente estudio de mercado.

2.1.1 Estudio de mercado

Una de las tareas que se realizó al iniciar este PFC [1] fue el estudio de Frameworks [12] para el desarrollo de visores de mapas para iPhone disponibles en el mercado. Uno de estos Frameworks es Route-me, todavía en desarrollo, capaz de mostrar mapas de OSM (OpeenStreetMap) o de Yahoo! Maps, ambos tileados [4].

Por otro lado, la compañía Apple, ofrece, en su entorno de desarrollo para iPhone XCode, el Framework MapKit, capaz de crear vistas plenamente funcionales de los mapas de Google de un modo muy sencillo.

Ambos Framework dan la posibilidad de añadir POIs [10] al mapa y tienen las funciones básicas de movimiento y zoom sobre el mapa, pero ninguno de ellos esta preparado para añadir datos vectoriales complejos a partir de archivos con formato GeoJSON ni para funcionar con servicios WMS-OGC o WMS-C de OSGeo [3]. La importancia de estos servicios no solo radica en el cumplimiento de estándares mundialmente aceptados, también se basa en su frecuente uso en el ámbito de las Infraestructuras de Datos Espaciales, por las cuales la empresa GSL muestra un claro

interés. Estas son las principales funcionalidades que pretende aportar el Framework [12] desarrollado en este proyecto.

2.2. Desarrollo del sistema

La realización de este PFC [1] fue pensada como un proceso basado en una serie de iteraciones incrementales. El desarrollo de cada una de las iteraciones era necesario para la siguiente, bien por el uso directo de un componente sobre otro o por el simple hecho de ir añadiendo funcionalidad al sistema final de forma ordenada y asegurando el correcto funcionamiento de una parte antes de pasar a otra. Además, los problemas surgidos durante el desarrollo de alguna de las iteraciones eran siempre útiles para el más rápido avance de las posteriores.

Una vez definida la metodología de trabajo, entraremos más en profundidad a detallar cómo se definieron cada una de las iteraciones y qué objetivos había que cumplir antes de pasar a la siguiente.

En primer lugar, la prioridad era poder visualizar un mapa básico sobre el que empezar a trabajar. Así pues, la primera iteración consiste en la creación de una aplicación para iPhone con una única vista que sea capaz de visualizar servicios de mapas WMS-OGC de una única *tile* [4] que asentara la estructura de los futuros mapas más complejos.

Las siguientes iteraciones se basaron en ir añadiendo funcionalidades y complejidad al mapa básico de la primera iteración. En primer lugar se trabajará con la posibilidad de añadir información vectorial al mapa, después con los controles básicos que nos permitirán movernos por el mapa, continuaremos incrementando la complejidad del mapa para poder hacer uso de servicios WMS-C, para finalmente terminar con la posibilidad de extraer información de archivos formateados y poder añadirla al mapa.

Una vez terminadas las funcionalidades básicas del Framework [12], se procederá a crear una aplicación más compleja que nos permita importar rutas en formato GeoJSON y su visualización sobre cualquier servicio de mapas compatible con los estándares antes citados.

2.3. Arquitectura general del sistema

El conjunto del sistema desarrollado está compuesto por dos componentes, la librería o Framework [12] gsl_mobileMap_iPhone, que es la encargada de crear el mapa, su visualización y sus controles, y la aplicación iRoutes [11], que hace uso de la posibilidad que nos da la librería de añadir datos vectoriales al mapa extrayéndolos de archivos GeoJSON con una estructura predeterminada, permitiéndonos visualizar información de rutas y puntos de interés de un modo rápido y sencillo.

Las siguientes imágenes muestran, por un lado las partes de la interfaz que pertenecen a cada uno de los componentes y por otro las partes más importantes de la estructura interna de los componentes y la relación existente entre ella y cualquier servicio externo:

Figura 2.1. Arquitectura del sistema

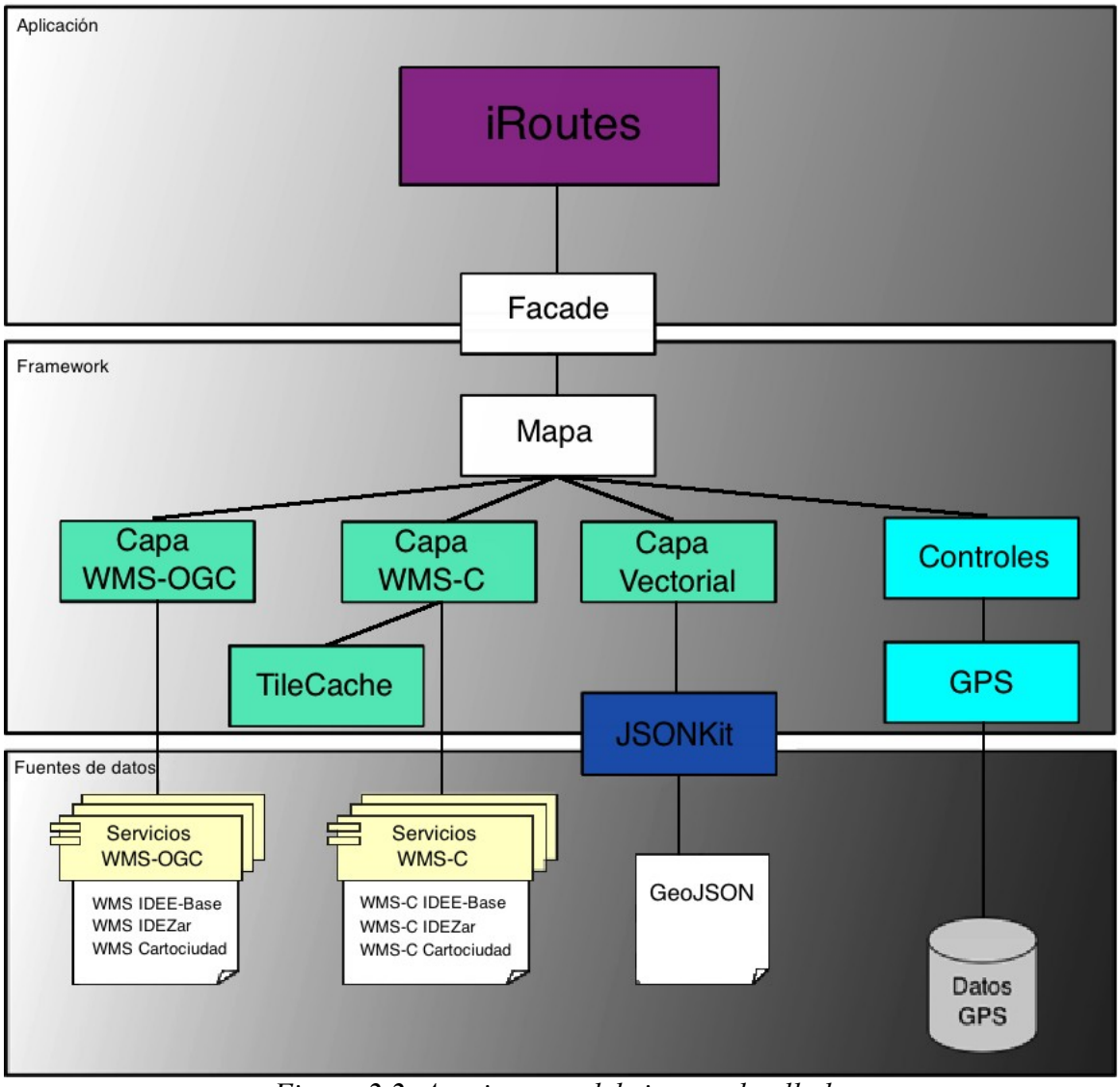

Figura 2.2. Arquitectura del sistema detallada

Para que la comunicación entre los dos componentes sea más sencilla y la integración de la librería en otras aplicaciones sea menos complicada, se utilizó el patrón de diseño Facade [13]. Este patrón se basa en establecer un único punto de acceso al conjunto de clases existentes, abstrayendo el comportamiento interno y permitiendo que otro desarrollador sea capaz de utilizar la librería conociendo únicamente los métodos del Facade, ahorrando mucho tiempo en el desarrollo de futuras aplicaciones que puedan usar este Framework [12]. Por eso en la arquitectura general, la aplicación iRoutes [11] únicamente tiene conexión con el Facade.

En la parte del Framework se encuentran las estructuras necesarias para crear los tres tipos diferentes de capas que podrán ser agregadas al mapa, las capas WMS-OGC, las WMS-C y las Vectoriales. Estas últimas están asociadas a la estructura JSONKit, que no aparece por completo en el interior del Framework debido a que es un Framework externo, que no ha sido desarrollado en este PFC [1] y que se explicará detenidamente más adelante. También contamos con la estructura que se encargará de manipular todo lo relacionado con el GPS.

Los tres servicios de los que se obtienen datos WMS-OGC y con los que se han realizado pruebas en este proyecto son:

- IDEE_Base: http://www.idee.es/wms/IDEE-Base/IDEE-Base?
  WMS-OGC ofrecido por la Infraestructura de Datos Espaciales de España (IDEE).

- IDEZAR: http://idezar.zaragoza.es/wms/IDEZar_base/IDEZar_base?
  WMS-OGC ofrecido por la Infraestructura de Datos Espaciales de Zaragoza (IDEZar).

- Cartociudad: http://www.cartociudad.es/wms/CARTOCIUDAD/CARTOCIUDAD?
  WMS-OGC ofrecido por la Administración General del Estado que muestra la cartografía en el ámbito urbano.

En cuanto a los servicios WMS-C, cabe destacar la implementación en el mismo Framework [12] de una cache de *tiles* [4] en memoria, que utilizará la cache del iPhone para guardar temporalmente los datos recibidos por los siguientes servicios y mejorar drásticamente la eficiencia del visualizador al utilizar este tipo de servicios, ya que evita, en la medida de los posible, repetir peticiones de *tiles* ya realizadas. Los servicios de este tipo que se han utilizado son estos:

- IDEE_Base: http://www.idee.es/wms-c/IDEE-Base/IDEE-Base?
  WMS-C ofrecido por la Infraestructura de Datos Espaciales de España (IDEE).

- IDEZAR: http://clamosa.cps.unizar.es:8080/TileCache_IDEZar/WMSTileCache?
  WMS-C ofrecido por la Infraestructura de Datos Espaciales de Zaragoza (IDEZar).

- Cartociudad: http://www.cartociudad.es/wms-c/CARTOCIUDAD/CARTOCIUDAD?
  WMS-C ofrecido por la Administración General del Estado.

Una vez descrita la arquitectura general, se va a explicar cada uno de los componentes principales en más profundidad.

2.4. Framework gsl_mobileMap_iPhone

Como ya se ha mencionado anteriormente, el diseño de esta librería se basó en los modelos de la librería OpenSource OpenLayers y en el Framework [12] para Android de GSL, comenzando por un diseño básico al que se le fueron añadiendo funcionalidades hasta cumplir con los objetivos de este PFC [1].

Debido al desconocimiento inicial del lenguaje Objective C y de la metodología a seguir a la hora de realizar aplicaciones para móviles, el diseño inicial no se corresponde al cien por cien con el final. Conforme se fueron afianzando conocimientos y profundizando en la complejidad del lenguaje, se mejoraron ciertos aspectos del diseño inicial. Estos cambios se explicarán más detenidamente en el anexo B, pero por poner un ejemplo, la petición que realiza cada *tile* [4] al servicio WMS-OGC, al principio se realizaba mediante una conexión síncrona, lo que causaba el bloqueo de la aplicación hasta que se recibía la petición. Como este comportamiento no era aceptable, se utilizó la clase NSThread para que cada *tile* lanzara un *thread* o hilo distinto que era el que se encargara de realizar la petición y de recibirla en segundo plano. En iteraciones posteriores, se sustituyó esta clase por la connectionDelegate, una nueva clase que realiza peticiones asíncronas, combinada con el uso de notificación de eventos para informar de que un *tile* había sido descargada y ya estaba disponible para pintarla en el mapa, evitándonos las costosas operaciones de creación y destrucción de *threads*.

El primer objetivo a alcanzar en el desarrollo de la librería fue poder visualizar un mapa formado por una única *tile* [4], cuya imagen proviniera de una petición real a un servicio WMS-OGC. De este modo se crearía la estructura base del mapa, a la que posteriormente se iría añadiendo complejidad.

Para poder comprobar el correcto funcionamiento de las peticiones, se utilizó como servicio de pruebas el WMS-OGC IDEE-Base ofrecido por el Instituto Geográfico Nacional (IGN), de acceso público, y que sigue el estándar WMS-OGC. La siguiente petición es un ejemplo de una prueba realizada con este servicio, en la que se piden dos capas, fondo e hidrografía, de un área concreta de Zaragoza:

http://www.idee.es/wms/IDEE-Base/IDEE-Base?
TRANSPARENT=true&VERSION=1.3.0&BGCOLOR=0xFFFFFF&SERVICE=WMS&REQUEST=GetMap&STYLES=default&EXCEPTIONS=application/vnd.ogc.se_inimage&FORMAT=image/png&LAYERS=Fondo,Hidrografia&BBOX=41.6320789836574,-0.8934429101563,41.6715611003566,-0.8659770898438&CRS=EPSG:4326&WIDTH=320&HEIGHT=460

La siguiente imagen con el formato solicitado (PNG en este caso) es la respuesta construida por el servicio de visualización en base a los parámetros de la petición que incluyen el área geográfica y las capas solicitadas. Para información más detallada sobre el funcionamiento de estas peticiones, consultar el Anexo E.

Figura 2.3. Petición al WMS-OGC de IDEE-Base

La anterior imagen, generada mediante petición a un servicio WMS-OGC, constituye la primera versión del mapa, a la cual iremos añadiéndole complejidad hasta conseguir un mapa con múltiples *tiles* [4] y múltiples capas o *layers* provenientes incluso de diferentes servicios.

Antes de pasar a aumentar el número de *tiles* del mapa, se consideró interesante comprobar si se podía añadir otro tipo de información que no estuviese almacenada en forma de imagen, por ejemplo información vectorial. Para conseguirlo se añadió al mapa la posibilidad de incluir capas vectoriales con dos tipos de geometrías, puntos y líneas, georreferenciadas, es decir, asociadas a unas coordenadas. Este tipo de dato vectorial suele denominarse *feature* [14] y será el tipo de dato base de las capas vectoriales. Esta nueva funcionalidad permite que el mapa muestre varias capas superpuestas y haciendo coincidir la información que en ellas esta representada.

Para comprobar que esta nueva funcionalidad actuaba correctamente, se modificó la petición de la *tile* [4] inicial que componía el mapa, para que mostrara una zona de ejemplo (en este caso, la Plaza de Europa de Zaragoza) y se añadió una capa vectorial con un punto localizado en el centro de la plaza.

Figura 2.4. Plaza de Europa

Figura 2.5. Plaza de Europa y dato vectorial

Como puede observarse en la imagen, el punto azul se encuentra situado justo en el centro de la plaza, como se esperaba. Ese punto es el único componente de la capa vectorial que se encuentra superpuesta al mapa que nos ha devuelto el servicio. En este ejemplo, la capa vectorial no añade mucha información, pero como se verá más adelante, estas capas vectoriales pueden mostrar una gran variedad de información que es lo que realmente añade un valor especial al mapa.

El siguiente paso en el desarrollo, no se centró en añadirle más complejidad a la representación del mapa, si no en permitir al usuario moverse por el mapa y cambiar el nivel de zoom, haciéndolo interactivo.

Fue en este paso donde se detectó el problema de diseño descrito anteriormente y se eliminó la clase NSThread, ya que hasta entonces, el mapa siempre realizaba una única petición al servicio WMS-OGC, pero para poder hacer que el usuario mueva el mapa, es necesario que se realicen nuevas peticiones y pintar los datos correspondientes a la nueva área visible del mapa. El problema de los *threads* o hilos era que usaban conexiones síncronas para descargar los datos, pero cada movimiento en el mapa, conlleva la creación de nuevos *threads* que esperan pacientemente a recibir la respuesta del servicio, incluso si los datos que esperan recibir están obsoletos porque el usuario ha vuelto a mover el mapa. Este problema también se generaba como consecuencia de utilizar un servicio WMS-OGC para pedir los datos del mapa, ya que estos servicios no se caracterizan por una especial rapidez a la hora de enviar los datos, porque cada vez que se realiza una petición tienen que construir la imagen a partir de capas que incluyen los diversos datos pedidos.

El problema no solo se encontraba en el incremento del número de *threads*, también radicaba en

que sus conexiones eran síncronas, y para poder visualizar el último *thread*, que era el que tenía la posición actual del mapa, había que esperar a que todos los anteriores terminaran de descargarse, lo que causaba que movimientos sucesivos del mapa retrasaran cada vez más su correcta visualización.

Una vez corregido este problema mediante el uso de conexiones asíncronas, que ya no creaban *threads* adicionales y además tenían la ventaja de poder ser canceladas si sus datos ya no eran relevantes para la correcta visualización del mapa al habernos movido antes de recibir la respuesta del servicio de mapas, ya no se producirán esperas innecesarias en la recepción de los datos cuando el usuario se mueva por el mapa. Sin embargo, los servicios WMS-OGC pueden resultar lentos a la hora de enviar los datos, principalmente porque necesitan construir las imágenes cada vez que se realiza una petición, y el movimiento del mapa no resulta lo suficientemente fluido como para que el usuario lo pueda utilizar con comodidad.

Para mejorar la eficiencia del mapa cuando se navega por él, se añadirá la posibilidad de ofrecer cartografía proveniente de servicios WMS-C. Este tipo de servicio se diferencia principalmente de los WMS-OGC en que ya dispone de una serie de *tiles* [4] predefinidos y no necesita construir las imágenes en cada petición, de este modo consigue una velocidad de respuesta mucho más elevada y por lo tanto el mapa se visualiza mucho más rápido. Como contra de estos servicios se puede destacar que sus imágenes se pre-generan únicamente para unas capas y niveles de zoom concretos, denominados *tileSets*. Por lo tanto, si se necesita una combinación de capas, que no este pre-generada, no se podrá utilizar este servicio para solicitarla.

Estos servicios, para poder almacenar imágenes de todo el mundo y a diferentes niveles de zoom, pre-generan una serie de *tiles* [4], una malla por cada nivel de zoom permitido, y suelen tener entre 10 y 20 niveles de zoom diferentes, de modo que no se puede realizar una petición de cualquier extensión, sino que la extensión de cada petición se corresponderá con la extensión de una de las *tiles* pre-generadas. Para poder calcular exactamente los parámetros de las peticiones, se tiene en cuenta la extensión máxima, el número de niveles de zoom y el tamaño de *tile* del servicio. Otra ventaja de los WMS-C es que el tamaño de los datos asociados a cada petición es menor en número, ya que se corresponden con una porción del mapa en vez de con todo el mapa. Esto conlleva que las conexiones duren menos, sean menos vulnerables a errores y dan la sensación de que el mapa se descarga antes, ya que se va pintando a porciones con cada *tile* descargada.

En las siguientes imágenes se puede ver el resultado de una petición a un servicio WMS-C y el efecto que se produce a la hora de ir pintando las *tiles* en el mapa.

Figura 2.6. Tile del WMS-C de IDEE-Base *Figura 2.7. Mapa con tiles sin cargar*

Otra de las mejoras asociadas a los servicios WMS-C queda patente al navegar por el mapa, conforme el usuario se mueve por él, las nuevas zonas se van descargando y pintando, pero esta vez no se realiza la petición del mapa completo, sino solamente de unas pocas *tiles* [4]. Esto es debido a que las *tiles* que ya hemos recibido previamente, quedan almacenadas en una cache local que almacena temporalmente un número determinado de *tiles* a las que ya se había accedido, de este modo, si movemos el mapa a un sitio anteriormente visitado, no será necesario realizar una nueva petición al servicio.

Para aprovechar al máximo los beneficios de la cache de *tiles* y la velocidad de respuesta de los servicios WMS-C, el área que se pide al servicio en realidad es mayor del área que queremos visualizar. Esto conlleva un incremento del número de *tiles* pedidas al servidor en la primera carga del mapa, pero reduce considerablemente las peticiones producidas al mover el mapa y crea la sensación de que los datos se descargan instantáneamente, porque antes de haber realizado el movimiento, ya disponíamos de la mayoría de los datos que se visualizarán una vez concluido el movimiento.

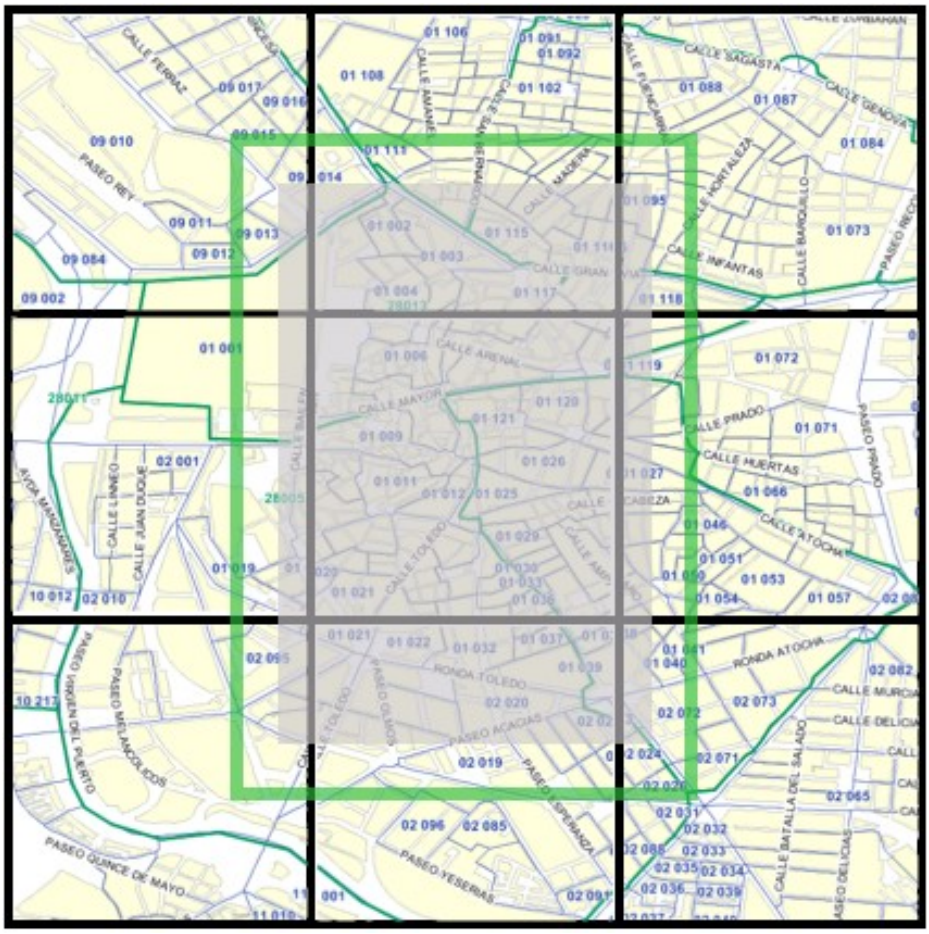

Figura 2.6. Efecto del tileado de mapas

La anterior imagen es una representación aproximada del efecto que se acaba de explicar. La zona gris sería la zona del mapa que queremos visualizar. La zona englobada por el rectángulo verde es la que pedimos al mapa, la diferencia de tamaño de ambas zonas es un parámetro configurable del Framework [12]. La malla negra sería el conjunto de *tiles* [4] que finalmente pedimos, que como puede observarse abarca mucha más área de la que queremos visualizar. En este caso concreto, las *tiles* que se hubieran pedido sin utilizar el rectángulo verde, hubieran sido las mismas, ya que la zona gris cubre al menos un pedazo de cada *tile* pedida. Pero en otros casos, evita que únicamente se pidan las cuatro *tiles* con las que se podría cubrir el área a visualizar.

Esta cache de *tiles*, no puede utilizarse con los servicios WMS-OGC debido a la gran flexibilidad de los parámetros de sus peticiones, que hace muy difícil comprobar si los datos ya habían sido pedidos con anterioridad. Este es otro de los motivos que hacen a los servicios WMS-OGC mucho menos eficientes que los WMS-C, ya que cada movimiento del mapa provoca que se descargue de nuevo el mapa entero.

Utilizando servicios WMS-C y los beneficios de la cache de *tiles*, la navegación por el mapa resulta suficientemente fluida como para que el usuario pueda utilizarlo con comodidad y no se provoquen esperas incomodas causadas por largas conexiones.

El último paso del desarrollo se centró en añadir diversas funcionalidades a la librería necesarias para cumplir con los objetivos del proyecto. Entre ellas, está la capacidad de utilizar el GPS del

iPhone para poder geolocalizar al usuario y representar su posición sobre el mapa o utilizarla con cualquier otro propósito, como hacer un seguimiento del recorrido realizado por ejemplo.

Otra de las funcionalidades es la posibilidad de hacer que los POIs [10] situados en el mapa reaccionen al evento de tacto, mostrando gráficamente si están seleccionados y se pueda realizar alguna acción, como puede ser mostrar información asociada a ese POI en alguna zona del interfaz. Para que esto sea posible, los POIs se representan sobre el mapa con algún tipo de imagen que está asociada a un punto vectorial y se denomina *marker*. Esta imagen puede estar en local o en una que apunte a una imagen.

En el segundo caso, se encontraron ciertos problemas similares a los que ocurrían con las *tiles* [4] cuando se descargaban con conexiones síncronas, para evitarlos se decidió descargar las imágenes asociadas a los POIs [10] de un modo similar a las *tiles*, creando un delegado que se encargara de descargarlas asíncronamente y de avisar cuando había concluido su descarga, pero no se consideró necesario utilizar algún tipo de cache, ni almacenar en local su imagen, debido a que estas imágenes son de un tamaño reducido y su descarga se realiza con suficiente velocidad como para no hacer necesarias estas acciones.

La última funcionalidad añadida al Framework, fue la capacidad de añadir datos vectoriales que estuvieran almacenados en un archivo con formato GeoJSON. Esta funcionalidad, permite añadir información vectorial compleja al mapa de un modo rápido y fácil. Estos archivos proporcionan la información necesaria para crear geometrías asociadas a unas coordenadas concretas. También se pueden añadir atributos a las geometrías, como puede ser nombre, descripción, el estilo con el que queremos representarla, etc. Es un tipo de archivo muy utilizado para transferir información geográfica de cualquier tipo, pero no es el único, XML o GML [15] son alternativas a GeoJSON y permitir su uso en la librería proporciona otra posible linea de trabajo futuro, ya que el modelo diseñado permite incorporar de forma sencilla nuevos formatos.

El desarrollo de esta última fase fue acompañado del desarrollo de la aplicación iRoutes [11], que hace uso de la mayoría de las funcionalidades de la librería y que pasaremos a explicar a continuación. Para mejorar la comunicación entre ambos componentes, se utilizó el patrón de diseño Facade [13], como ya se ha explicado anteriormente.

Los métodos más importantes que proporciona la clase Facade para permitir esta comunicación son los siguientes:

- initWithMap: Constructor del Facade.
- getVectorLayer: Devuelve la primera capa vectorial que esté agregada al mapa.
- addGeoJSON: Utiliza un archivo GeoJSON para crear una capa vectorial nueva y agregarla al mapa.
- addLayer: Agrega la capa indicada al mapa.
- removeLayer: Elimina la capa cuyo nombre coincida con identificador.
- getSelectedFeature: Devuelve la *feature* [14] que se haya seleccionada en este momento.

- selectNextFeature: Devuelve la *feature* siguiente a la seleccionada.
- selectPreviousFeature: Devuelve la *feature* anterior a la seleccionada.
- selectFeatureById: Devuelve la *feature* que coincida con identificador.
- centerExtentInFeatures: Centra la extensión del mapa en el conjunto de *features* que posee.
- refreshMap: Repinta el mapa.

Para una explicación más detallada de los métodos del Facade y de su uso, consultar el anexo D donde se encuentra el manual del Framework [12].

Al utilizar la librería en una aplicación más compleja, se fueron detectando y solucionando carencias que no se habían previsto cuando se utilizaba con una aplicación de prueba que simplemente mostraba el mapa por pantalla, lo que ayudó a incrementar la robustez de la librería al proporcionar un nuevo escenario de pruebas ideal.

2.5. Aplicación iRoutes

Figura 2.7. Menú principal iRoutes

Figura 2.8. Visor integrado en iRoutes

La aplicación iRoutes [11] es el segundo componente desarrollado en este PFC [1], con el principal objetivo de utilizar la mayoría de las funcionalidades que proporciona el Framework [12]

gsl_mobileMap_iPhone. Su implementación comenzó durante la última fase del desarrollo de la librería. Desde ese momento, ambos componentes evolucionaron en paralelo, si bien la gran mayoría de la funcionalidad del Framework ya había sido implementada.

Con un interfaz bastante sencillo, el usuario puede importar ficheros de rutas, previamente creadas en otra aplicación o de forma manual, que contienen un recorrido y puntos de interés con una breve descripción asociada a ellos. Estos ficheros contienen un formato especifico, denominado GeoJSON, que permite codificar una gran variedad de estructuras de datos geográficos. Su estructura base es la siguiente:

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
      "properties": {"prop0": "value0"}
      },
    { "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
          ]
        },
      "properties": {
        "prop0": "value0",
        "prop1": 0.0
        }
      },
    { "type": "Feature",
       "geometry": {
         "type": "Polygon",
         "coordinates": [
           [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
             [100.0, 1.0], [100.0, 0.0] ]
           ]
       },
       "properties": {
         "prop0": "value0",
         "prop1": {"this": "that"}
         }
       }
     ]
   }
```

Figura 2.9. Ejemplo de archivo GeoJSON

De su estructura puede adivinarse que muestra la información de una colección de *features* [14], estando cada una de estas *features* compuestas por geometrías de diversos tipos con sus correspondientes coordenadas y una lista de propiedades o atributos que puede almacenar información muy variada. Para una descripción más detallada de este formato, consultar las referencias web de la bibliografía. La elección de este formato entre todos los posibles está basada en su frecuente uso en todas las aplicaciones de visualización de la empresa GSL al estar ampliamente extendido en el entorno de las aplicaciones web.

Para poder cargar los datos contenidos en estos ficheros, es necesario un *parser* o analizador sintáctico que descodifique los datos y construya los objetos definidos en ellos. Esta tarea no resulta trivial, de modo que se consideró apropiado buscar algún Framework [12] que ya pudiera hacer este

trabajo.

Apple ha desarrollado su propio JSON Framework, pero es un Framework privado que no se incluye en el entorno de desarrollo de Objective C, quedando fuera de nuestro alcance. Por suerte, el uso de formatos de archivos similares a GeoJSON, como puede ser JSON, es muy común en la red y existen multitud de analizadores. Entre los enlaces de la bibliografía, puede encontrarse un estudio sobre los Frameworks más utilizados para este propósito, del que se ha extraído la siguiente imagen:

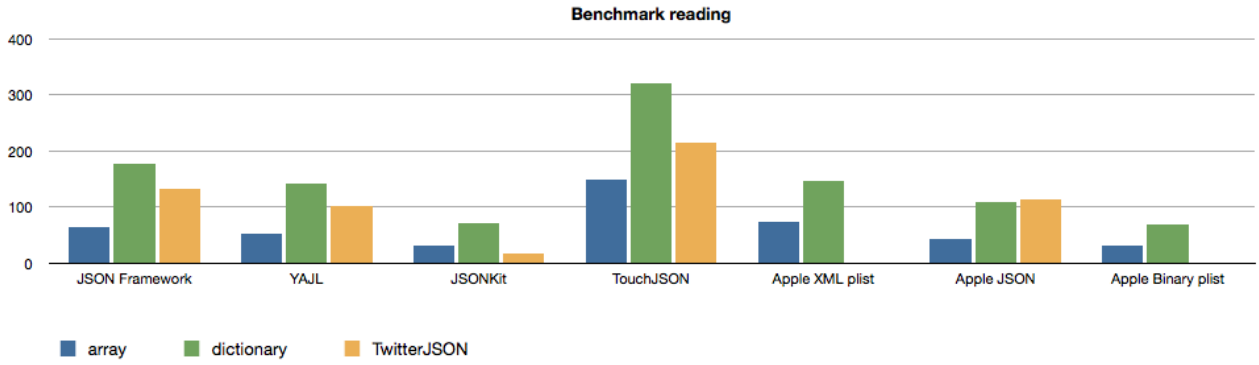

Figura 2.10. Tiempo de lectura de los diferentes Frameworks existentes. Fuente

En ella se puede observar el tiempo que tarda cada uno de los Frameworks [12] en leer un archivo y decodificar un *array* o vector, un diccionario de datos o un TwitterJSON, resultando ganador con clara ventaja el Framework JSONKit. Éste fue el seleccionado para ser utilizado en nuestro desarrollo por sus excelentes tiempos de decodificación y por poseer licencias BSD y Apache License 2.0, que permiten incluir y modificar el código, e incluso utilizarlo en software no libre, siempre que se respeten las cabeceras de las clases usadas. Gracias a esta librería, se podrán añadir capas vectoriales complejas al mapa de un modo muy simple.

Una vez el fichero que contiene la ruta esta importado en la aplicación, el usuario puede visualizar su información sobre el mapa. Para cargar la información de la ruta se utiliza el Facade [13], que se encarga de analizar el fichero, construir los objetos necesarios y asociarlos a una nueva capa vectorial que se añadirá a las ya existentes en el mapa. La interfaz de la aplicación cambiará, dejando de mostrar la lista de rutas importadas y mostrando el mapa con su extensión centrada en la información de la nueva capa vectorial.

Hecho esto, es posible navegar por el mapa o activar alguno de los controles que se le han añadido. Uno de estos controles nos permite seleccionar los POIs [10] del mapa tocándolos directamente sobre el mapa o navegando por ellos utilizando el interfaz y mostrará información asociada al POI seleccionado en la parte inferior de la interfaz. Otro de los controles que se pueden activar obtendrá la posición del usuario gracias al GPS del iPhone y dibujará una *feature* [14] o *marker* sobre el mapa, señalizando esta posición, que se irá actualizando mientras el control esté activo.

Estas funcionalidades de la librería pueden activarse a través de los botones que nos proporciona la interfaz de la aplicación.

El botón de información  , activa o desactiva el control para seleccionar los POIs. También se hacen visibles los botones laterales que permiten navegar por los POIs  . Por defecto, uno de los POIs de la ruta aparecerá como seleccionado y siempre habrá un POI seleccionado mientras el control esté activo. La librería nos permite definir los POIs como no seleccionables o no navegables, para evitar que un POI se pueda seleccionar directamente sobre el mapa o para evitar que se navegue por ese POI al utilizar las flechas. En la aplicación iRoutes [11], únicamente se ha utilizado esta funcionalidad para que el POI asociado a la posición del usuario (GPS) no sea seleccionable ni navegable.

El botón del GPS  , activa o desactiva el control que localiza al usuario, mientras este control este activo, el mapa se centrará en la posición del usuario y le situará sobre el mapa con el siguiente símbolo  .

Por ultimo, el botón central  , carga de nuevo el menú principal para poder seleccionar o importar otra ruta.

Toda la comunicación entre la aplicación y el Framework [12] se realiza a través de alguno de los métodos del Facade [13], abstrayendo del funcionamiento interno a cualquier desarrollador que quiera utilizar la librería y permitiendo que se pueda reutilizar en otras aplicaciones de un modo sencillo. En el anexo D se han incluido manuales que explican la forma de configurar y utilizar tanto el Facade como la aplicación desarrolladas. Estos manuales proporcionan información suficiente para poder utilizar el Framework como parte de cualquier aplicación, la forma en la que proporciona la vista del mapa y el modo de configurar el servicio de mapas.

Si no se utilizara este Facade [13], cualquier programador que quisiera utilizar la librería, debería examinar el comportamiento de muchas de las clases que la componen, perdiendo tiempo en comprender cómo funciona internamente el Framework [12].

Para concluir este apartado, se mostrará el aspecto de un fichero de una ruta concreta, del que se ha eliminado parte de la información geográfica y sustituido por puntos suspensivos debido a su gran extensión:

```
{
    "type": "FeatureCollection",
    "crs": {
        "type": "EPSG",
        "properties": {
            "code": "4326"
        }
    },
    "features": [
        {
            "type": "Feature",
            "properties": {
                "type": "route",
                "name": "Ruta Centro ZGZ",
                "description": "Ruta por el corazón de Zaragoza",
                "id": "route1",
                "distance": 4.333,
                "duration": 118.50,
                "style": {
                    "default": {
                    "strokeColor": "#2020FF",
                    "strokeWidth": 3,
                    "strokeOpacity": 0.8
                }
            }
        },
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [
                    -0.8797900000000003,
                    41.65542004200881
                ],
                [
                    -0.8804200000000003,
                    41.655730042008955
                ],
                ……,
                [
                    -0.8760100000000001,
                    41.65621004200925
                ]
            ]
        }
    },
    {
        "type": "Feature",
        "properties": {
            "type": "poi",
            "nane": "poiInicio",
            "id": "ini",
            "description": "Inicio de ruta",
            "style": {
                "default": {
                    "externalGraphic": "initial_POI_18x27.png",
                    "graphicWidth": 18,
                    "graphicHeight": 27
                }
            }
        },
```

```
        "geometry": {
            "type": "Point",
            "coordinates": [
                -0.8797900000000003,
                41.65542004200881
            ]
        }
    },
    {
        "type": "Feature",
        "properties": {
            "type": "poi",
            "name": "mypoi1",
            "id": "poi01",
            "description": "Mercado central",
            "style": {
                "default": {
                    "externalGraphic": "POI_market_25x29.png",
                    "graphicWidth": 25,
                    "graphicHeight": 29
                }
            }
        },
        "geometry": {
            "type": "Point",
            "coordinates": [
                -0.8828115463256836,
                41.65627274671068
            ]
        }
    },
    ……,
    }
]
}
```


Figura 2.11. Ejemplo de ruta descrita con formato GeoJSON

En este caso, la colección de *features* [14] o *FeatureCollection* esta compuesta por una ruta y un número indeterminado de POIs [10], pudiendo ser este igual a cero. Los componentes de la ruta y los POIs son los siguientes:

Una ruta consta de los siguientes atributos, además de la geometría:
- **type**: tipo de elemento, en éste caso siempre “route”.
- **id**: identificador de la ruta.
- **name**: nombre asociado a la ruta.
- **distance**: distancia recorrida por la ruta.
- **duration**: tiempo total en recorrer la ruta.
- **style**: objeto que define las propiedades de pintado de la línea (color, grosor, etc).

Un POI consta de los siguientes atributos:
- **type**: tipo de elemento, en este caso siempre “poi”.
- **id**: identificador del POI.
- **name**: nombre del POI (puede ser visualizado).
- **description**: descripción del POI (puede ser visualizado).
- **style**: objeto que define las propiedades de pintado del punto (icono, tamaño, etc).

Capítulo 3. Conclusiones

En este capítulo se analiza el grado de cumplimiento de los objetivos marcados al inicio, mostrando los resultados obtenidos. Además, se propondrán líneas futuras de trabajo.

3.1. Resultados obtenidos

Como resultado de este PFC [1] se ha obtenido un Framework [12] que posibilita la visualización tanto de mapas compatibles con los estándares WMS-OGC y WMS-C, como de todo tipo de datos vectoriales, permitiendo usar la pantalla táctil del iPhone para movernos por el mapa y su GPS para geolocalizar al usuario sobre el mapa. También una aplicación de visualización de rutas, gracias a la cual queda demostrada la facilidad de integración de visores de mapas basados en la librería como parte de otras aplicaciones.

El aspecto final de los componentes desarrollados es el siguiente:

Figura 3.1. iRoutes: importando

Figura 3.2. iRoutes: menú

Figura 3.3. iRoutes: visor

Las dos imágenes de la izquierda son parte de la interfaz de la aplicación iRoutes [11], concretamente la pantalla para importar una ruta que se encuentre en una URL y el menú principal, donde se listan las rutas importadas. La tercera imagen muestra la visualización de una ruta sobre el mapa del servicio WMS-C de IDEZAR.

Uno de los objetivos principales de este PFC [1] era crear un Framework [12] que permitiera la visualización de servicios WMS-OGC y WMS-C. Para demostrar que se ha cumplido este objetivo, se van a mostrar capturas de diferentes rutas en las que se han utilizado diversos servicios de mapas de ambos tipos o incluso combinados.

Las dos primeras imágenes muestran dos rutas distintas por la ciudad de Zaragoza, utilizando de nuevo el servicio WMS-C de IDEZAR. Éste servicio solo tiene información de Zaragoza y no muestra información fuera de la zona suministrada.

Figura 3.4. iRoutes: Ruta centro de Zaragoza

Figura 3.5. iRoutes: Ruta por los Galachos

Las siguientes imágenes muestran una zona céntrica de Madrid. Los mapas pertenecen a IDEE-BASE en la izquierda y a Cartociudad en la derecha. Ambos son servicios WMS-C pero su estilo de representación de la información es muy diferente. Cartociudad incluye nombres de calles, códigos postales y en niveles más cercanos de zoom, se muestran incluso los números de todos los edificios.

Figura 3.6. iRoutes: Ruta por Madrid

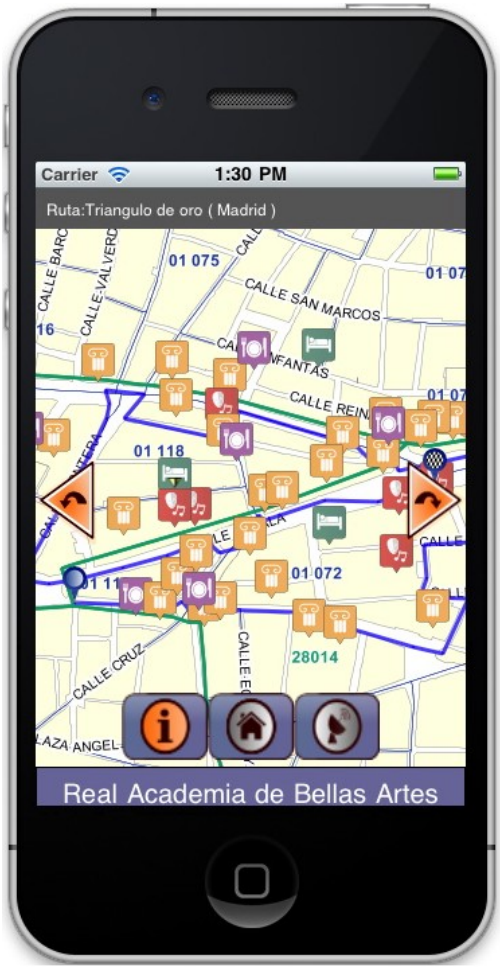

Figura 3.7. iRoutes: Ruta por Madrid 2

Se han utilizado estos dos servicios porque se ofrecen desde la web del Instituto Geográfico Nacional, y son de uso común en el mundo de las IDEs en España. La elección del servidor a utilizar dependerá del uso que pretendan tener las aplicaciones que lo usen. Sin embargo, no es estrictamente necesario elegir entre un servicio u otro, ya que podemos utilizar combinaciones de capas provenientes de servicios diferentes. A modo de ejemplo, se ha probado a añadir las siete capas que ofrece el WMS-OGC de Cartociudad y una capa de hidrografía del WMS-OGC de IDEE-BASE sobre los servicios WMS-C de IDEZAR por un lado y de IDEE-BASE por otro, siendo estos los resultados.

Figura 3.8. iRoutes: prueba capas 1

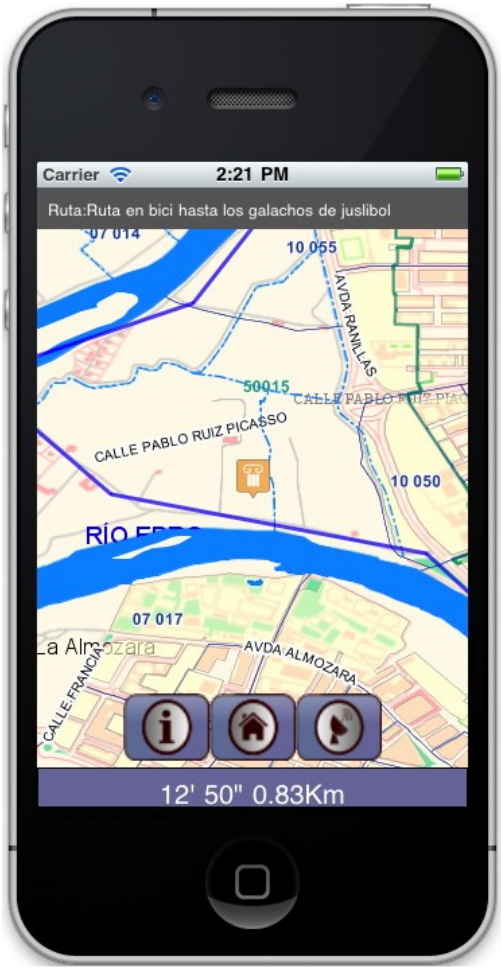

Figura 3.9. iRoutes: prueba capas 2

El objetivo de este ejemplo es exponer la flexibilidad que tiene el visor para mostrar diferentes combinaciones de los servicios soportados. Por el modo en el que se configuran los servidores WMS-C, resulta más difícil utilizar dos o más al mismo tiempo. Sería necesario que coincidieran en el número de niveles de zoom, extensión máxima, resoluciones y algún parámetro más. En cambio, se pueden superponer tantos servidores WMS-OGC como se desee, ya que ofrecen menos restricciones.

En las ultimas imágenes puede verse la mejora que supone usar un buen conjunto de capas que termine dando como resultado un mapa con información útil.

Figura 3.10. iRoutes: usando IDEZar

Figura 3.11. iRoutes: IDEZar y Cartociudad

Por ahora, la aplicación iRoutes [11] está diseñada para que utilice siempre una única combinación de servicios y capas que se define en tiempo de codificación, quedando como trabajo futuro dar la posibilidad al usuario de cambiar los servicios y seleccionar las capas con las que prefiera visualizar el mapa, una vez se esté ejecutando la aplicación.

Los objetivos que se habían propuesto para este proyecto han sido alcanzados satisfactoriamente, dentro de los plazos establecidos y dando como resultado un software eficiente, algo muy importante en el mundo de las aplicaciones para móviles donde la autonomía esta restringida por una escasa batería.

Algunos datos referentes a la eficiencia de los componentes desarrollados pueden consultarse en el anexo C de esta memoria, del cual se han extraído las siguientes gráficas.

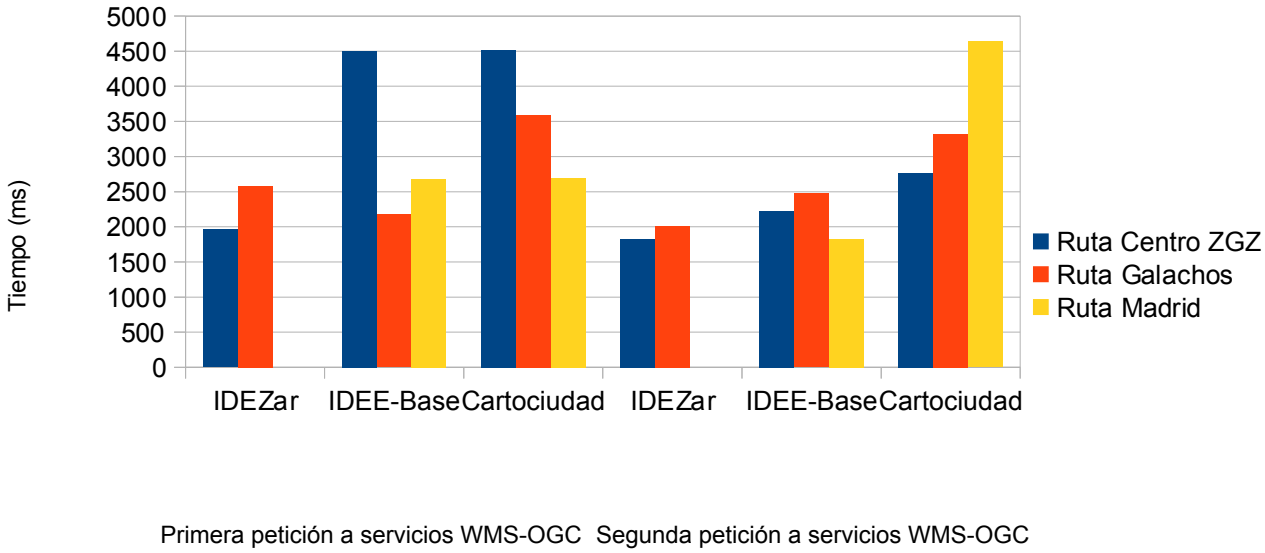

Figura 3.12. Tiempos de respuesta de servicios WMS-OGC

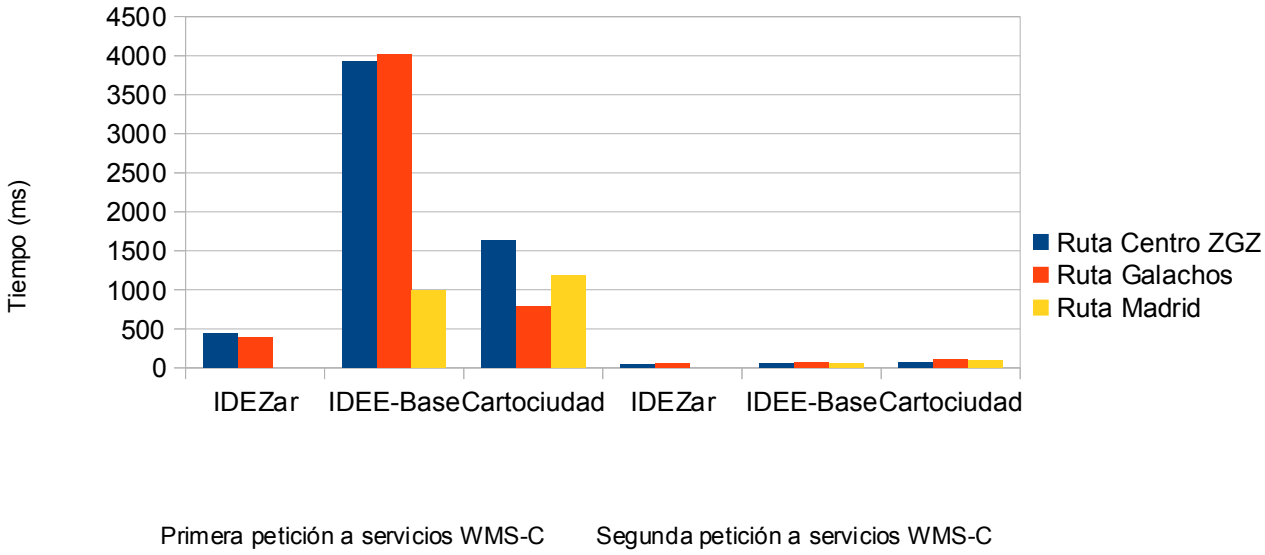

Figura 3.13. Tiempos de respuesta de servicios WMS-C

Como puede observarse en las gráficas, por lo general, los servicios WMS-C tardan menos en responder a las peticiones, aunque dependiendo del área pedida y del servicio utilizado, pueden dar peor resultado que los WMS-OGC, como ocurre con el servicio IDEE-Base al pedir el área de la segunda ruta. Sin embargo, cuando se realiza una segunda petición de un área que ya había sido visitada, la mejora obtenida es drástica.

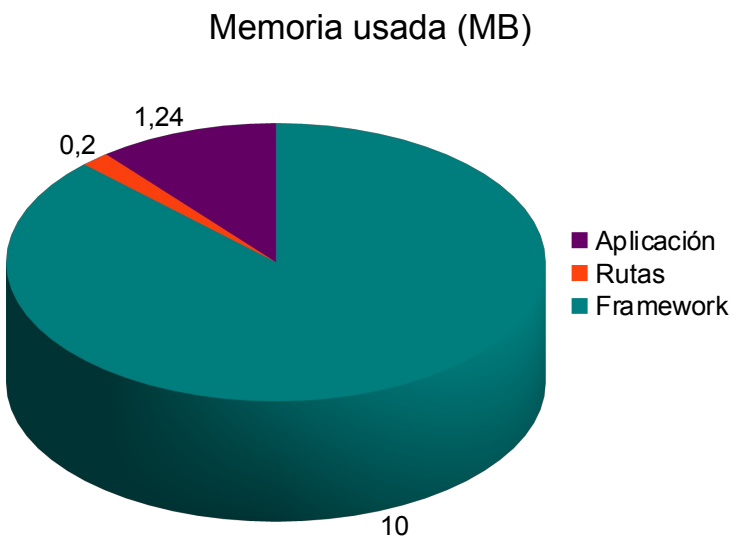

Figura 3.14. Relación del uso de memoria de los diferentes componentes

La gráfica anterior muestra, a modo de resumen, la cantidad de memoria utilizada relativa a cada componente de la aplicación. Para ser exacto, las partes relativas a las rutas y al Framework son variables, siendo su relación de 0.01 a 0.05 MB por ruta y de 1.5 a 10 MB dependiendo del mapa.

Estos datos son bastante positivos en cuanto a eficiencia se refiere, algo muy importante en el mundo de las aplicaciones para móviles donde la memoria es limitada y la autonomía esta restringida por una escasa batería.

3.2. Líneas futuras

Partiendo del sistema desarrollado, pueden tomarse múltiples vías de desarrollo que mejoren o añadan más funcionalidades. A continuación se muestra una lista de posibilidades:

- Mejoras relativas a la visualización de rutas y POIs [10]
  - Mostrar información más detallada en otra ventana al seleccionar un POI
  - Visualizar imágenes asociadas a las rutas y los POIs
  - Reproducir audios y videos asociados a las rutas y POIs

- Mejoras relacionadas con la gestión de las rutas
  - Permitir al usuario hacer una búsqueda de rutas
  - Permitir al usuario crear una ruta usando posiciones calculadas con el GPS
  - Importar rutas en otros formatos de archivos vectoriales: GeoRSS, KML, GML [15]

- Mejoras del visualizador
  - Poder transformar las coordenadas de las peticiones que se realizan a los servicios de mapas y/o los datos vectoriales representados para superponer en un mismo mapa capas de servicios de mapas que ofrecen distintos sistemas de referencia.
  - Añadir herramientas de navegación avanzadas: ir a extensión anterior y posterior, leyendas, volver a la extensión inicial, mediciones, guardar imagen o contexto, etc.

  - Acceso a mapas proporcionados por la API de GoogleMaps
  - Permitir al usuario seleccionar el servicio con el que visualizar los mapas y las combinaciones de capas que le interesen
  - Acceso *offline* a caché de *tiles* [4] generada previamente, basada en la especificación WMS-C de OSGeo [3] y almacenada en la SD o memoria interna del teléfono
  - Acceso a servicios de mapas basados en otros estándares, como WMTS de OGC

- Mejoras generales
  - Poder orientar el mapa hacia el Norte usando los sensores del iPhone a modo de brújula

Capítulo 4. Descripción de los anexos

En este capítulo se describe el contenido de los anexos que acompañan a la memoria de este PFC [1].

Anexo A. Análisis. Estudio del problema a abordar con una clara definición de requisitos.

Anexo B. Diseño. Modelos de cada uno de los componentes del sistema detallando las decisiones de diseño adoptadas.

Anexo C. Pruebas. Definición de los casos de prueba y resultados de las pruebas realizadas.

Anexo D. Manuales. Explicación del funcionamiento del sistema desarrollado.

Anexo E. Servicios Web Estándar. Explicación del funcionamiento de los servicios de mapas utilizados.

Anexo A. Análisis

En este apéndice se detallará de forma exhaustiva las características que deberá cumplir el sistema una vez finalizado el proyecto. En primer lugar se definirán los requisitos funcionales y no funcionales exigidos y, posteriormente, se detallarán los casos de uso del sistema.

A.1. Requisitos

A continuación se listan las condiciones de funcionalidad y rendimiento que debe cumplir el software desarrollado.

A.1.1. Requisitos funcionales

Los requisitos funcionales definen las principales funcionalidades que debe poseer el software desarrollado. Al tratarse de un sistema con una clara división en componentes, conocida a priori, estos requisitos se dividen entre el Framework [12] y la aplicación de rutas.

Framework:

- **RF-1:** Acceso *online* a WMS basados en especificaciónWMS-OGC v1.0.0 o superior
- **RF-2:** Acceso *online* a WMS-C basados en especificación WMS-C de OSGeo [3]
- **RF-3:** Herramientas básicas de navegación
  - **RF-3.1:** Zoom para acercar
  - **RF-3.2:** Zoom para alejar
  - **RF-3.3:** mover mapa (panning)
- **RF-4:** Visualizar rutas en el visor de mapas a partir de formato vectorial GeoJSON
- **RF-5:** Visualizar puntos de interés (POIs [10]) sobre el mapa a parir de formato vectorial GeoJSON
- **RF-6:** Ofrecer Facade de comunicación con otros componentes
  - **RF-6.1:** Método para pasarle al visor una ruta inicial a visualizar
  - **RF-6.2:** Método para pasarle al visor el fichero de puntos de interés a visualizar
  - **RF-6.3:** Método para resaltar un POI
  - **RF-6.4:** Método para eliminar rutas cargadas en el visor
- **RF-7:** Geoposicionamiento por GPS
  - **RF-7.1:** Ofrecer interfaz para seleccionar si se quiere utilizar el geoposicionamiento por GPS
  - **RF-7.2:** Dibujar en el mapa la posición GPS

RF-7.3: Actualizar la posición GPS cada X tiempo

- **RF-8:** Poder seleccionar POI haciendo clic sobre su marcador en el mapa
- **RF-9:** Permitir definir POIs no navegables pero si seleccionables con clic
- **RF-10:** Acceso *offline*/*online* a fichero de rutas y fichero de POIs
- **RF-11:** Integración visor de mapas en aplicación principal

Aplicación de rutas:

- **RF-1:** Resaltar POIS al moverse con los botones laterales o hacer clic sobre él
- **RF-2**: Mostrar información del POI en la parte inferior de la ventana al resaltarlo
- **RF-3**: Guardar el listado de las rutas al cerrar la aplicación, para que al salir y volver a entrar muestre siempre el estado anterior

A.1.2. Requisitos no funcionales

Son los siguientes:

- **RNF-1:** Soporte en sistema operativo iOS de iPhone
- **RNF-2:** Interfaz clara e intuitiva
- **RNF-3:** El visor deberá poder integrarse fácilmente en otras aplicaciones para iPhone
- **RNF-4:** Facilitar configuración estética del visor

A.2. Casos de uso

Mediante esta técnica de captura de requisitos potenciales se define la comunicación y el comportamiento de los distintos componentes del sistema ante la interacción con usuarios y/u otros sistemas. Para ello detallaremos los casos de uso, distinguiendo entre los diferentes componentes que conforman el sistema.

Framework gsl_mobileMap_iPhone:

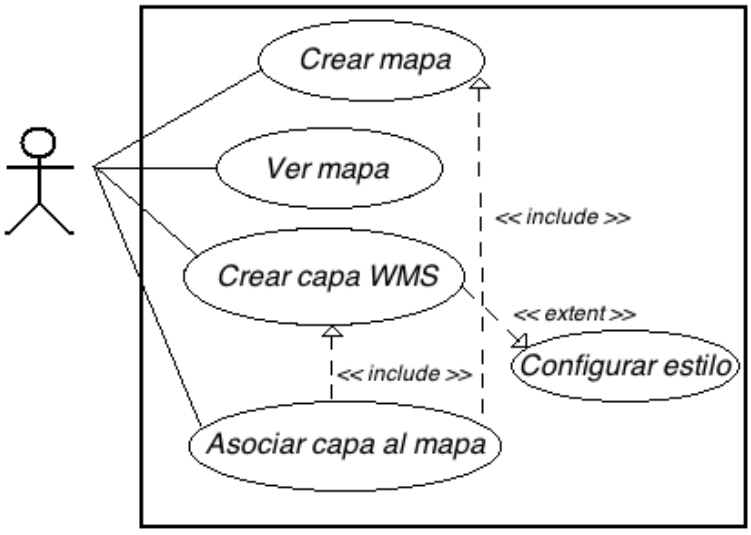

Figura A.1. Caso de uso: Mapa con capas WMS-OGC

Caso de uso “Crear mapa”:

- Descripción: crear el mapa y configurar parámetros.
- Actores: ninguno.
- Precondición: -
- Flujo:
  i. Se crea el mapa base.
  ii. Se añade el servicio de mapas que nos proporcionara las imágenes del mapa.
  iii. Se añade el resto de parámetros necesarios para su correcta visualización.
- Postcondición: el mapa está creado y listo para añadirle datos.

Caso de uso “Ver mapa”:

- Descripción: visualizar el mapa.
- Actores: ninguno.
- Precondición: el mapa está creado.
- Flujo:
  i. Se asigna el mapa a un contexto.
- Postcondición: el mapa se visualiza en el contexto asignado.

Caso de uso “Crear capa WMS”:

- Descripción: crear una capa que obtiene los datos de un servicio WMS-OGC.
- Actores: ninguno.
- Precondición: -
- Flujo:
  i. Se definen los valores de la petición WMS-OGC.
  ii. Se crea una capa con los valores especificados.
- Postcondición: la capa está creada y lista para ser asociada al mapa.

Caso de uso “Configurar estilo de la capa”:

- Descripción: configurar el estilo con el que se visualizará la capa WMS-OGC.
- Actores: ninguno.
- Precondición: capa WMS-OGC creada
- Flujo:
  i. Se definen los nuevos valores de estilo con los que queremos representar la capa WMS-OGC.
  ii. Se añaden los nuevos valores a la capa WMS-OGC.
- Postcondición: la capa WMS-OGC se visualizara con el nuevo estilo definido.

Caso de uso “Asociar capa al mapa”:

- Descripción: añadir la capa seleccionada al mapa.
- Actores: ninguno.
- Precondición: capa WMS-OGC creada y mapa creado
- Flujo:

   i. Se añade la capa WMS-OGC a las capas que ya pertenecían al mapa.
- Postcondición: el mapa se visualizara con la nueva capa superpuesta a las ya existentes.

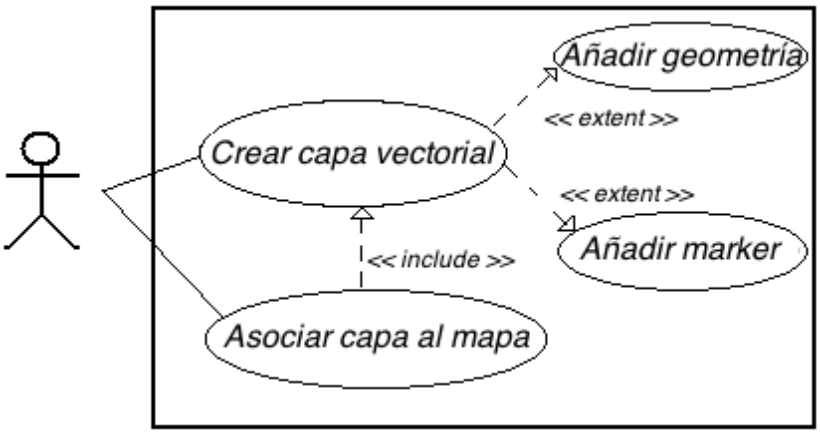

Figura A.2. Caso de uso: Mapa con capas Vectoriales

Caso de uso "Crear capa vectorial":

- Descripción: crear una capa con datos vectoriales.
- Actores: ninguno.
- Precondición: el mapa está creado.
- Flujo:
   i. Se crea la capa vectorial.
- Postcondición: la capa vectorial está creada.

Caso de uso "Añadir geometría":

- Descripción: añadir una geometría a la capa.
- Actores: ninguno.
- Precondición: la capa está creada.
- Flujo:
   i. Se añaden los datos de la geometría a la capa vectorial.
- Postcondición: la capa contiene los datos de la geometría.

Caso de uso "Añadir Marker":

- Descripción: añadir un *marker* a la capa.
- Actores: ninguno.
- Precondición: la capa está creada.
- Flujo:
   i. Se añaden los datos del *marker* a la capa vectorial.
- Postcondición: la capa contiene los datos del *marker*.

Caso de uso "Asociar capa al mapa":

- Descripción: añadir la capa vectorial al mapa.

- Actores: ninguno.
- Precondición: capa vectorial creada y mapa creado
- Flujo:
  i. Se añade la capa vectorial a las capas que ya pertenecían al mapa.
- Postcondición: el mapa se visualizará con la nueva capa superpuesta a las ya existentes.

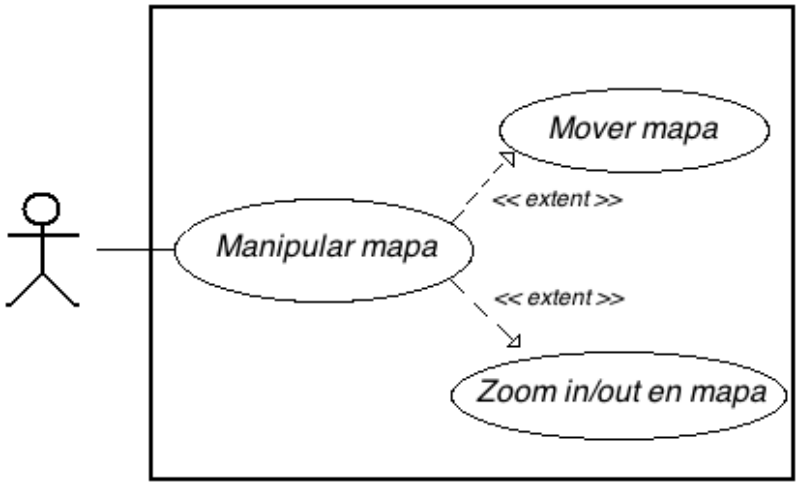

Figura A.3. Caso de uso: Movimiento del mapa

Caso de uso "Manipular mapa":

- Descripción: manipular la visión del mapa.
- Actores: usuario.
- Precondición: el mapa está siendo visualizado.
- Flujo:
  i. Se detecta la manipulación del mapa por parte del usuario.
  ii. Se comprueba el tipo de movimiento realizado.
  iii. La librería realiza los cambios requeridos en el mapa.
- Postcondición: el mapa muestra los nuevos datos correspondientes a la manipulación del mapa.

Caso de uso "Mover mapa":

- Descripción: mover zona representada por el mapa.
- Actores: usuario.
- Precondición: el mapa está siendo visualizado y se ha detectado un movimiento de *panning* sobre el mapa.
- Flujo:
  i. Se detecta un movimiento del mapa por parte del usuario.
  ii. La librería modifica la extensión del mapa.
  iii. La librería refresca la información del mapa en base a la nueva extensión
- Postcondición: el mapa muestra los nuevos datos correspondientes al movimiento del mapa.

Caso de uso "Zoom in/out en mapa":

- Descripción: realizar zoom en el mapa.
- Actores: usuario.
- Precondición: el mapa está siendo visualizado y se ha detectado un movimiento de zoom sobre el mapa.
- Flujo:
  i. Se detecta un movimiento de zoom por parte del usuario.
  ii. La librería modifica la extensión y el nivel de zoom del mapa.
  iii. La librería refresca la información del mapa en base a los nuevos datos.
- Postcondición: el mapa muestra los nuevos datos correspondientes al zoom del mapa.

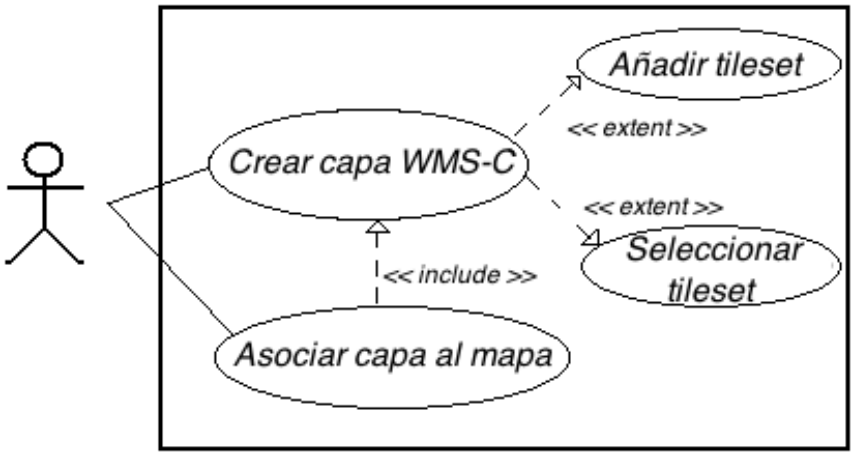

Figura A.4. Caso de uso: Mapa con capas WMS-C

Caso de uso "Crear capa WMS-C":

- Descripción: crear una capa que obtiene los datos de un servicio WMS-C.
- Actores: ninguno.
- Precondición: el mapa está creado.
- Flujo:
  i. Se crea la capa WMS-C.
- Postcondición: la capa WMS-C está creada.

Caso de uso "Añadir tileset":

- Descripción: añadir un tileset a la capa.
- Actores: ninguno.
- Precondición: la capa está creada.
- Flujo:
  i. Se añade una configuración de *tiles* a la capa.
- Postcondición: la capa contiene una nueva configuración de *tiles*.

Caso de uso "Seleccionar tileset":

- Descripción: seleccionar uno de los tilesets de la capa.
- Actores: ninguno.
- Precondición: la capa está creada.
- Flujo:

   i. Se selecciona una de las configuraciones de *tiles* asociadas a la capa.
- Postcondición: la capa representa su información con la nueva configuración

Caso de uso "Asociar capa al mapa":

- Descripción: añadir la capa WMS-C al mapa.
- Actores: ninguno.
- Precondición: capa WMS-C creada y mapa creado
- Flujo:
   i. Se añade la capa WMS-C a las capas que ya pertenecían al mapa.
- Postcondición: el mapa se visualizará con la nueva capa superpuesta a las ya existentes.

Aplicación iRoutes:

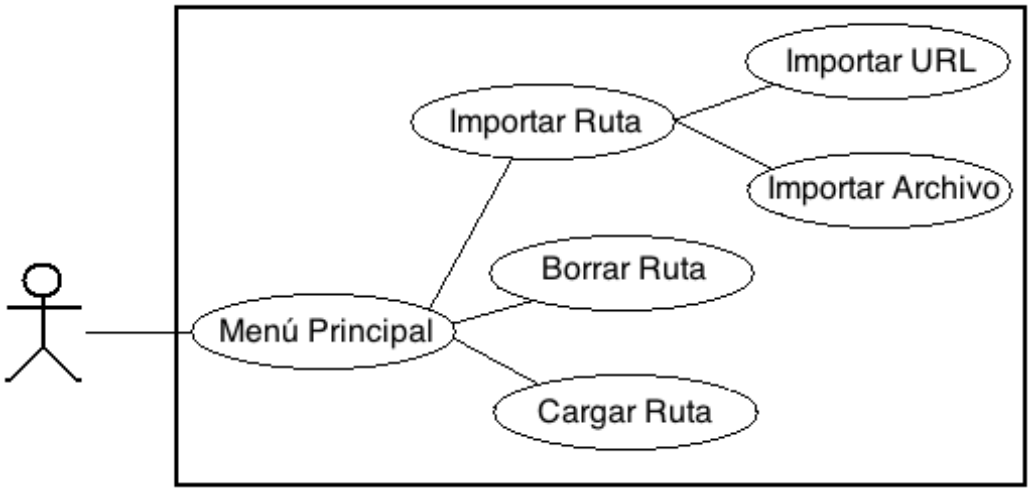

Figura A.5. Caso de uso: aplicación iRoutes

Caso de uso "Menú principal":

- Descripción: mostrar el menú principal de la aplicación.
- Actores: usuario.
- Precondición: ninguna.
- Flujo:
   i. El usuario ha ejecutado la aplicación o pulsado el botón de menú principal.
   ii. La aplicación carga la pantalla principal de la aplicación con la lista de rutas.
- Postcondición: la aplicación está cargada y muestra la pantalla principal.

Caso de uso "Importar ruta":

- Descripción: importar una nueva ruta.
- Actores: usuario.
- Precondición: aplicación cargada.
- Flujo:
   i. El usuario pulsado el botón importar.

   ii. La aplicación da a elegir al usuario si importar una ruta desde un archivo o usando una .
- Postcondición: la aplicación queda en espera de la elección del usuario.

Caso de uso “Importar URL”:

- Descripción: importar una nueva ruta a partir de un archivo.
- Actores: usuario.
- Precondición: usuario ha pulsado el botón importar.
- Flujo:
   i. El usuario elige importar usando una URL.
   ii. La aplicación muestra el interfaz para importar desde una URL.
   iii. El usuario introduce la URL.
   iv. La aplicación descarga el archivo de la URL, si su formato es correcto, lo añade a la lista, en caso contrario muestra mensaje de error.
- Postcondición: la aplicación muestra el menú principal con la lista de rutas actualizada.

Caso de uso “Importar Archivo”:

- Descripción: importar una nueva ruta a partir de una URL.
- Actores: usuario.
- Precondición: usuario ha pulsado el botón importar.
- Flujo:
   i. El usuario elige importar un archivo existente en disco.
   ii. La aplicación muestra los archivos disponibles en disco.
   iii. El usuario selecciona uno de los archivos.
   iv. La aplicación comprueba el formato del archivo y si es correcto, lo añade a la lista, en caso contrario muestra mensaje de error.
- Postcondición: la aplicación muestra el menú principal con la lista de rutas actualizada.

Caso de uso “Borrar Ruta”:

- Descripción: eliminar ruta de la lista de rutas.
- Actores: usuario.
- Precondición: una ruta está seleccionada.
- Flujo:
   i. El usuario pulsa el botón borrar ruta.
   ii. La aplicación pide confirmación al usuario.
   iii. El usuario confirma que quiere borrar la ruta seleccionada.
   iv. La aplicación elimina la ruta seleccionada de la lista de rutas.
- Postcondición: la aplicación muestra el menú principal con la lista de rutas actualizada.

Caso de uso “Cargar Ruta”:

- Descripción: carga la información de la ruta en el mapa.
- Actores: usuario.
- Precondición: una ruta está seleccionada.
- Flujo:
   i. El usuario pulsa el botón para cargar la ruta.

ii. La aplicación sale del menú principal y carga la vista del mapa, los botones de la interfaz del mapa y añade la información de la ruta al mapa.

- Postcondición: la aplicación muestra el interfaz del mapa con la información de la ruta cargada sobre él.

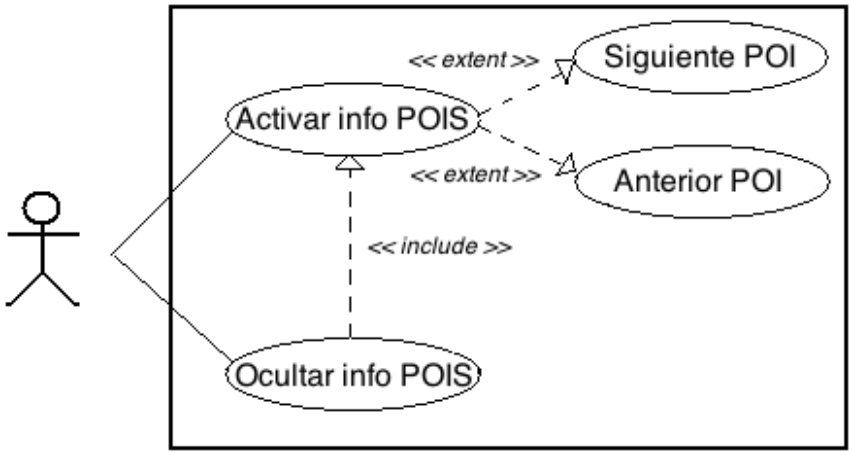

Figura A.6. Caso de uso: activar/desactivar información

Caso de uso "Activar info POIs":

- Descripción: activar la visualización de información asociada a los POIs [10].
- Actores: usuario.
- Precondición: la aplicación está mostrando un mapa.
- Flujo:
  i. El usuario pulsa el botón información.
  ii. La aplicación activa los *markers* del mapa, haciéndolos seleccionables y añade dos nuevos botones a la interfaz que permiten navegar pos los POIs.
  iii. Uno de los *markers* queda seleccionado.
- Postcondición: la aplicación muestra la información del *marker* que está seleccionado en la sección correspondiente del interfaz.

Caso de uso "Siguiente POI":

- Descripción: seleccionar el siguiente POI.
- Actores: usuario.
- Precondición: la aplicación está mostrando un mapa y el botón de información permanece activo.
- Flujo:
  i. El usuario pulsa el botón que selecciona el siguiente POI.
  ii. El siguiente *marker* queda seleccionado.
- Postcondición: la aplicación muestra la información del *marker* que está seleccionado en la sección correspondiente del interfaz.

Caso de uso "POI Anterior":

- Descripción: seleccionar el POI anterior.
- Actores: usuario.

- Precondición: la aplicación está mostrando un mapa y el botón de información permanece activo.
- Flujo:
  i. El usuario pulsa el botón que selecciona el POI anterior.
  ii. El anterior *marker* queda seleccionado.
- Postcondición: la aplicación muestra la información del *marker* que está seleccionado en la sección correspondiente del interfaz.

Caso de uso "Ocultar info POIs":

- Descripción: desactivar la visualización de información asociada a los POIs.
- Actores: usuario.
- Precondición: la aplicación está mostrando un mapa y el botón de información permanece activo.
- Flujo:
  i. El usuario pulsa el botón información.
  ii. La aplicación elimina los botones para navegar por los POIs, desactiva los *markers* para que no puedan seleccionarse y no muestra ninguna información asociada a ellos.
- Postcondición: la aplicación deja de mostrar información de los *markers*.

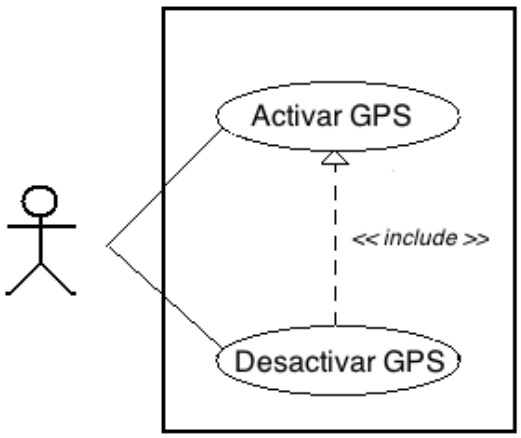

Figura A.7. Caso de uso: Caso de uso: activar/desactivar GPS

Caso de uso "Activar GPS":

- Descripción: activar el uso del GPS.
- Actores: usuario.
- Precondición: la aplicación está mostrando un mapa.
- Flujo:
  i. El usuario pulsa el botón del GPS.
  ii. La aplicación añade un *marker* al mapa señalando la posición del usuario sobre el mapa.
  iii. Mientras esté activo, la posición del usuario se irá actualizando cuando se mueva.
- Postcondición: la aplicación muestra el interfaz del mapa y la posición del usuario sobre él.

Caso de uso "Desactivar GPS":

- Descripción: desactivar el uso del GPS.

- Actores: usuario.
- Precondición: la aplicación está mostrando un mapa y el GPS está activo.
- Flujo:
  i. El usuario pulsa el botón del GPS.
  ii. La aplicación elimina el *marker* que señalaba la posición del usuario sobre el mapa.
- Postcondición: la aplicación muestra el interfaz del mapa.

Anexo B. Diseño

En este anexo se mostrará el diseño del sistema, centrándonos en cada uno de sus componentes. Para ello se detallarán los modelos de datos obtenidos, las decisiones de diseño tomadas y sus motivaciones. Además, en el caso del Framework [12], se detallarán las diferentes fases que fueron dando forma al modelo, al haber sido basado su desarrollo en una serie de fases incrementales.

B.1. Framework gsl_mobileMap_iPhone

Este componente es el encargado de representar la visualización del mapa, crear sus controles y toda su estructura interna. Su diseño se realizó en 5 fases o iteraciones que fueron incrementando sus funcionalidades hasta cumplir con los objetivos deseados. Por tanto en este anexo se presentarán todos los modelos de clases creados, desde el más simple, correspondiente a la versión del visualizador con funcionalidad básica, hasta el modelo correspondiente al visualizador con todas sus funcionalidades.

B.1.1. Modelo estático

El modelado estático se ocupa de establecer, en el paradigma de la orientación a objetos, las clases que forman un sistema y sus atributos, métodos y relaciones.

El diagrama del modelo de la versión más básica del visualizador, que únicamente soportaba capas de mapas de una única *tile* [4] de servicios WMS-OGC, es el siguiente:

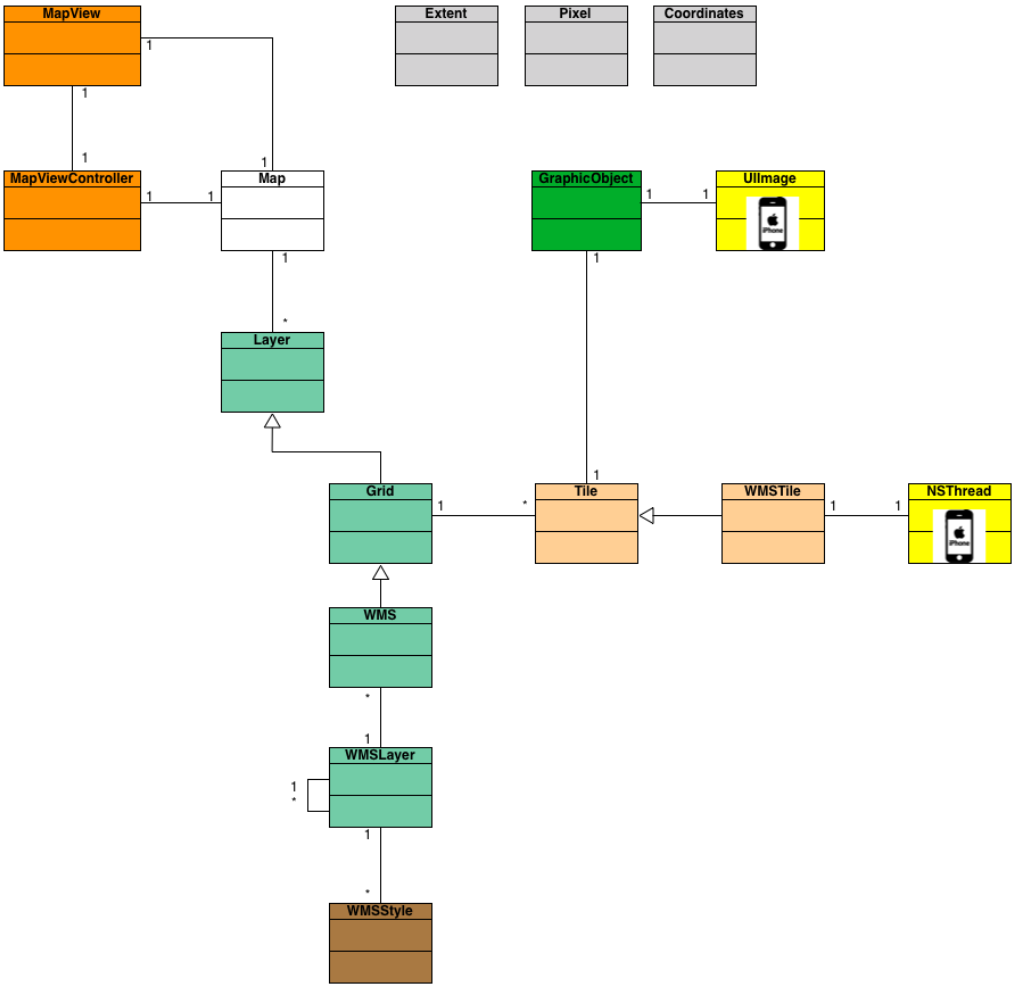

Figura B.1. Diagrama de clases de la primera iteración

Los colores de las clases del diagrama están relacionados con la funcionalidad asociada a cada clase, siendo estas las relaciones:

- Blanco (Map): proporcionan los métodos de creación de las clases principales del Framework [12]
- Naranja (MapView): clases relacionadas con la visualización de los datos
- Verde claro (Layer): clases relacionadas con las capas
- Naranja claro (Tile): clases que definen los elementos básicos de las capas
- Verde (GraphicObject): clases que almacenan las información gráfica de otros elementos
- Amarillo (UIImagen): clases propias del lenguaje Objective-C
- Marrón (WMSStyle): clases que definen los estilos de representación de otras clases
- Gris (Extent): son clases que definen tipos de datos básicos, extensión, pixel y coordenadas, son usadas por muchas de las clases y no se ha dibujado su relación con ellas para no estropear la claridad de los diagramas.

Sus clases principales son: *Map*, *Layer*, *Grid*, *WMS*, *Tile* y *WMSTile*.

- *Map*: es la clase principal en torno a la que sea crea cualquier visualizador de mapas e incluye tanto las distintas capas que lo componen como las herramientas para su manejo.
- *Layer*: representa una capa de alguno de los tipos de datos implementados.
- *Grid*: representa una *Layer* formada como una cuadrícula de *tiles* [4]. Es la estructura sobre

la que se sostienen los *tiles* e incorpora los métodos para su administración, pero no los datos en sí.

- *WMS*: es la clase que especifica los parámetros de la *Layer* y la encargada de construir las peticiones que posteriormente realizaran las *tiles* al servicio.
- *Tile*: es la unidad básica de una capa *tileada* y contiene los datos necesarios para realizar la petición de una *tile*.
- *WMSTile*: tipo de *tile* cuya información proviene de un servicio de mapas.

A esta versión básica, se añadió la posibilidad de crear capas que estuvieran formadas por datos vectoriales en vez de datos provenientes de servicios de mapas, lo que permite incorporar información muy diversa a la representación del mapa. Para ello se añadieron las clases *Vector*, *Feature*, *Marker* y una colección de diversas *Geometries*.

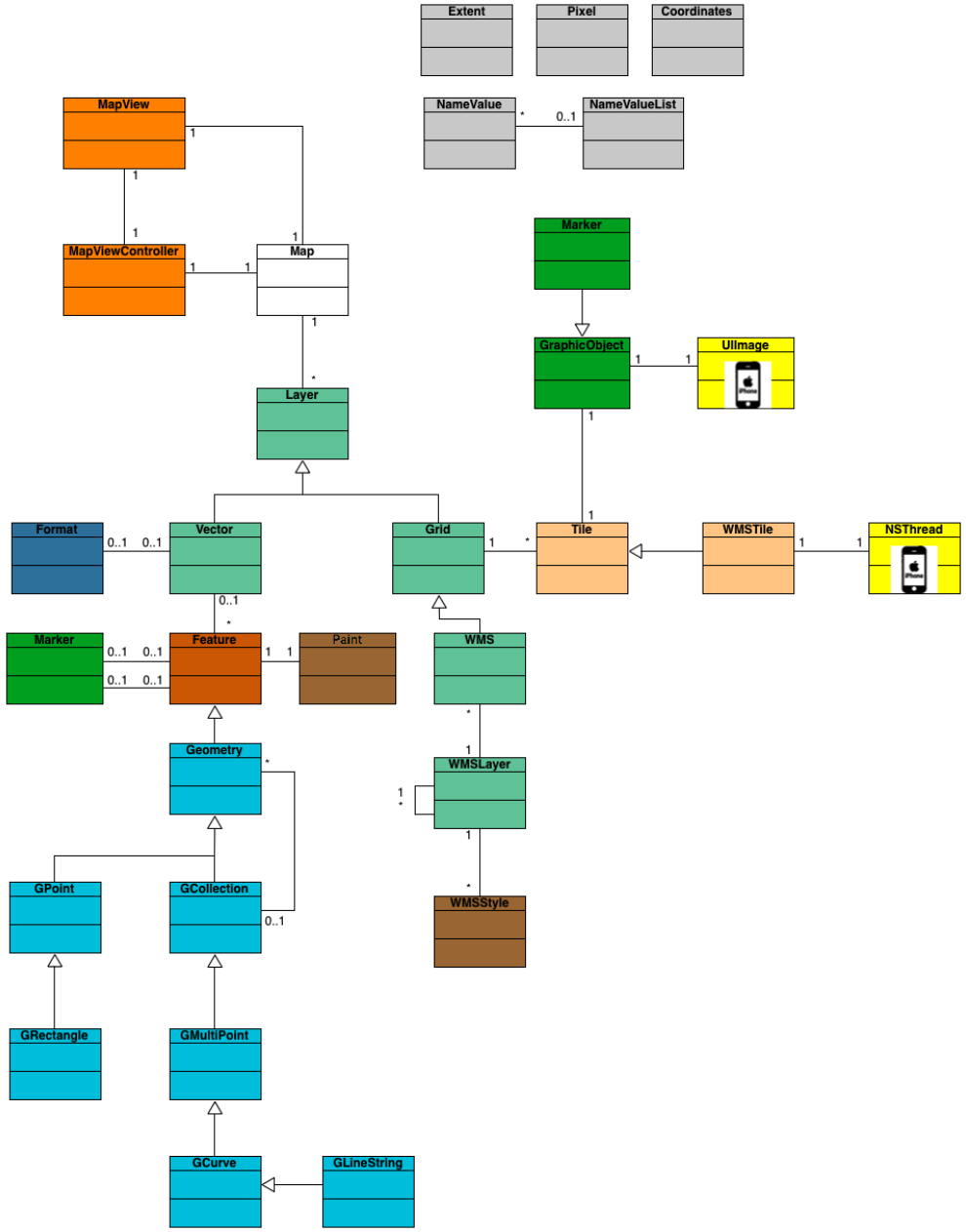

Figura B.2. Diagrama de clases de la segunda iteración

Siguiendo con la relación de los colores, se añaden los siguientes:

- Naranja oscuro (Feature): clases que definen los elementos básicos de las capas vectoriales
- Azul (Geometry): clases que definen el conjunto de geometrías soportadas
- Azul oscuro (Format): clases relacionadas con la importación de datos usando archivos formateados

Sus clases principales son: *Vector*, *Feature*, *Marker* y *Geometrías*.

- *Vector*: representa una *Layer* formada por un conjunto de *features* [14].
- *Feature*: es la unidad básica de la capa vectorial, contiene las propiedades básicas de cada elemento.
- *Marker*: si la *feature* lleva asociada una imagen, esta se almacena en la clase *marker*, que es una subclase de GraphicObject.
- *Geometrías*: conjunto de clases que definen estructuras geométricas que representan la información contenida en los features e incluyen métodos para su visualización.

En la siguiente fase únicamente se añadieron 3 clases al diagrama, *Control*, *ZoomAndPan* y *ConnectionDelegate* para poder mover el mapa y realizar zooms. *ConnectionDelegate* fue la clase que sustituyó a NSThread para poder realizar las peticiones de un modo asíncrono, como ya se explicó en el apartado desarrollo.

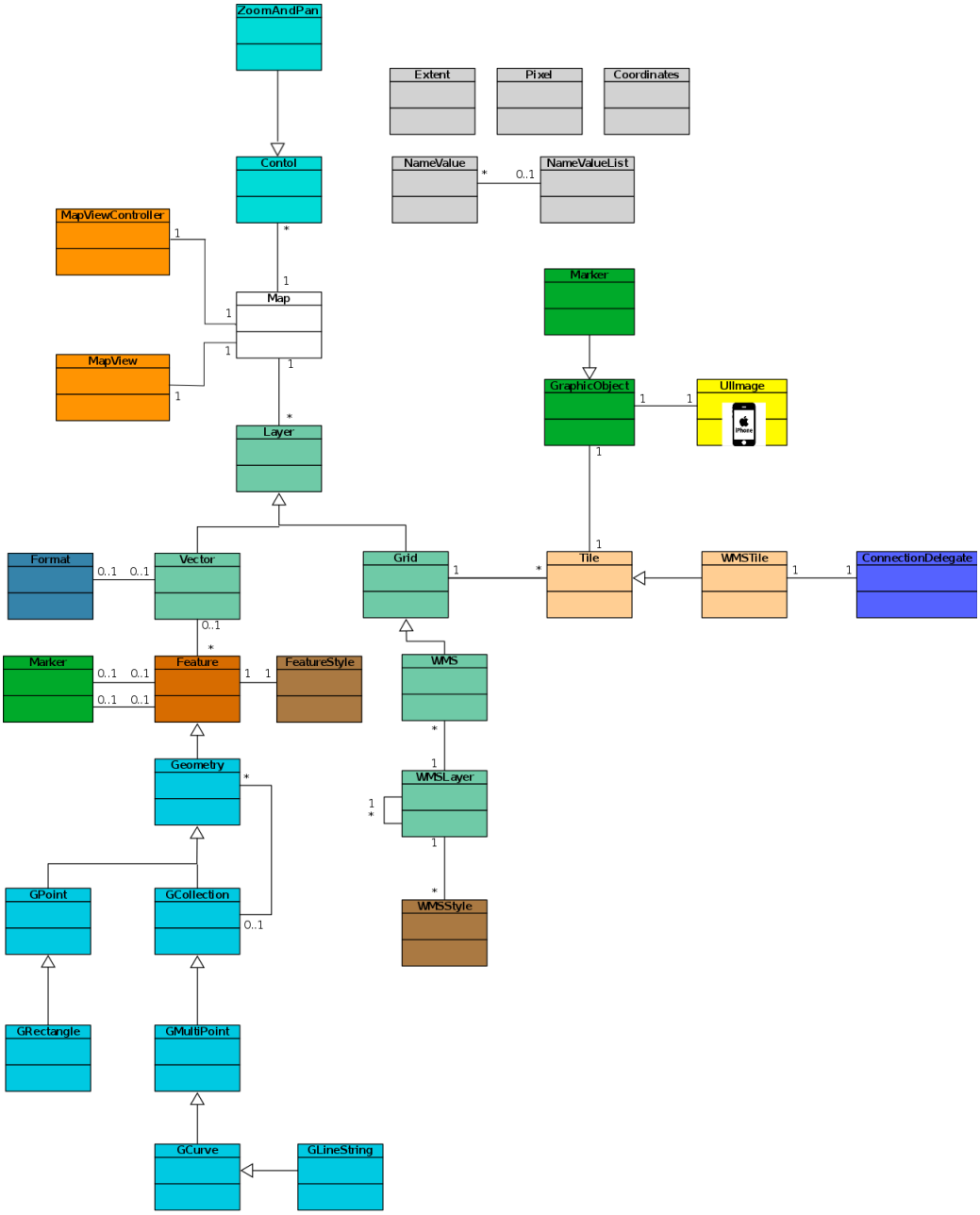

Figura B.3. Diagrama de clases de la tercera iteración

Colores asociados a las nuevas clases:

- Azul claro (Control): clases que definen los controles del mapa
- Morado (ConnectionDelegate): clases que realizan conexiones con servicios externos y se encargan de gestionar la descarga de datos

Sus clases principales son: *Control*, *ZoomAndPan* y *ConnectionDelegate*.

- *Control*: clase que representa un control genérico y gestiona su activación y desactivación, además de otros comportamientos comunes.
- *ZoomAndPan*: control de movimiento básico del mapa que permite hacer zoom in/out y *panning* (desplazamientos del mapa). Este control se agrega por defecto al mapa en su creación.

- *ConnectionDelegate*: clase encargada de realizar las conexiones con los servicios de datos de un modo asíncrono y de recibir y almacenar la respuesta.

La siguiente fase se centró en posibilitar la utilización de servicios WMS-C, por lo general más rápidos a la hora de responder a las peticiones que los WMS-OGC. A partir de esta fase, la clase *Grid,* que prácticamente no se utilizaba al estar los mapas compuestos por una única *tile* [4], empezó a ganar importancia, ya que de ella dependía el correcto tratamiento de las *tiles* que resultaban necesarias para poder implementar peticiones de servicios WMS-C.

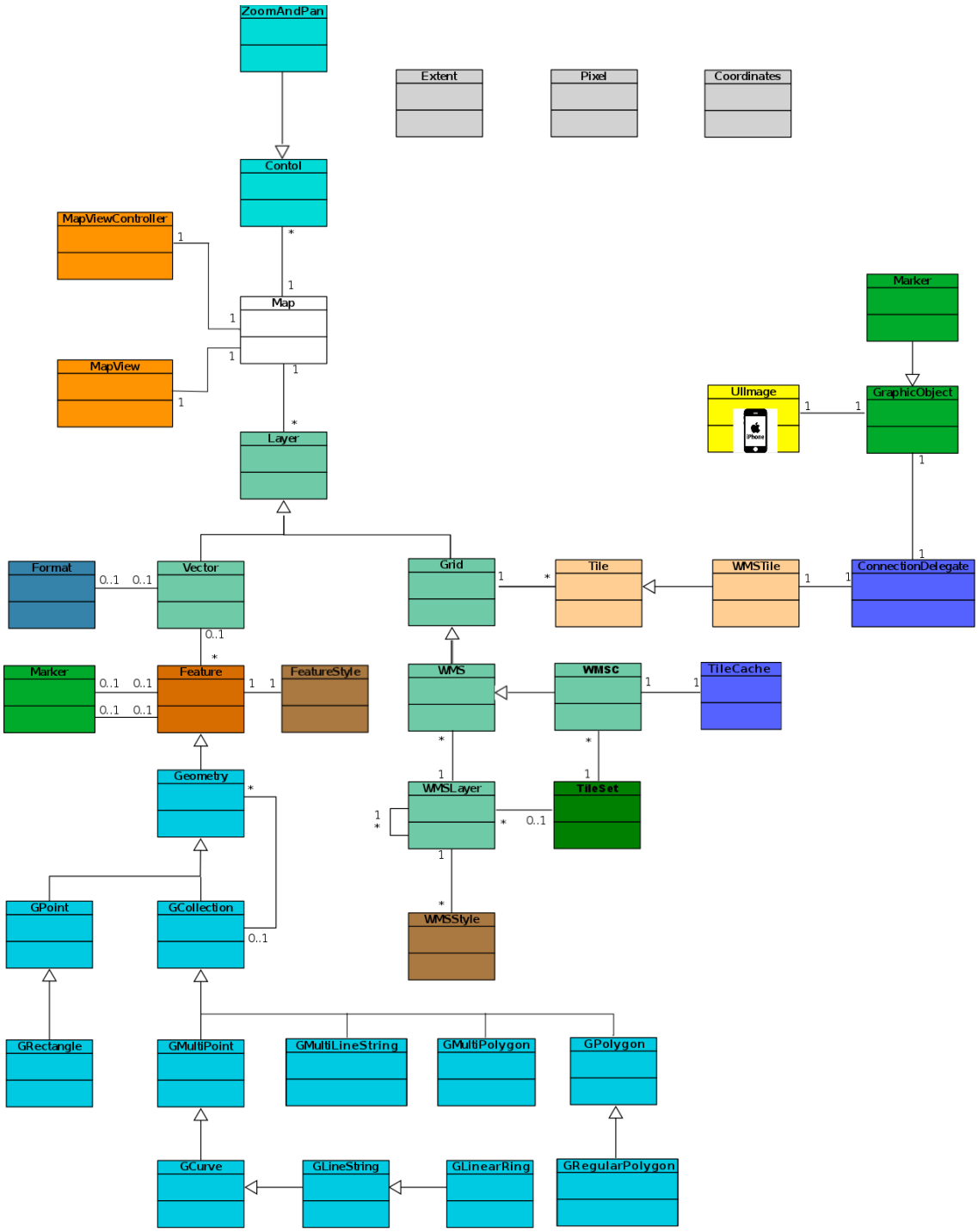

Figura B.4. Diagrama de clases de la cuarta iteración

Colores asociados a las nuevas clases:

- Verde oscuro (TileSet): clases que definen los parámetros de los servicios WMS-C

Sus clases principales son: *WMS-C*, *TileSet* y *TileCache*.

- *WMS-C*: es la clase que fija la mayoría de los parámetros de la *WMS* en función del *TileSet* que tenga activado en ese momento. Podría tratarse como un perfil de *WMS*.
- *TileSet*: especifica un conjunto de capas y estilo para representar los datos del *WMS-C*. Es una de las herramientas que se usan para fijar ciertos parámetros de las peticiones.
- *TileCache*: cache local que es capaz de almacenar *tiles* [4] para evitar peticiones recursivas de la misma zona del mapa y disminuir el trafico generado al navegar por el mapa.

Por último, se añadieron diversas funcionalidades a la librería, los controles *FeatureSelector* y *GPSLocation*, *Facade*, el formato *GeoJSON* y se desdobló la clase *ConnectionDelegate* en *TileConnectionDelegate* y *MarkerConnectionDelegate* para mejorar la descarga de las imágenes de los *markers* que se encontraban en una URL en vez de en local.

- *FeatureSelector*: dota a las *features* [14] del mapa de la capacidad de poder ser activadas al ser tocadas en la pantalla táctil o navegando por ellas mediante algún componente de la interfaz.
- *GPSLocation*: permite representar la posición actual del usuario en el mapa.
- Facade: centraliza la comunicación con aplicaciones externas para mejorar su reusabilidad.
- GeoJSON: clase que *parsea* los datos de un fichero escrito con formato GeoJSON y crea las geometrías necesarias para representar esos datos sobre el mapa en una capa vectorial.
- *MarkerConnectionDelegate:* encargado de descargar imágenes de los *markers* que se encuentren en una dirección URL utilizando un método similar al que se usa con las *tiles* [4].

En la siguiente imagen puede observarse el estado final del diagrama de clases del Framework [12] en el que se han remarcado las clases pertenecientes a cada iteración a modo de resumen.

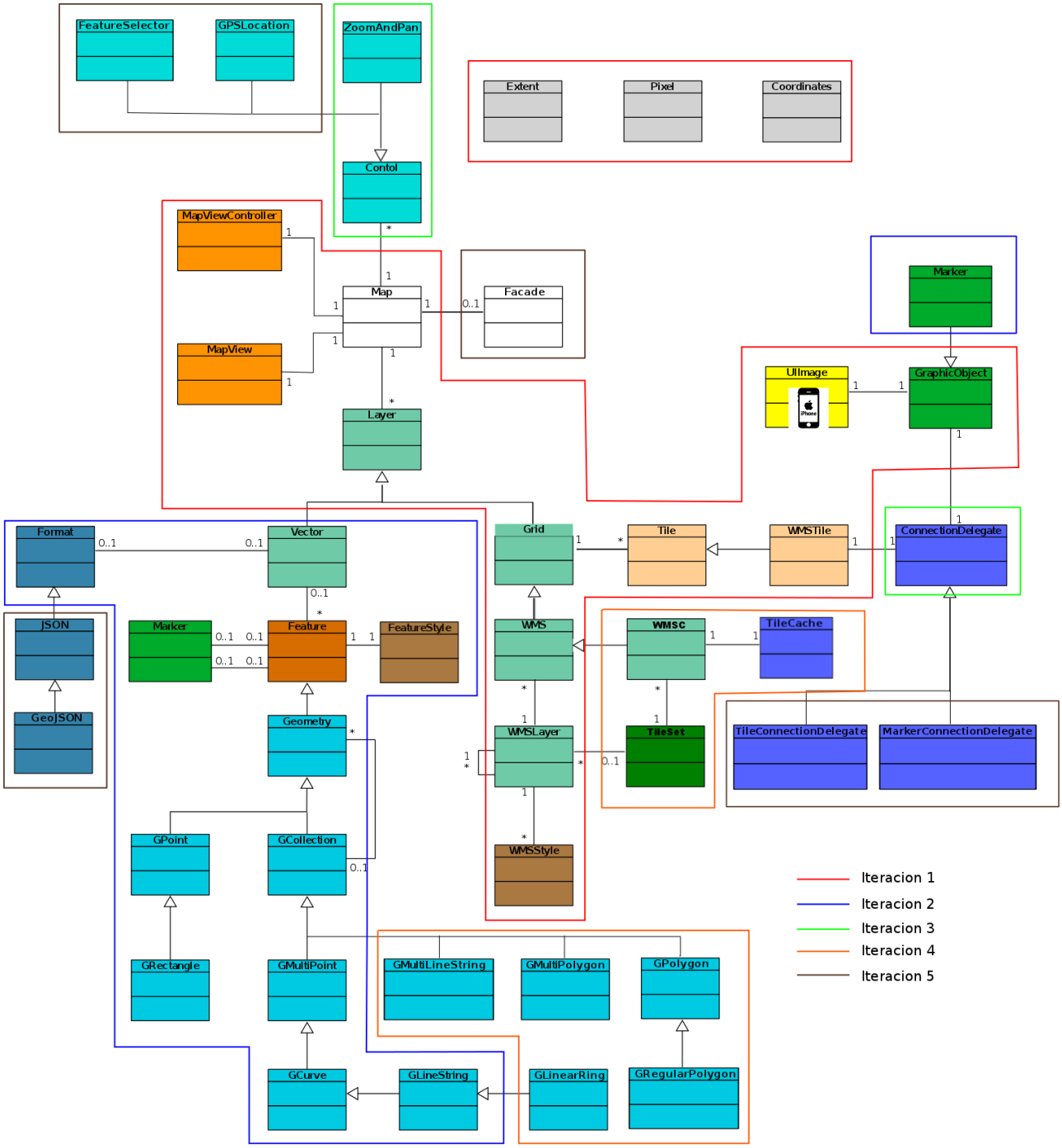

Figura B.5. Diagrama de clases final

Resumen de la relación de colores;

- Blanco (Map y Facade): proporcionan los métodos de creación de las clases principales del Framework [12]
- Naranja (MapView): clases relacionadas con la visualización de los datos
- Verde claro (Layer): clases relacionadas con las capas
- Naranja claro (Tile): clases que definen los elementos básicos de las capas
- Verde (GraphicObject): clases que almacenan la información gráfica de otros elementos
- Amarillo (UIImagen): clases propias del lenguaje Objective-C
- Marrón (WMSStyle): clases que definen los estilos de representación de otras clases

- Verde oscuro (TileSet): clases que definen los parámetros de los servicios WMS-C
- Naranja oscuro (Feature): clases que definen los elementos básicos de las capas vectoriales
- Azul (Geometry): clases que definen el conjunto de geometrías soportadas
- Azul claro (Control): clases que definen los controles del mapa
- Azul oscuro (Format): clases relacionadas con la importación de datos usando archivos formateados
- Morado (ConnectionDelegate): clases que realizan conexiones con servicios externos y se encargan de gestionar la descarga de datos
- Gris (Extent): son clases que definen tipos de datos básicos, extensión, pixel y coordenada, son usadas por muchas de las clases y no se ha dibujado su relación con ellas para no estropear la claridad de los diagramas.

B.1.2. Modelo dinámico

Una vez definidas las distintas clases que implementarán las nuevas funcionalidades del sistema, se describe, con la ayuda de los diagramas de secuencia, cómo estas clases interactúan entre sí para llevar a cabo algunas de las funciones principales de esta librería.

Los diagramas de secuencia más relevantes son los que representan la adición de una capa, WMS-OGC o WMS-C, al mapa, ya que en estas acciones intervienen muchas de las clases de la librería y son necesarias para que el mapa muestre algún tipo de información. Al añadir la capa al mapa, se desencadenan las sentencias necesarias para que se realice la conexión con el servicio correspondiente y una vez se reciben los datos, el mapa repinta su información enviando una sentencia *draw* a todos sus componentes. Este repintado se realiza siempre que la información del mapa cambia y se ha detallado en estos primeros diagramas, pero para los siguientes se dará por supuesto que después de la sentencia *redraw* del mapa, se produce el repintado de todos sus componentes.

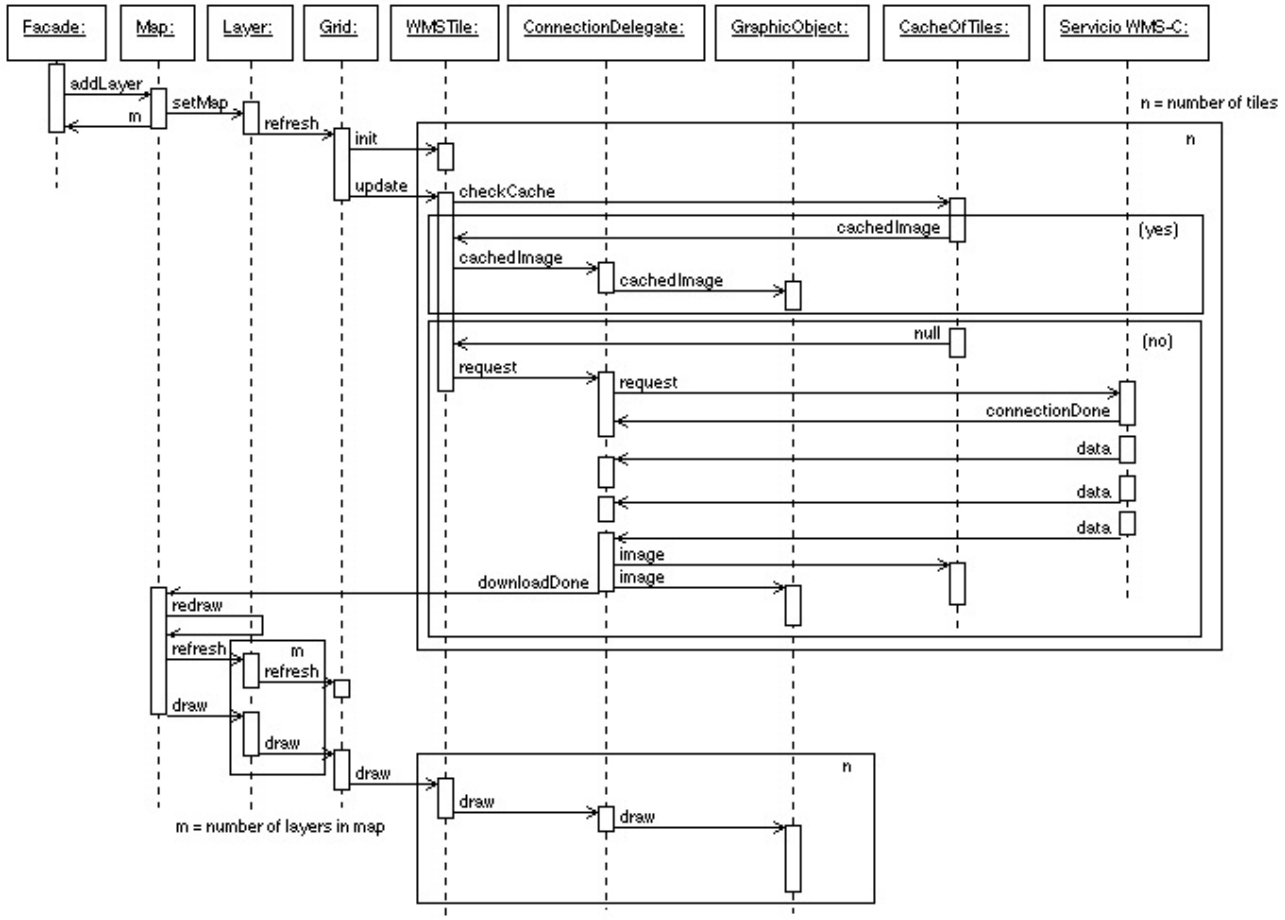

Figura B.6. Diagrama de secuencia: Añadir capa WMS-C al mapa

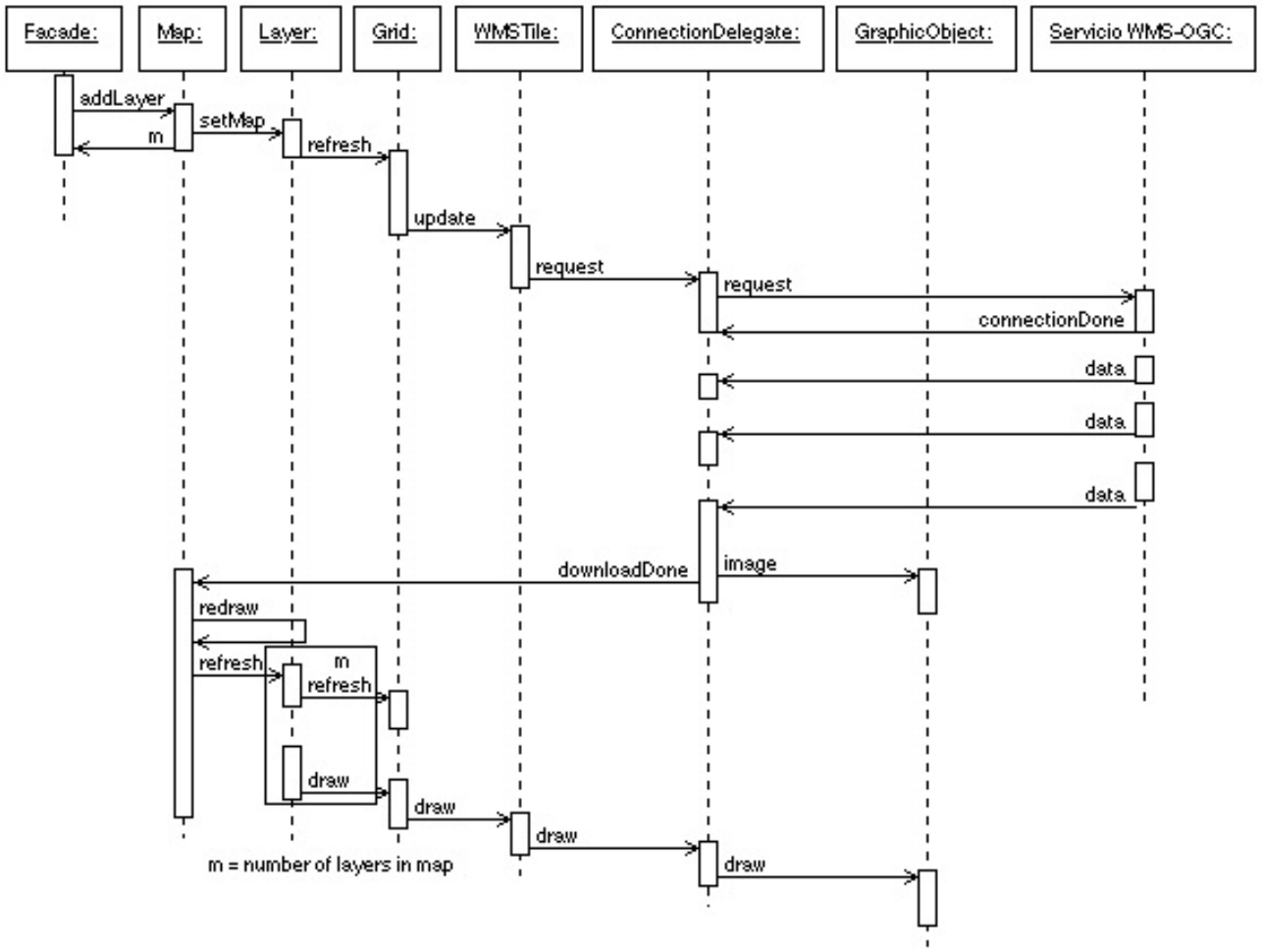

Figura B.7. Diagrama de secuencia: Añadir capa WMS-OGC al mapa

En cuanto a las capas vectoriales, al no obtener datos de un servidor, siguen un comportamiento ligeramente diferente. El modo más común de crear estas capas es usando la funcionalidad de la librería que permite añadir datos vectoriales usando un fichero GeoJSON, aunque podrían ser añadidas por otros medios, como por ejemplo crear la capa vectorial y las *features* [14] una a una, pero se considera un caso particular del primer modo y por ello no se detallará en otro diagrama.

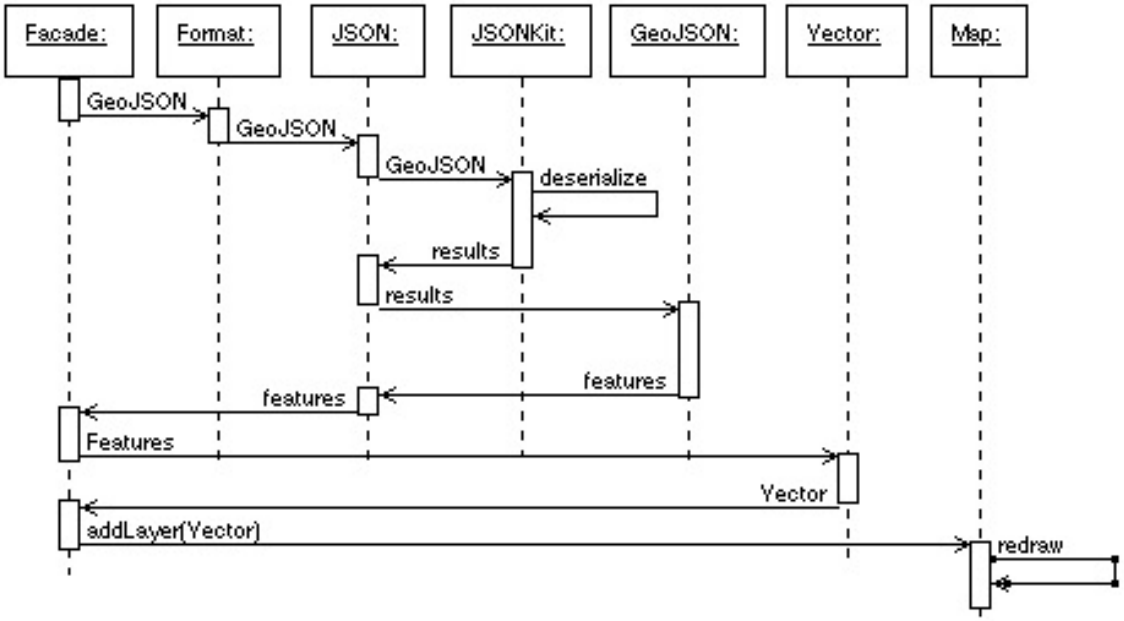

Figura B.8. Diagrama de secuencia: Reconocimiento de archivo GeoJSON

El siguiente diagrama muestra la eliminación de una capa, la sentencia *release* equivale a un *dealloc*, es decir, a una liberación de memoria. En realidad esta sentencia *release* se propagaría por todos los objetos que posee layer, sean grid, *tiles* [4], graphicObjects, etc. pero no aporta ninguna información útil al diagrama, por lo que se ha simplificado.

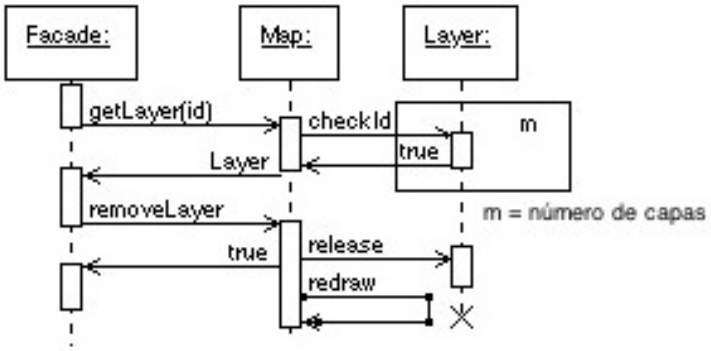

Figura B.9. Diagrama de secuencia: Eliminación de una capa del mapa

Por último, se detallan los diagramas de los tres controles que proporciona la librería, FeatureSelector, ZoomAndPan y GPSLocator. Todos ellos son activados por el usuario a través de la interfaz o de la pantalla táctil del iPhone, siendo los únicos elementos de la librería que pueden comunicarse directamente con el usuario sin necesidad de pasar por le Facade [13]. Esto se produce porque el usuario puede interaccionar directamente con una de las clases de la librería, el MapView, provocando cambios directos en otras clases, pero sin producir interacción con la aplicación que incluye el visor.

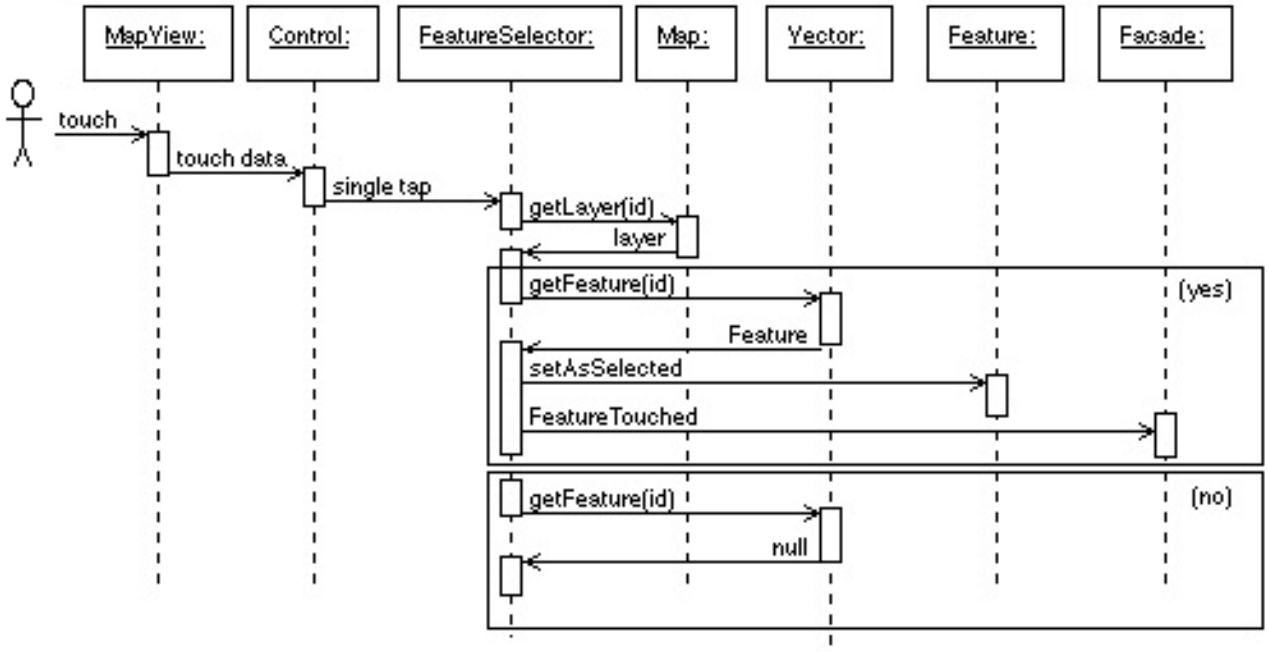

Figura B.10. Diagrama de secuencia: Selección de una feature

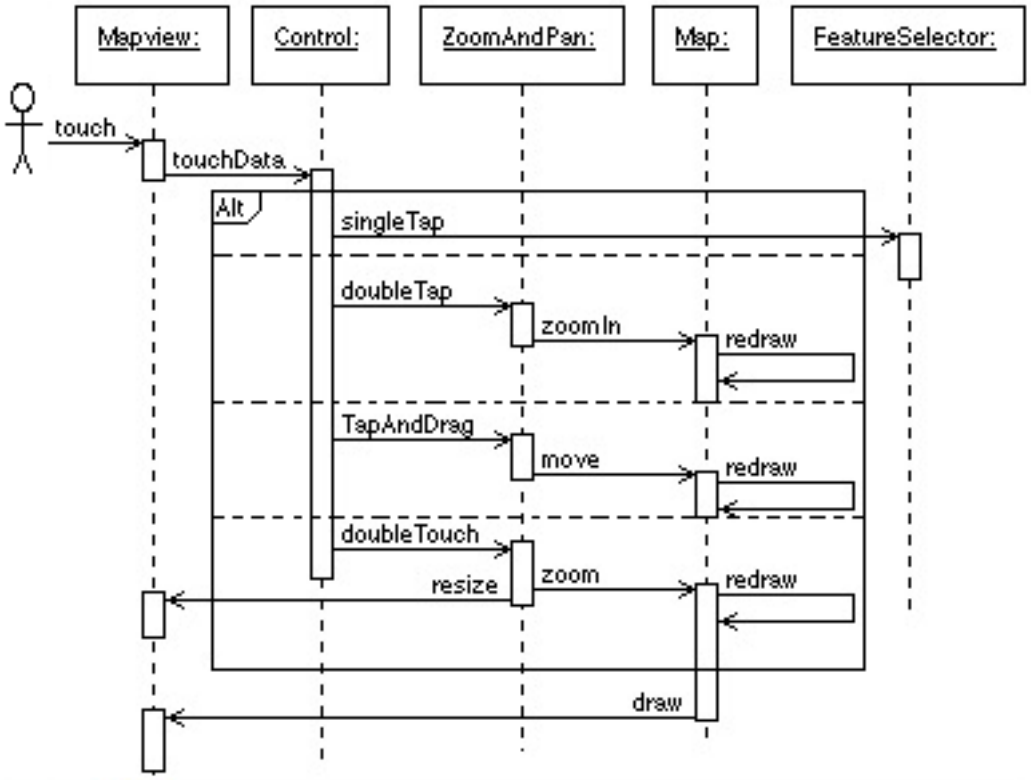

Figura B.11. Diagrama de secuencia: Tratamiento de los touchs o toques del usuario

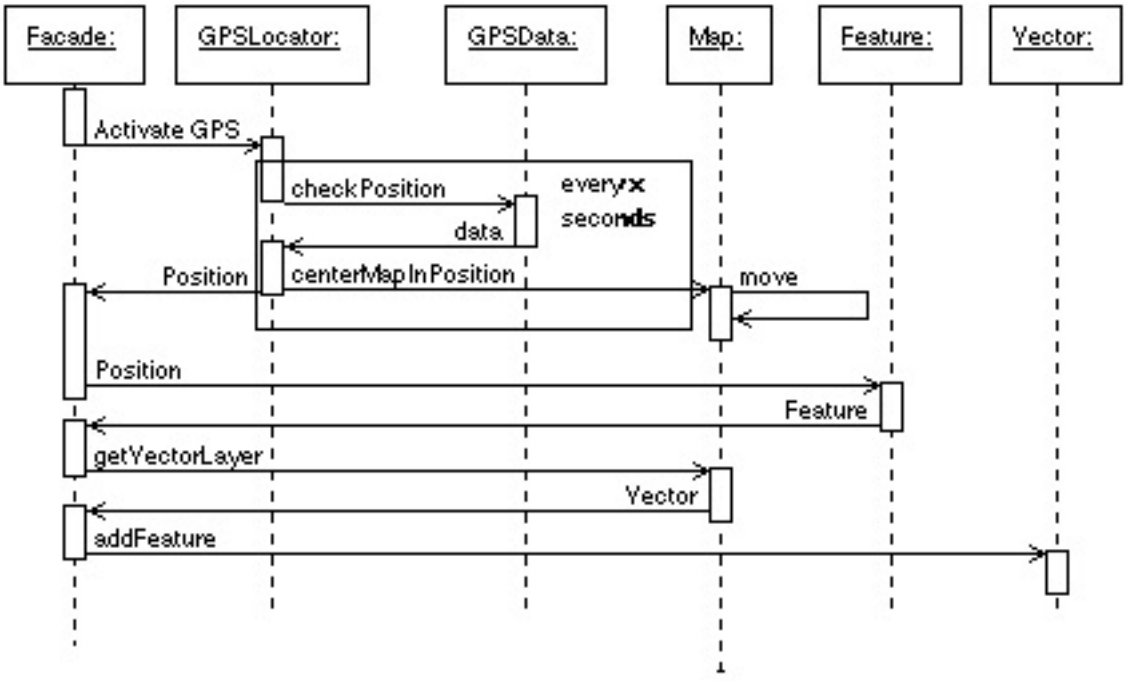

Figura B.12. Diagrama de secuencia: Geoposicionamiento del usuario

B.1.3. Configuración

El fichero de cabecera gsl_mobileMap_iPhone.h, define un conjunto de parámetros que representan los valores por defecto que utilizará el Framework [12] para crear sus componentes. Muchos de estos parámetros definen constantes utilizadas por defecto en la librería que no deberían modificarse, pero, la siguiente lista de parámetros pueden ser re-definidos para modificar cierto comportamiento del Framework:

- defaultOffScreen: Representa el número de pixels extra en cada dirección que se pedirá a los servidores
- defaultTimeOut: Tiempo de espera hasta considerar que la petición ha resultado fallida
- MarkerPOISelectedHeight: Altura de la imagen que crea el efecto de POI [10] seleccionado
- MarkerPOISelectedWidth: Anchura de la imagen que crea el efecto de POI seleccionado
- DEFAULTZOOMIN: Incremento de zoom que se realizara con un doble *tap,* siendo un *tap* un toque rápido sobre la pantalla.
- RESIZEFACTOR: modifica el efecto del *resize* que se realiza a la imagen del mapa cuando se realiza zoom con *touch doble*
- ZOOMTOLERANCE: número de pixels mínimo que deberá recorrer un *touch double* para realizar zoom
- FIRSTZOOMAT: se realizará un nivel de zoom mientras el tamaño de movimiento haya sido entre ZOOMTOLERANCE y FIRSTZOOMAT pixels
- LASTZOOMAT: se realizarán dos niveles de zoom mientras el tamaño movimiento haya sido entre FIRSTZOOMAT y LASTZOOMAT, y tres niveles de zoom si se supera el tamaño de LASTZOOMAT
- defaultColor: color por defecto con el que se pintarán las *features* [14]
- defaultStroke: ancho por defecto de la linea que dibujará las geometrías de una *feature*
- defaultTransform: transformación aplicada por defecto al tamaño de los gráficos asociados a una *feature*

- defaultAlpha: porcentaje de transparencia aplicada por defecto a los gráficos asociados a una *feature*
- defaultName: nombre del estilo de representación por defecto de las *features* que incluye los parámetros anteriores
- defaultCapacity: tamaño por defecto de la cache de *tiles* [4]
- DEBUG: más que un parámetro es un *flag*, que puede ser activado para que se muestre información por consola que puede ayudar a *debugear* el comportamiento del Framework [12]

B.2. Aplicación iRoutes

El objetivo fundamental de esta aplicación es ofrecer al usuario un interfaz intuitivo mediante el cual pueda importar un fichero con formato GeoJSON donde se encuentre definida una ruta y un conjunto de POIs [10]. Este fichero puede encontrarse en memoria local o en una dirección URL.

Una vez elegida la ruta a visualizar, la aplicación mostrará una vista creada usando el Framework [12] gsl_mobileMap_iPhone que contendrá un mapa base de algún servicio de mapas y una capa vectorial, añadida a este mapa base, con la información contenida en el fichero de la ruta. Para extraer los datos del fichero, se utiliza la clase GeoJSON que creará los objetos necesario para representar los datos del fichero en una capa vectorial.

La interfaz de la aplicación debía permitir seleccionar las *features* [14] del tipo *marker* añadidas al mapa y poder geoposicionar al usuario. Para ello se añaden los controles *FeatureSelector* y *GPSLocation* al mapa y dos botones para activarlos/desactivarlos a voluntad del usuario.

B.2.1. Modelo estático

El diagrama de clases de la aplicación es bastante simple, principalmente porque delega en el Framework [12] la construcción y funcionamiento del visor de mapas, ocupándose únicamente de la gestión de los archivos de rutas y de dotar al usuario de la interfaz necesaria para la interacción con la aplicación.

Las clases que componen la aplicación son *AppDelegate*, *AppFacade*, *AppViewController*, *RoutesTableViewController*, *MyTableView* y *Route*.

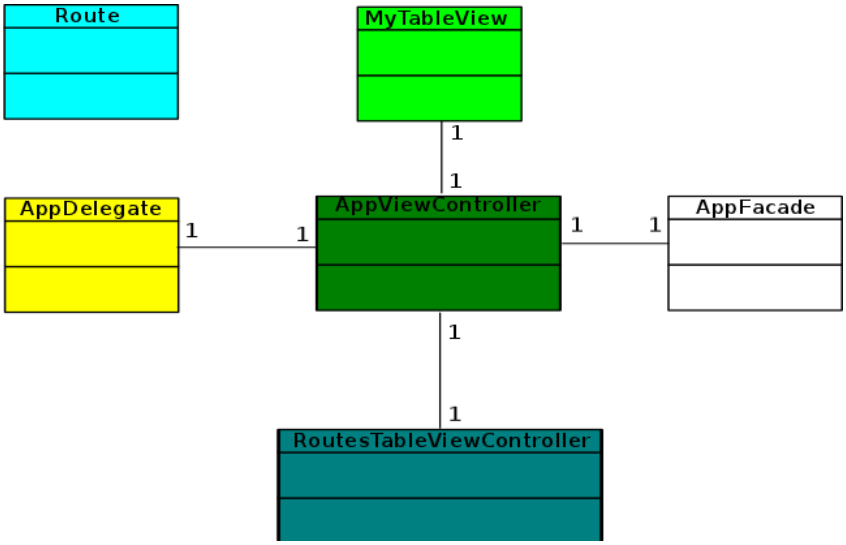

Figura B.13. Diagrama de clases de la aplicación iRoutes

- *AppDelegate*: encargado de lanzar la aplicación, mostrar la pantalla inicial y crear el *AppViewController*.
- *AppViewController*: controlador de las vistas de la aplicación, crea los componentes de la interfaz y se encarga de gestionar las interacciones del usuario con la interfaz.
- *AppFacade*: subclase de la clase Facade [13] del Framework [12] que añade métodos específicos para la comunicación entre los componentes de esta arquitectura.
- *MyTableView:* vista con forma de tabla que se usa principalmente para mostrar las rutas disponibles en la aplicación.
- *RoutesTableViewController*: dota de diversos controles a *MyTableView* y gestiona la interacción del usuario con la vista de la tabla.
- *Route:* almacena datos de las rutas como nombre, distancia, etc. para poder mostrarlos en diversas zonas de la interfaz.

B.2.2. Modelo dinámico

Tras detallar las clases que componen el sistema pasamos ahora a explicar cómo éstas interactúan entre ellas para conseguir sus objetivos. Nos centraremos en las acciones más relevantes que puede realizar el usuario, importar ruta desde URL, importar ruta desde disco, ver ruta en mapa, borrar ruta, activar información y volver al menú principal. Empezaremos detallando los modos posibles de importar un archivo de rutas.

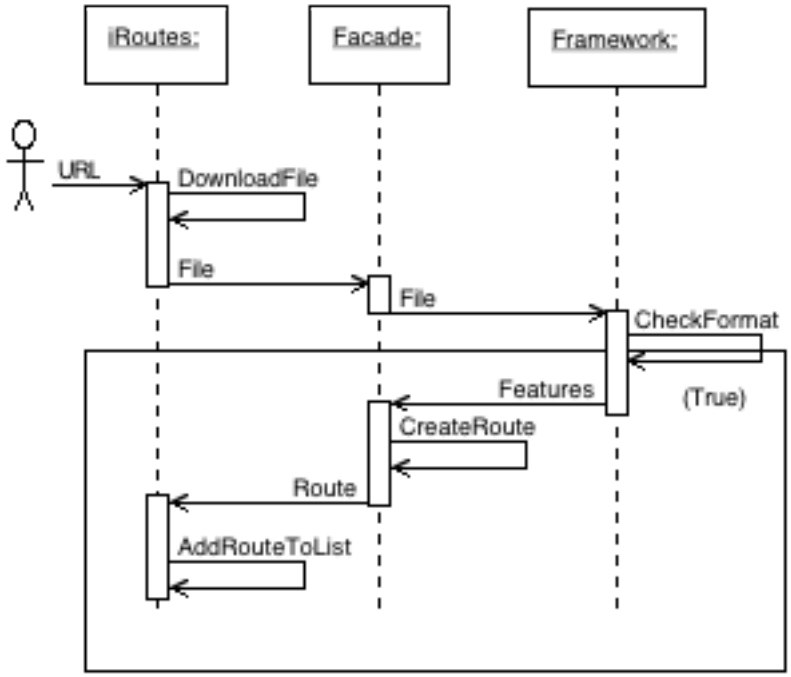

Figura B.14. Diagrama de secuencia: Importación de ruta desde URL

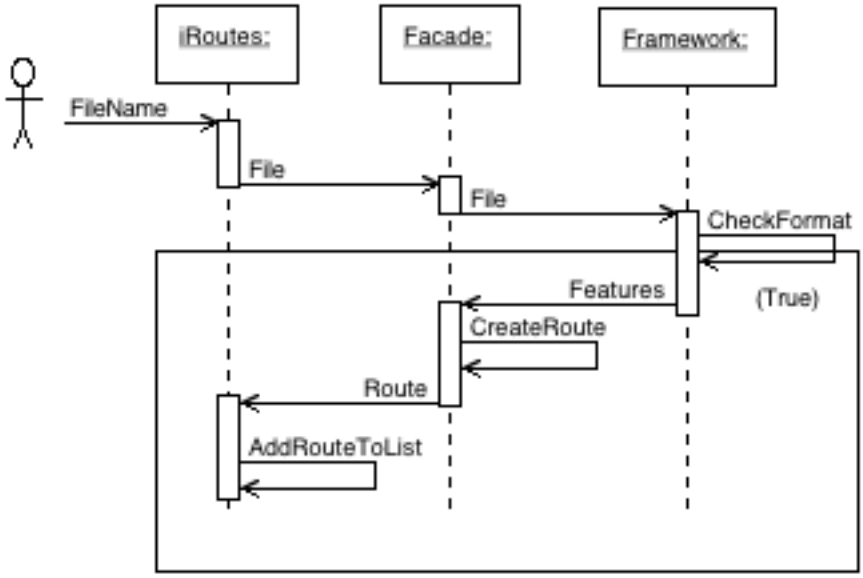

Figura B.15. Diagrama de secuencia: Importación de ruta desde disco

Como puede observarse, el Facade [13] siempre actúa de intermediario entre la aplicación y la librería. No se ha considerado relevante representar el caso en el que se importa un fichero con formato no soportado por la librería, ya que únicamente se mostrará al usuario un aviso de formato erróneo y no cambiará el estado de la aplicación. Una vez que la ruta está importada, el usuario puede borrarla o visualizarla del modo que representan los siguientes diagramas.

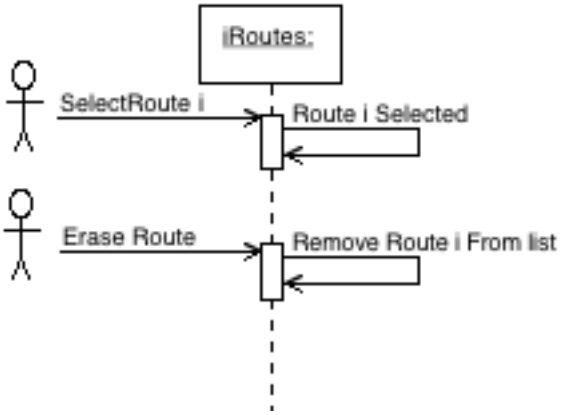

Figura B.16. Diagrama de secuencia: Borrado de una ruta

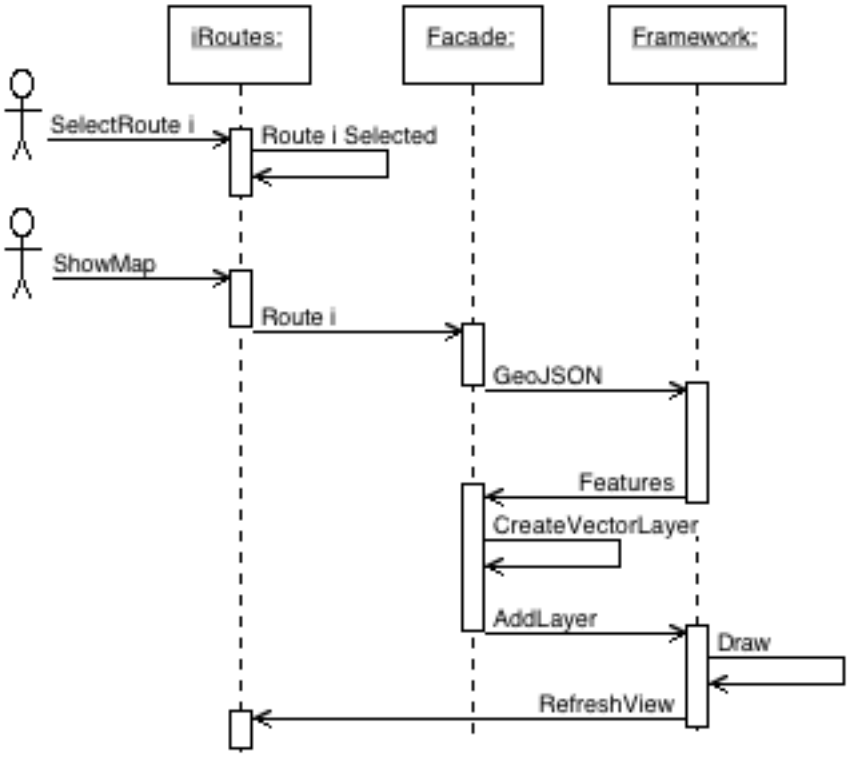

Figura B.17. Diagrama de secuencia: Visualización de una ruta

La acción de borrar una ruta únicamente afecta a la aplicación, resultando una operación muy sencilla. En cambio al visualizar una ruta, es el Framework [12] el encargado de hacer prácticamente todo el trabajo, se *parsea* de nuevo el archivo GeoJSON, pero esta vez para obtener las *features* [14] que formarán la nueva capa vectorial. Una vez esté creada, se añadirá al mapa, desencadenando una actualización de los datos del mapa, que ya se explicó en el modelado dinámico de la librería, para terminar pintando y mostrando la nueva información añadida.

Por último, se detallan las acciones relacionadas con la aplicación que puede realizar el usuario cuando está visualizando una ruta.

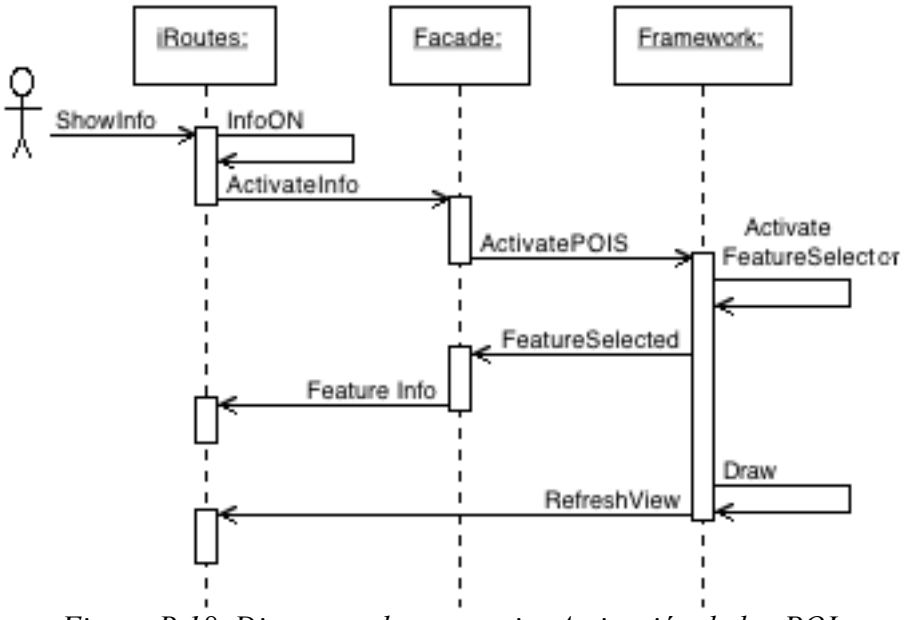

Figura B.18. Diagrama de secuencia: Activación de los POIs

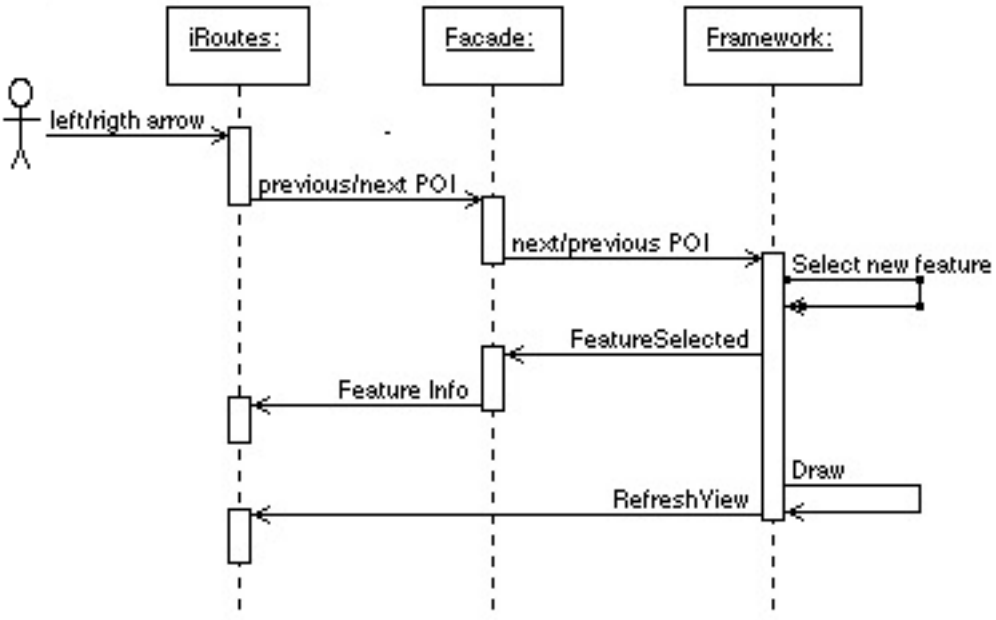

Figura B.19. Diagrama de secuencia: selección de nuevo POIs

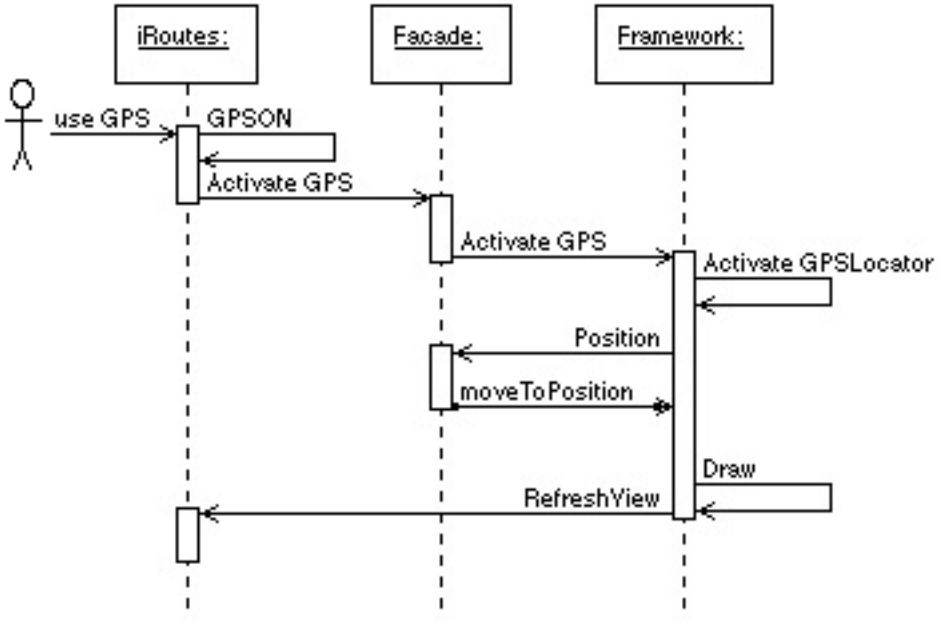

Figura B.20. Diagrama de secuencia: Activación del GPS

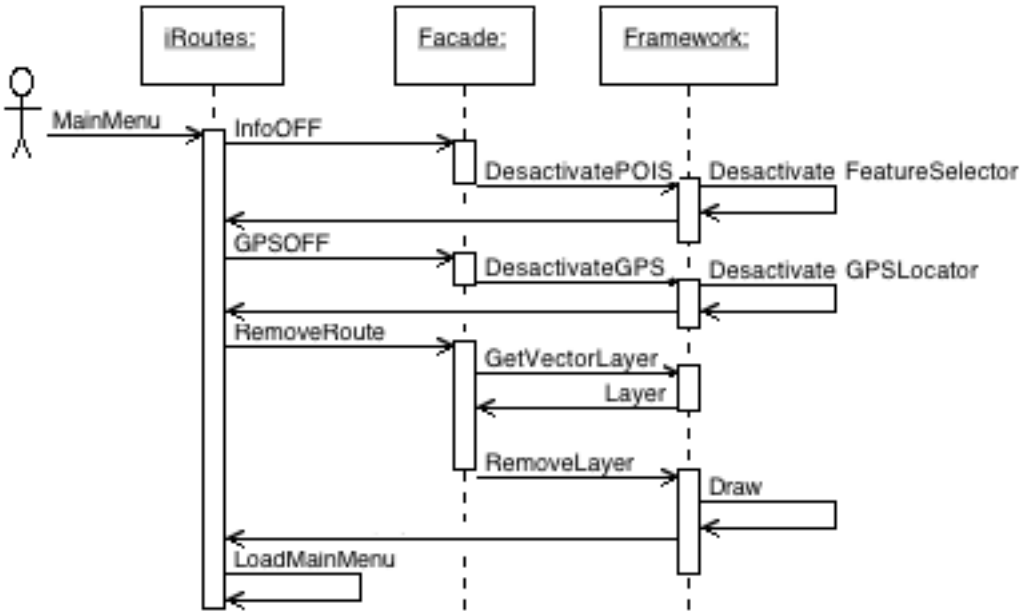

Figura B.21. Diagrama de secuencia: Retorno al menú principal

B.2.3. Interfaz

La interfaz se diseñó para ser muy simple e intuitiva posible. Aparentemente eran suficientes dos botones para realizar todas acciones necesarias, uno para importar una ruta y otro para poder borrarla, más una lista de las rutas importadas en la aplicación, quedando el menú principal con un aspecto similar al siguiente:

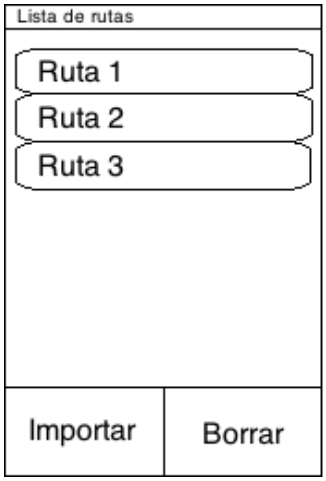

Figura B.22. Interfaz: Menú principal

Sin embargo, el mecanismo para importar un archivo no estaba claro al desconocer el modo de tratar los archivos dentro de la estructura de datos del iPhone. Se esperaba que el entorno de desarrollo ofreciera algún Framework [12] o similar para poder explorar la memoria local en busca del archivo correspondiente a la ruta que queremos importar, pero al parecer no está permitido navegar por la estructura interna de los archivos, únicamente puede navegarse por las carpetas de la aplicación. Teniendo esto en cuenta, se optó por mostrar los nombres de los archivos, existentes en esa carpeta, en la propia tabla donde se mostraban las rutas para que el usuario pudiera seleccionar uno de ellos. Además se incorporó la posibilidad de importar un archivo usando una URL, que descargaría el archivo a la memoria local y a la vez lo añadiría a la lista de rutas, siendo este el mapa de navegación resultante:

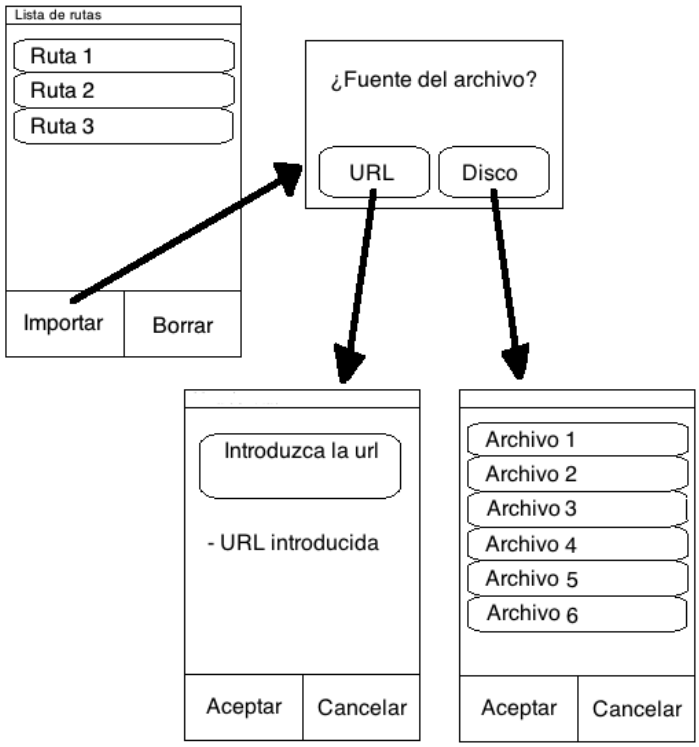

Figura B.23. Interfaz: Importación de ruta

De este modo, quedaba resuelta la importación de archivos, pero no resultaba intuitivo como cargar los datos de una ruta sobre el mapa. Inicialmente se pensó en que al hacer doble *touch* sobre una de ellas, esta se cargara sobre el mapa. Posteriormente, se decidió que si una ruta estaba seleccionada en la lista, el botón importar se cambiara por uno llamado "ver en mapa" que cargara la información de la ruta. Adicionalmente las rutas debían poder seleccionarse y deseleccionarse en la lista, ya que si no, sería imposible importar una nueva ruta una vez se hubiera seleccionado una ya existente.

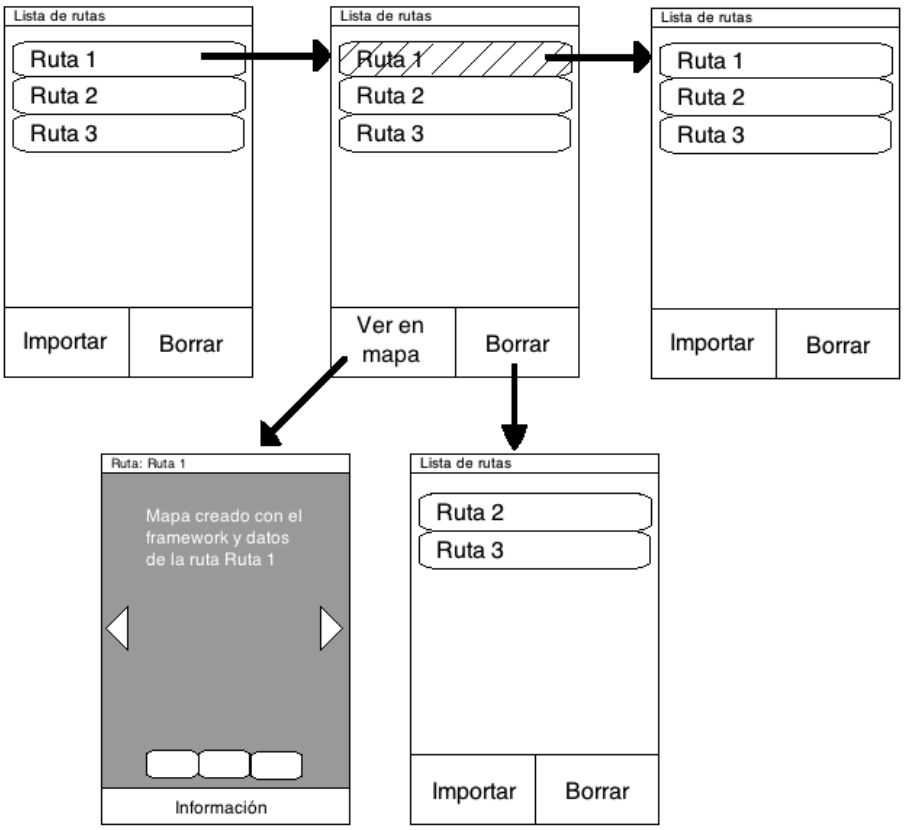

Figura B.24. Interfaz: Mapa de navegación del interfaz

B.2.4. Configuración

Al igual que con el Framework [12], la aplicación iRoutes [11] también tiene un fichero de cabecera donde se definen muchos parámetros que permiten cambiar cierto comportamiento o el aspecto de la interfaz. Estos parámetros no son tan importantes como los de la librería, la mayoría de ellos tienen relación con la interfaz, las imágenes que se usan en los botones, los colores del texto, los textos usados en la interfaz, parámetros de las animaciones, etc. Estos son los más relevantes:

- Dimensiones de los diferentes objetos de la interfaz:
  - appWidth: anchura de la aplicación
  - appHeight: altura de la aplicación
  - topBarHeight: altura de la barra superior
  - bottomBarHeight; altura de la barra inferior

  - buttonsHeight: altura de los botones
  - menuButtonsHeight: altura de los botones del menú
  - navButtonsWidth: anchura de los botones de navegación
  - navButtonsHeight: altura de los botones de navegación
  - menuButtonsDisp: desplazamiento de los botones del menú con respecto a la barra inferior
  - urlViewDisplace; desplazamiento del cuadro donde se introduce la URL

- Dimensiones del visor de mapas:
  - MapWidth: anchura del visor
  - MapHeight altura del visor
  - MapDispX: desplazamiento del visor en el eje X
  - MapDispY: desplazamiento del visor en el eje Y

- Parámetros de las animaciones de la interfaz
  - introDuration: duración de la animación de la portada
  - introDelay: retraso de la animación de la portada
  - loadDuration: duración de la animación que esconde el menú principal al cargar una ruta
  - loadDelay: retraso de la animación que esconde el menú principal al cargar una ruta
  - deactivateTime: tiempo durante el cual se desactivan los touchs de la interfaz, para que no se produzcan efectos extraños mientras se esta reproduciendo la animación

- Lista de servicios que se han probado en el desarrollo de este proyecto, puede elegirse uno o ningún servicio WMS-C y cualquier número de servicios WMS. También es posible añadir más servicios, consultar el manual del Framework [12] para instrucciones detalladas de cómo hacerlo.

  - WMSC_IDEE_BASE: WMS-C ofrecido por la Infraestructura de Datos Espaciales de España (IDEE).

  - WMSC_IDE_ZAR: WMS-C ofrecido por la Infraestructura de Datos Espaciales de Zaragoza (IDEZar).

  - WMSC_CARTOCIUDAD: WMS-C ofrecido por la Administración General del Estado.

  - WMS_IDEE_BASE: WMS-OGC ofrecido por la Infraestructura de Datos Espaciales de España (IDEE).

  - WMS_IDE_ZAR: WMS-OGC ofrecido por la Infraestructura de Datos Espaciales de Zaragoza (IDEZar).

  - WMS_CARTOCIUDAD: WMS-OGC ofrecido por la Administración General del Estado que muestra la cartografía en el ámbito urbano.

- ROUTES: nombre del archivo .plist donde se guarda la información de las rutas importadas

- DEBUG: *flag* que ayuda a la hora de *debugear* la aplicación cuando esta activado, mostrando información por la consola

Anexo C. Pruebas

En este anexo se detallarán todas las pruebas realizadas al sistema desarrollado. Su objetivo es tanto descubrir posibles fallos de implementación como asegurar que se cumplen todos los requisitos funcionales y no funcionales definidos previamente. Especialmente nos centraremos en probar el rendimiento de los distintos componentes del sistema.

C.1. Pruebas de funcionalidad

Para comprobar el correcto funcionamiento de los componentes desarrollados, se realizó una batería de pruebas que cubriera la mayoría de los escenarios posibles en la utilización de la aplicación iRoutes [11]. Al probar la aplicación de forma implícita se comprueba también el correcto funcionamiento del Framework [12]. Además, después de cada una de las iteraciones se fueron realizando pruebas unitarias que garantizaban el buen funcionamiento de las funcionalidades añadidas en cada iteración.

Las pruebas realizadas fueron las siguientes:

Descripción/propósito	Acciones de ejecución	Resultados esperados
Pruebas generales o comunes del Framework		
Acceso online a WMS basados en especificación WMS-OGC v1.0.0 o superior	- Añadir servidor WMS al visor, con las capas y estilos requeridos, definir el área visible inicial y lanzarlo	- Se visualizan las capas requeridas provenientes del WMS en el área geográfica indicada
Acceso online a WMS-C basados en especificación WMS-C de OSGeo	- Añadir servidor WMS-C al visor, con las capas y estilos requeridos, definir el área visible inicial y lanzarlo	- Se visualizan las capas requeridas provenientes del WMS-C en el área geográfica indicada
Comprobar funcionamiento herramientas básicas de navegación usando la pantalla táctil	- Moverse sobre el mapa con los gestos que se corresponden con: zoom in, zoom out y panning	- El mapa se mueve de forma esperada tras realizar cada uno de los gestos
Visualizar puntos georreferenciados (*markers*) sobre el mapa a partir de unas coordenadas y con un estilo determinado	- Inicializar el visor y moverse sobre el mapa con las herramientas de navegación - Probar a modificar el estilo del *marker* (e inicializar el visor de nuevo)	- Se visualizan los *markers* en su posición correcta y se mueven correctamente con las herramientas de navegación - El estilo del *marker* se representa correctamente
Visualizar geometrías georreferenciadas (puntos, líneas, rectángulos,...) sobre el mapa a partir de unas coordenadas y con un estilo	- Inicializar el visor y moverse sobre el mapa con las herramientas de navegación - Probar a modificar el estilo de la	- Se visualizan las geometrías en su posición correcta y se mueven correctamente con las herramientas de navegación

determinado	geometría (e inicializar el visor de nuevo)	- El estilo de las geometrías se representa correctamente
Visualizar rutas y POIs en visor de mapas a partir de formato vectorial GeoJSON	- Añadir ruta con POIs al mapa usando la apliación iRoutes	- La ruta y los POIs se visualizan correctamente en la posición esperada
Resaltar POIs al moverse con los botones laterales o hacer clic sobre él	- Activar botón de información, seleccionar POI haciendo click sobre el, usar los botones laterales para seleccionar otros POI	- Los POIs Se seleccionan correctamente de Los dos modos
Pruebas específicas para probar integración aplicación		
Comprobar integración Facade con otros componentes	- Cargar ruta desde la aplicación, si no existe ninguna, importar una - Seleccionar al menos un POI - Eliminar ruta	- La ruta se carga correctamente, los POIs se seleccionan correctamente y la ruta se elimina correctamente
Comprobar funcionamiento geoposicionamiento por GPS	- Activar GPS, movernos con el iPhone	- Comprobar que se pinta sobre el mapa un punto con nuestra posición geográfica. Si nos movemos, ese punto cambia de posición adecuándose a la nueva.
Permitir definir POIs no navegables pero sí seleccionables con clic	- Moverse por los POI con los botones laterales del mapa, y hacer clic en el marcador de los POIs seleccionables pero no navegables	- Comprobar que los POIs que hemos indicado en el fichero de propiedades como seleccionables pero no navegables, no se seleccionan al moverse con los botones, pero si al hacer clic sobre ellos.
Comprobar acceso offline a fichero de rutas y fichero de POIs	- Cargar ruta y fichero de POIs desde SD (offline) mediante método del Facade.	- Comprobar que tanto la ruta como los POIs se cargan correctamente
Comprobar acceso online a fichero de rutas y fichero de POIs	- Cargar ruta y fichero de POIs desde una URL mediante el método del Facade.	- Comprobar que tanto la ruta como los POIs se cargan correctamente
Integración visor de mapas en aplicación principal	- Cargar aplicación principal y entrar en una ruta concreta	- Comprobar que al entrar en la ruta, se muestra ésta en el mapa
Mostrar información del POI en la parte inferior de la ventana al resaltarlo	- Entrar en una ruta y hacer clic en uno de los POIs para resaltarlo	- Comprobar que se muestra una breve descripción del POI resaltado
Comprobar que se guarda el estado de la aplicación al salir	- Cargar una nueva ruta en la aplicación - Salir de la aplicación - Lanzar de nuevo la aplicación	- Comprobar que se listan las rutas que teníamos antes de cerrar.
Pruebas de robustez del Framework		
Comprobar comportamiento cuando se pierde la conexión con el servidor	- Se produce una perdida de la conexión	- El visor debe responder con algún mensaje de error y no aborta la ejecución - El visor vuelve a funcionar correctamente cuando se recupera la conexión
Comprobar comportamiento cuando el GPS no consigue localizar al usuario	- Se produce un error en el calculo de la posición del usuario	- El Framework debe responder con algún mensaje de error y no aborta la ejecución

Pruebas de robustez de la aplicación frente a entradas erróneas o incorrectas		
Comprobar comportamiento cuando se intenta visualizar una ruta cuyo fichero ya no existe	- Importar ruta desde disco o - Eliminar el archivo del disco local - Visualizar ruta cuyo archivo ha sido eliminado	- La aplicación debe responder con algún mensaje de error y no aborta la ejecución
Comprobar comportamiento cuando se intenta importar una ruta sin formato GeoJSON, como puede ser una imagen, o con formato GeoJSON erróneo	- Importar una imagen - Importar un archivo GeoJSON erróneo	- La aplicación debe responder con algún mensaje de error y no aborta la ejecución

Gracias a estas pruebas se encontraron y solucionaron ciertos errores de implementación. La versión final de los componentes supero con éxito cada una de las pruebas.

C.2. Pruebas de eficiencia

Una vez finalizado el desarrollo y pasadas las pruebas de funcionalidad, se realizaron mediciones de tiempos y carga en memoria para varios casos de uso de los componentes desarrollados, que se detallan a continuación. Para realizarlas se utilizo la aplicación Instruments que viene incluida en el entrono Xcode.

- Tiempo de carga de la aplicación

  La carga completa del simulador y la ejecución de la aplicación se realiza en apenas 5 seg.

- Tiempo de carga fichero de rutas y POIs [10]
  - Importación desde disco
    - Fichero con dos POIs: 0.005 ms.
    - Fichero con 48 POIs: 0.009 ms.
  - Importación desde URL
    - Fichero con dos POIs: 0.007 ms.
    - Fichero con 48 POIs: 0.012 ms.

- Tiempo de acceso a WMS-OGC y WMS-C

Para realizar estas mediciones se han utilizado tres rutas distintas que cargan zonas diferentes del mapa, dos de ellas pertenecen al área de Zaragoza y la otra al área de Madrid. Las peticiones se han hecho a todos los servicios descritos en el apartado arquitectura general (pág. 13 de esta memoria). Las peticiones del área de Madrid no se han podido probar en los servicios de IDEZar porque no posee datos de esta zona.

Estas mediciones están condicionadas por la respuesta de los sitios web, velocidad de conexión, etc. que impiden medir con exactitud la velocidad de respuesta de los servicios. En vez de realizar múltiples peticiones para luego calcular medias y obtener datos más exactos, se ha optado por realizar solamente dos peticiones de las mismas áreas, ya que es lo único necesitamos para comprobar la principal diferencia entre un servicio WMS-OGC y uno WMS-C.

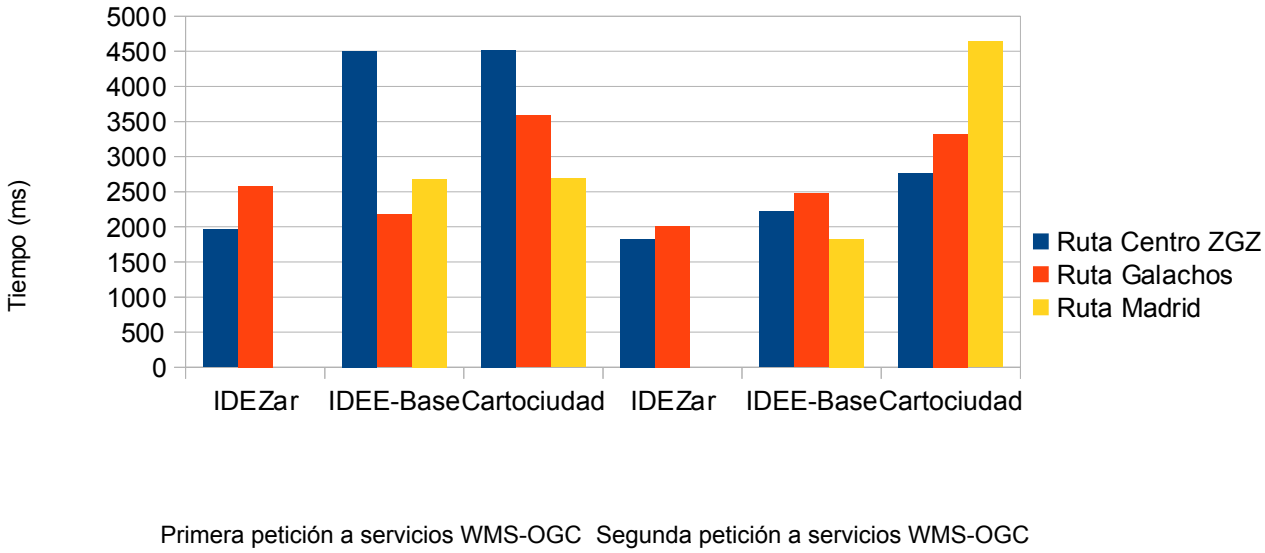

Figura C.1. Tiempos de respuesta de servicios WMS-OGC

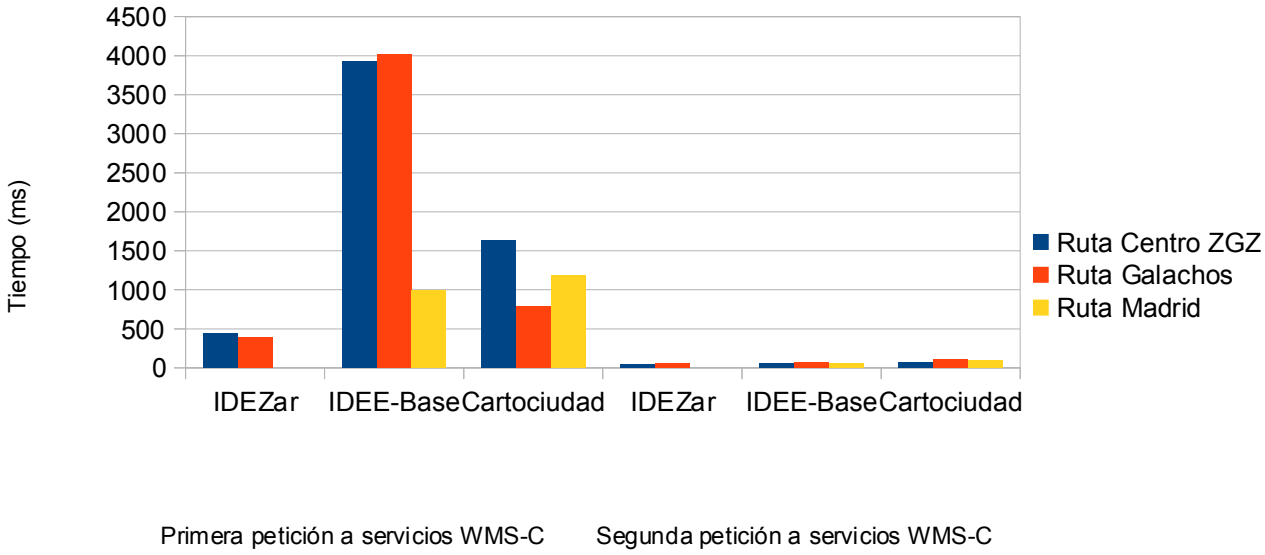

Figura C.2. Tiempos de respuesta de servicios WMS-C

Como puede observarse en las gráficas, por lo general, los servicios WMS-C tardan menos en responder a las peticiones, aunque dependiendo del área pedida y del servicio utilizado, pueden dar peor resultado que los WMS-OGC, como ocurre con el servicio IDEE-Base al pedir el área de la segunda ruta. Sin embargo, cuando se realiza una segunda petición de un área que ya había sido visitada, la mejora obtenida es drástica.

- Cantidad de memoria utilizada

La aplicación iRoutes solo ocupa alrededor de 1.36 MB en el momento de arrancar, debido a que el visor todavía no se inicializa y no existe ningún mapa cargado en la aplicación. Después de varios minutos de un uso normal con mapas del servicio WMS-C de IDEZar, el uso de memoria no alcanza los 3MB. Este incremento de memoria usada está relacionado con la cache de *tiles*, que va aumentando de tamaño conforme nos movemos por el mapa.

Utilizando los otros servicios, la memoria usada es diferente debido a que el peso de las *tiles* es diferente. Concretamente, al usar el WMS-C de IDEE-Base, el uso de memoria se llega a incrementar hasta los 11MB, manteniéndose casi siempre entre 5 y 8, y el de Cartociudad no supera los 7MB de uso de memoria. El Framework [12] está configurado para que la cache almacene un máximo de 50 *tiles*, este número se puede configurar como se explica en el Anexo B, pero habrá que tener en cuenta que el tamaño de las *tiles* es muy variable, normalmente entre 150 y 5 KB.

En el caso de los servicios WMS-OGC la memoria usada es todavía menor manteniéndose alrededor de los 2MB. Esto se debe a que los datos no son cacheados.

Además de la escasa memoria utilizada por la aplicación, produce muy pocos *leaks* o goteras, es el termino utilizado en el desarrollo con Objective-C para denominar a la memoria que ha sido reservada pero no liberada, ya que este lenguaje no posee recolector de basura y debe ser el propio programador el que se encargue de liberarla. Para confirmarlo, se realizó un uso intenso de la aplicación, utilizando todas sus funcionalidades. El resultado fue que la aplicación seguía ocupando únicamente 2.96 MB y el número total de *leaks* era de 160 bytes. Estos *leaks* se provocan por una referencia cíclica entre varias clases del Framework [12] que impiden la liberación de esta memoria. Este problema se detectó en las pruebas finales y se decidió no solucionarlo por ser un problema menor que no afecta al buen funcionamiento de la aplicación y porque conllevaba una modificación en el diseño de la estructura de datos.

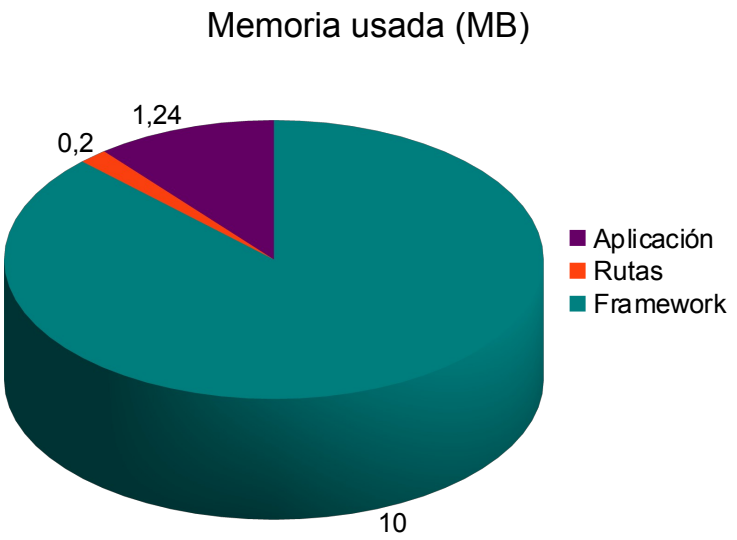

Figura C.3. Memoria utilizada por los componentes desarrollados

La gráfica anterior muestra, a modo de resumen, la cantidad de memoria utilizada relativa a cada componente de la aplicación. Para ser exacto, las partes relativas a las rutas y al Framework son

variables, siendo su relación de 0.01 a 0.05 MB por ruta y de 1.5 a 10 MB dependiendo del mapa.

Estos datos son bastante positivos en cuanto a eficiencia se refiere, algo muy importante en el mundo de las aplicaciones para móviles donde la autonomía esta restringida por una escasa batería.

- Geolocalización (GPS)

La posición del usuario se calcula perfectamente, el mapa se centra al instante en esa posición y se actualiza cada vez que el usuario se desplaza una distancia determinada, que por defecto esta configurada a diez metros. Esta distancia y la precisión con la que se calcula la posición del usuario se puede configurar en los parámetros del Framework [12], pero se debe tener en cuenta que a mayor precisión se produce mayor consumo de la batería.

Anexo D. Manuales

En este anexo se explicará la manera en la que los usuarios deben utilizar y configurar la aplicación iRoutes [11], también se añade la documentación técnica del Framework [12] que explica como debería utilizarlo cualquier desarrollador.

D.1. Documentación técnica del Framework gsl_mobileMap_iPhone

En este manual se explicará el uso principal de la librería para poder utilizarla en cualquier aplicación, sin necesidad de tener que estudiar el código en profundidad. Las explicaciones se centrarán en los métodos del Facade [13], ya que están diseñados para ser el centro de comunicación entre la librería y la aplicación que la use.

D.1.1. Descripción del producto

El componente gsl_mobileMap_iPhone implementa un visualizador de mapas que se utiliza dentro del entorno iOS. Los siguientes puntos describen las características más importantes de esta librería.

- Acceso OnLine a WMS-OGC. El visor es capaz de funcionar con cualquier servidor basado en especificación WMS-OGC v1.0.0 o superior.
- Acceso OnLine a WMS-C. El visor es capaz de funcionar con cualquier servidor basado en especificación WMS-C de OSGeo.
- Acceso OffLine a WMS-C. Una vez se han descargado las *tiles* necesarias, se almacenan en cache para poder acceder a ellas sin necesidad de conexión.
- Herramientas de navegación. La librería proporciona las herramientas básicas de navegación, zoom in/out y *panning*.
- Visualización de datos vectoriales. La librería permite añadir capas vectoriales al mapa con distinta geometría y estilo de visualización modificable. Esta información vectorial suele estar asociada a una *feature* que puede definirse como seleccionable y/o navegable para poder interactuar con ella de diversos modos. Un modo de crear estas capas vectoriales es a partir de la información contenida en un archivo GeoJSON.

- Geoposicionamiento. Da la posibilidad de utilizar el GPS del iPhone para geolocalizar al usuario y pintar su posición en el mapa. La posición del usuario se actualizara cada cierto tiempo mientras este activado el GPS.

D.1.2. Instalación

A continuación se detallan los pasos para, partiendo de cero, empezar a desarrollar en un proyecto que se base en esta librería hasta acabar instalando la aplicación resultante en la plataforma requerida (bien sea de desarrollo o producción).

Este módulo contiene todas las clases necesarias para crear un visor para iPhone. Para instalarlo, debemos seguir estos pasos:

(1) Abrir el proyecto con Xcode.

(2) Asegurarnos de que en la carpeta Copy Headers, dentro del grupo Targets, todos los archivos .h tienen el Role definido como public, excepto JSONKit que puede estar definido como project.

(3) Compilar para generar el framework gsl_mobileMap_iPhone y la carpeta usr donde se encontraran las cabeceras de las clases de la librería.

(4) En el nuevo proyecto, importar el archivo creado como librería existente y añadir también la carpeta usr.

(5) Por ultimo, bastará con importar el archivo de cabecera de la librería gsl_mobileMap_iPhone.h para tener acceso a todas las clases.

D.1.3. Acceso a la funcionalidad ofrecida por el Framework

Para poder utilizar este Framework [12] será suficiente con seguir los pasos de instalación detallados anteriormente. Si se han seguido correctamente, en alguna parte de nuestro código, tendremos la siguiente linea:

```
#import "gsl_mobileMap_iPhone.h"
```

Al importar la cabecera de la librería, ya tenemos acceso a todos sus componentes y sus respectivos métodos. El siguiente paso, debería ser la creación del mapa y el Facade [13], usando las siguientes instrucciones:

```
Map *map = [[Map alloc] initWithMap:MapWidth Height:MapHeight DispX:MapDispX
DispY:MapDispY];
Facade *facade = [[Facade alloc] initWithMap:map];
```

Sus parámetros definirán las características principales del mapa:

- MapWidth: Anchura del mapa, 320 en el constructor por defecto.

- MapHeight: Altura del mapa, 480 en el constructor por defecto.
- MapDispX: Desplazamiento en el eje X de la vista mapa, 0 por defecto.
- MapDispY: Desplazamiento en el eje Y de la vista mapa, 0 por defecto.

Una vez hemos creado el mapa y el Facade, bastará añadir como subvista de nuestra aplicación la vista del mapa:

```
[self.view addSubview:map.view];
```

Sin embargo, el mapa todavía no será visible, ya que no tiene asociado ningún servicio de mapas y no se le han añadido capas que visualizar. El siguiente ejemplo muestra cómo añadir una capa proveniente del servicio WMS-OGC IDEE-Base que sólo contiene la capa de hidrografía:

```
//url del servicio
NSString *url = @"http://www.idee.es/wms/IDEE-Base/IDEE-Base";
//parámetros de la petición y sus correspondientes valores
NSMutableArray *paramNames = [[NSMutableArray alloc] initWithArray: [NSArray
arrayWithObjects:@"TRANSPARENT",@"VERSION",@"BGCOLOR",@"SERVICE",@"REQU
EST",@"STYLES",@"EXCEPTIONS",@"FORMAT",nil]];
NSMutableArray *paramValues = [[NSMutableArray alloc] initWithArray: [NSArray
arrayWithObjects:@"true",@"1.3.0",@"0xFFFFFF",@"WMS",@"GetMap",@"default",@"applicat
ion/vnd.ogc.se_inimage",@"image/png", nil]];
//creamos una capa WMS con los parámetros anteriores
WMS *wms = [[WMS alloc] initWithName:@"OGC:WMS" Url:url
ParamName:paramNames ParamNameValue:paramValues];
//creamos una WMSLayer con el nombre de la capa que pediremos al servicio
WMSLayer *layer = [(WMSLayer*)[WMSLayer alloc] initWithName:@"Hidrografia"];
//añadimos la WMSLayer a la capa wms
[wms addLayer:layer];
//repetir los dos últimos pasos por cada capa que se quiera pedir al servicio
//...
//añadimos la capa wms al mapa
[map addLayer:wms];
```

Si se prefiere utilizar un servicio WMS-C en vez de un WMS-OGC, deben de añadirse varios parámetros más, como las resoluciones disponibles del servicio, su extensión máxima y los *tileSets*. El siguiente ejemplo utiliza un servicio WMS-C con datos de IDEZar:

```
//url del servicio
NSString *url = @"http://clamosa.cps.unizar.es:8080/TileCache_IDEZar/WMSTileCache?";
//parámetros de la petición y sus valores
NSMutableArray *paramNames = [[NSMutableArray alloc] initWithArray: [NSArray
arrayWithObjects:@"TRANSPARENT",@"VERSION",@"BGCOLOR",
@"SERVICE",@"REQUEST",@"STYLES",@"FORMAT",nil]];
NSMutableArray *paramValues = [[NSMutableArray alloc] initWithArray: [NSArray
arrayWithObjects:@"false",@"1.1.1",@"0xFFFFFF",@"WMS",@"GetMap",@"default",@"image/
jpeg", nil]];
//resoluciones soportadas por el servicio
```

```
        NSMutableArray *resoluciones =[[NSMutableArray alloc] initWithArray:[NSArray
arrayWithObjects:[NSNumber numberWithFloat: 3.4332275390625E-4], [NSNumber
numberWithFloat: 1.71661376953125E-4], [NSNumber numberWithFloat: 8.58306884765625E-5],
[NSNumber numberWithFloat: 4.291534423828125E-5], [NSNumber numberWithFloat:
2.1457672119140625E-5], [NSNumber numberWithFloat: 1.0728836059570312E-5], [NSNumber
numberWithFloat: 5.364418029785156E-6], nil]];
        //extensión máxima permitida, en este caso todo el mundo
        Extent *extensionMax = [(Extent *)[Extent alloc] initWithMinX:-180.0 MinY:-90.0
MaxX:180.0 MaxY:90.0 Projection:@"EPSG:4326"];
        //configuración de tiles que se utilizara
        TileSet *tileSet = [[TileSet alloc] initWithProjection:@"EPSG:4326" Extent:extensionMax
        resList:resoluciones Format:@"image/jpeg" tileWidth:defaultTileWidth
tileHeight:defaultTileHeight];
        //creamos la capa WMS-C con los parámetros anteriores
        WMSC *capaTileada = [[WMSC alloc] initWithName:@"OGC:WMS" Url:url
SingleTile:FALSE ParamName:paramNames ParamNameValue:paramValues];
        //creamos la WMSLayer con el nombre de la capa que pediremos al servicio
        WMSLayer *layer = [(WMSLayer *)[WMSLayer alloc] initWithName:@"base"];
        //añadimos el tileset a la capa
        [capaTileada addTileSet:tileSet];
        //añadimos la WMSLayer a la capa
        [capaTileada addLayer: layer];
        //añadimos la capa al mapa
        [map addLayer:capaTileada];
```

Con la capa añadida al mapa, tanto si es WMS-OGC como WMS-C, comenzarán a realizarse las peticiones necesarias que, una vez finalizadas, permitirán pintar el mapa en el contexto que se le haya especificado. Si no se hace ninguna otra modificación, se realizará una petición de la extensión máxima que permita el mapa, normalmente el mundo entero. En caso de querer un área especifica, existe la posibilidad de modificar la extensión del mapa con la siguiente instrucción:

```
        //definimos la extensión que queremos visualizar en el mapa
        Extent *newExtension = [[Extent alloc] initWithMinX:MinX MinY:MinY MaxX:MaxX
MaxY:MaxY Projection:@"EPSG:4326"];
        //modificamos la extensión del mapa
        [map setExtent:newExtension];
```

Los parámetros de la variable newExtension definen el *Bounding Box* o más comúnmente el área que definirá la extensión del mapa, especificando las coordenadas de las esquinas inferior izquierda y superior derecha de la caja o rectángulo. Aunque sea posible modificar la extensión del mapa por parte del programador, esta función solo suele usarse para establecer una extensión inicial y serán los controles de movimiento del mapa los que se encarguen de ir modificando los parámetros del mapa conforme el usuario se mueva por él. Estos controles de movimiento se añaden por defecto cuando se crea el mapa e incluyen el movimiento o *panning* y el cambio de nivel de zoom.

La librería nos proporciona otros dos controles, pero para poder usarlos deben ser creados y añadidos a la lista de controles del mapa. El siguiente ejemplo muestra como se añade el control del GPS:

```
// Añade un nuevo control al mapa, en este caso, el control del gps
- (void)addGPSControl {
        Control *newControl = [[GPSLocation alloc] initWithTarget:map.view action:
@selector(handleGesture:)];
        int numcontrols = [map addControl: newControl];
        [newControl release];
        [map.view addGestureRecognizer:[map.controls objectAtIndex:numcontrols-1]];
}
```

Para añadir el control de selección de *features* [14], bastará seguir los mismos pasos pero creando el control *FeatureSelector*, en vez del *GPSLocation*. Si hemos añadido el control *FeatureSelector*, estarán disponibles los siguientes métodos del Facade para seleccionar las *features* del mapa y poder trabajar con ellas:

```
- (FeatureSelector*)getSelector;
- (Feature*)getSelectedFeature;
- (Feature*)selectNextFeature;
- (Feature*)selectPreviousFeature;
- (Feature*)selectFeatureById:(NSString*)identificador;
```

El Facade también nos proporciona métodos para añadir o eliminar elementos del mapa:

```
- (Vector*)getVectorLayer;
- (int)addLayer:(Layer*)layer;
- (bool)removeLayer:(NSString*)identificador;
- (void)addPOIWithX:(double)x Y:(double)y Projection:(NSString*)projection Image:
(UIImage*)image Position:(NSString*)position Touchable:(bool)isTouchable;
- (void)setPosition:(NSString*)position OfMarker:(Marker*)marker;
- (void)addLine:(NSArray*)points WithStyle:(FeatureStyle*)style;
- (bool)addGeoJSON:(GeoJSON*)geoJSON Identificador:(NSString*)identificador
Centered:(bool)centerInData;
```

El resto de métodos del Facade sirven para modificar alguno de los parámetros del mapa, como puede ser su *extent* o también activar y desactivar *flags* o controles. Para una descripción más detallada del propósito de estos métodos, consultar el fichero Facade.h donde se encuentran definidos.

Por último, cabe destacar el uso de notificaciones que utiliza el Framework [12] para comunicar ciertos eventos de importancia. La mayoría de estas notificaciones se utilizan para comunicación interna de los objetos que componen el mapa de un modo transparente, pero al perder la conexión con los servicios, la notificación *connectionLosed* es capturada por el Facade [13] que ejecutará el método *onConnectionLosed*. Este método se encuentra sin implementar, para que el programador de la aplicación que use la librería lo implemente, por ejemplo, en una subclase del Facade, y le de la utilidad que considere necesaria, ya sea mostrar una alerta que indique conexión perdida o intente recuperar la conexión enviando un *refresh* al mapa.

Otra de las notificaciones importantes es la notificación *FeatureTouched*, se activa al tocar una *feature* [14] del mapa que pueda ser seleccionada. Al igual que la anterior, es capturada por el Facade para llamar al método *onFeatureTouched*, también sin implementar.

D.2. Documentación pública iRoutes

D.2.1. Descripción del producto

El componente iRoutes implementa un visualizador de rutas con acceso a servicios WMS-OGC y WMS-C. La aplicación permite importar rutas usando ficheros con formato GeoJSON alojados en local o en una URL. Los siguientes puntos describen las características más importantes de esta aplicación.

- Importación de rutas. Para poder ver la información de una ruta debemos importarla en la aplicación usando un archivo con formato GeoJSON que puede estar almacenado en local o en una URL.
- Visualización de las rutas importadas. Carga la información de la ruta sobre el mapa.
- Geolocalización por GPS. Utiliza el GPS del iPhone para localizar al usuario en el mapa.
- Navegación por el mapa. Permite navegación básica por el mapa, movernos y hacer zooms.
- Modo mostrar información. Muestra la información asociada a los POIs del mapa permitiendo seleccionar los POIs pulsando sobre ellos o con las flechas laterales que aparecerán sobre el mapa.

D.2.2. Instalación y ejecución

Para instalarlo, debemos seguir estos pasos:

(1) Abrir el proyecto con Xcode.

(2) Pulsar el botón "*Build and Run*", esto compilará la aplicación, la instalará y la ejecutará en el simulador.

D.2.3. Manual de usuario de la aplicación iRoutes

El interfaz de la aplicación iRoutes [11] consta de 2 vistas. La principal permite al usuario importar rutas, borrar las existentes o cargar la información de una de ellas. Al cargar una ruta, se pasa a la segunda vista, donde veremos el mapa con la información de la ruta y una serie de botones que nos permitirán actuar sobre el mapa.

Vamos a explicar más detenidamente cada una de las vistas y sus funcionalidades.

- Menú principal

Figura D.1. Menú principal iRoutes

El menú principal consta de la lista de rutas que ya han sido importadas anteriormente y los botones importar y borrar. Si pulsamos importar, se da a elegir la posibilidad de importar un archivo de rutas desde disco o usando una URL. Una vez seleccionado el modo, la vista cambia ligeramente de aspecto para dar la posibilidad al usuario de importar la ruta. Por otro lado, el botón borrar, nos permite eliminar la ruta seleccionada de la lista de rutas, pero no borrará el fichero que tengamos en disco.

Figura D.2. Importación de archivo usando una URL

En esta vista, si el usuario introduce una URL, el botón aceptar, descargará e importará la ruta enlazada por la URL, quedando añadida a la lista de rutas.

Figura D.3. Importación de archivo almacenado en disco

En el caso de importar desde disco, se permitirá seleccionar alguno de los archivos que se encuentran en la carpeta *documents* de la aplicación. El botón aceptar importará el archivo que se haya seleccionado.

En ambas vistas, el botón cancelar nos devuelve al menú principal, también volvemos automáticamente a este menú después de importar un archivo.

- Visualización de la ruta

Si seleccionamos una ruta, el botón importar se cambiará por el botón ver en mapa. Tanto si pulsamos este nuevo botón como si pulsamos dos veces sobre la misma ruta, pasamos a la segunda vista, donde se encuentran, el mapa con la información de la ruta cargada y tres botones, de izquierda a derecha, el botón de información, el de volver al menú principal y el botón del GPS. También podemos ver una barra inferior donde se muestra la distancia y duración de la ruta.

Figura D.4. Ruta seleccionada

Figura D.5. Visualización de ruta

Si activamos el botón de información, aparecen dos flechas a los laterales que nos permiten navegar por los POIs [10], además, uno de los POIs aparecerá marcado con una flecha sobre él y se mostrará la información asociada a ese POI en la barra inferior.

Figura D.6. Aplicación mostrando información de un POI

En este modo, también podemos seleccionar los POIs [10] directamente pulsando sobre ellos.

El botón del GPS nos permite activar la geolocalización y una vez calculada nuestra posición, se mostrará en el mapa como un POI más, pero sin dar la posibilidad de seleccionarlo.

Figura D.7. GPS desactivado

Figura D.8. GPS activado

En todas las vistas se utiliza la pantalla táctil para interactuar con la aplicación. Dependiendo del tipo de gesto que se realice, se provocarán diferentes comportamientos.

Los gestos básicos a los que responde la aplicación son:

- toque simple (*single tap*): se utiliza para seleccionar elementos, activar botones, etc.
- toque simple + arrastrar (*tap and drag*): se utiliza para mover ciertos elementos, como puede ser la lista de rutas o el mapa.
- toque doble (*double tap*): se utiliza para cargar rutas seleccionadas o para realizar zoom in en la vista del mapa.
- doble toque (*double touch*): se diferencia del toque doble en que se realiza con los dos dedos a la vez, se utiliza en la vista del mapa para realizar zoom in/out al separar o juntar los dedos.

Por último el botón *home*, desactiva los botones de información y del GPS, si estaban activos, y nos devuelve a la vista principal, donde podremos cargar una ruta diferente o importar una nueva.

Anexo E. Servicios Web Estándar

En este apéndice se explican los detalles de funcionamiento de cada uno de los servicios Web estándar de *OGC* [2] utilizados en este proyecto, incluyendo los tipos de peticiones soportados y los parámetros más importantes de éstas.

E.1. Servicio web de mapas WMS-OGC

El servicio *Web Map Service* (*WMS*) definido por el *OGC* [2] produce mapas dinámicamente a partir de datos espaciales georeferenciados . Según este estándar internacional, un 'mapa' es una representación de la información geográfica en forma de un archivo de imagen digital para ser presentada en la pantalla de un ordenador (generalmente en formato de imagen como *PNG*, *GIF* o *JPEG*, y ocasionalmente como gráficos vectoriales en SVG o WebCGM). Un mapa no consiste en los propios datos, sino en su representación.

El estándar define tres operaciones principales:

- Devolver metadatos del nivel de servicio (*GetCapabilities*).
- Devolver un mapa cuyos parámetros geográficos y dimensionales han sido bien definidos (*GetMap*).
- Devolver información de características particulares mostradas en el mapa (*GetFeatureInfo*).

Las operaciones *WMS* pueden ser invocadas usando un navegador estándar realizando peticiones en la forma de *URLs* (*Uniform Resource Locators*). El contenido de tales *URLs* depende de la operación solicitada. Concretamente, al solicitar un mapa, la *URL* indica qué información debe ser mostrada en el mapa, qué porción de la tierra debe dibujar, el sistema de coordenadas de referencia, y la anchura y la altura de la imagen de salida. Cuando dos o más mapas se producen con los mismos parámetros geográficos y tamaño de salida, los resultados se pueden solapar para producir un mapa compuesto. El uso de formatos de imagen que soportan fondos transparentes permite que los mapas subyacentes sean visibles. Además, se puede solicitar mapas individuales de diversos servicios.

En resumen, los parámetros obligatorios más importantes a incluir en una petición del tipo *GetMap* son los siguientes:

- *Service*: en este caso, *WMS*.
- *Version*: versión del *WMS*.
- *Request*: tipo de petición (en este caso, *GetMap*).
- *Layers*: capas del mapa solicitadas (si hay más de una se superponen).
- *Styles*: estilo de la imagen a generar (uno por capa).
- *CRS*: sistema de coordenadas.
- *BBox*: rectángulo que define el área geográfica solicitada.
- *Width* y *Height*: dimensiones en *pixels* de la imagen a generar.
- *Format*: formato de la imagen a generar.

Este es un ejemplo de una petición a un servicio de este tipo y su correspondiente respuesta:

http://www.cartociudad.es/wms/CARTOCIUDAD/CARTOCIUDAD?TRANSPARENT=true&VERSION=1.1.1&BGCOLOR=0xFFFFFF&SERVICE=WMS&REQUEST=GetMap&STYLES=toponimo,codigo_postal,seccion_censal,portal_pk,vial,manzanas,comunidadautonoma&EXCEPTIONS=&FORMAT=image/png&LAYERS=Toponimo,CodigoPostal,SeccionCensal,Portal,Vial,FondoUrbano,DivisionTerritorial&BBOX=-0.8960294723511,41.6553016119666,-0.8685636520386,41.6947837286658&SRS=EPSG:4326&WIDTH=320&HEIGHT=460

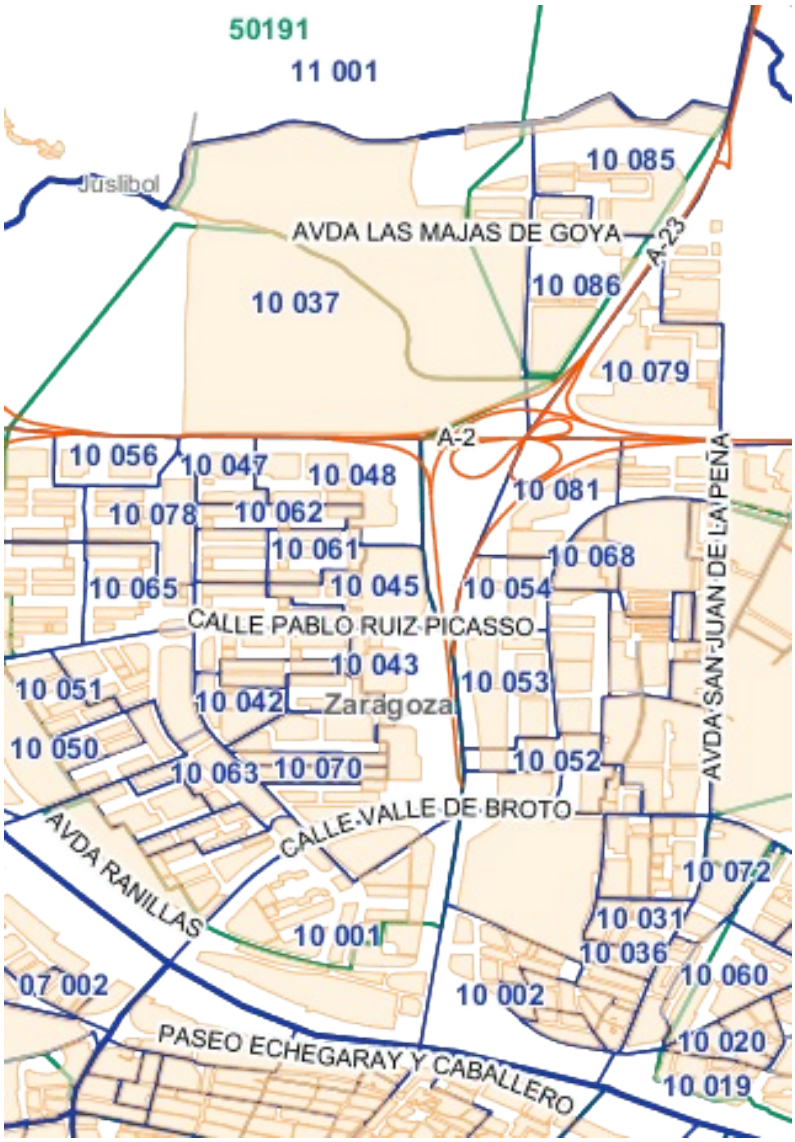

Figura E.1. Resultado de una petición a un servicio WMS-OGC

E.2. WMS Tile Caching de OSGeo

El objetivo del almacenamiento en caché de WMS tileados [4], también llamados WMS-C, es encontrar una manera de optimizar la entrega de las imágenes de mapas a través de internet. Este servicio todavía no es un estándar, debido a que se encuentra todavía en fase de borrador. Por eso su definición no es tan precisa como la de los servicios WMS. Una de las definiciones más comunes especifica a los servicios WMS-C como perfiles de los WMS.

WMS-C como perfil de WMS-OGC

Una posible definición de WMS-C es considerarlo un perfil limitado de WMS-OGC que permite a los servicios optimizar su generación de imágenes y que los *tiles* [4] se almacenen en cache en puntos intermedios. Un servicio WMS-C sólo ofrece imágenes de cuadrados delimitados y alineadas con una malla de origen determinado, y sólo en ciertos niveles de escala.

La idea básica es que, a diferencia de WMS, dos solicitudes para una *tile* WMS-C deben tener exactamente la misma petición HTTP. Esto limita las peticiones *GetMap* de los WMS-OGC:

1. Número de argumentos de consulta mínimos (por ejemplo, no permitir argumentos opcionales)
2. Número de argumentos fijo tanto en casos como en consultas
3. Intervalo fijo de posibles dimensiones de las cajas, calculada a partir de parámetros del perfil del WMS-C
4. Precisión fija en los valores del cuadro delimitadores
5. Tamaño de *tile* fijo en píxeles
6. Nombre de la capa fija y/o nombre de la capa ordenado
7. Estilo fijo
8. Formato de salida fija

Calculo de extensión valida de tiles para una solicitud determinada

El almacenamiento en caché de *tiles* [4] WMS-OGC implica fijar la escala o los niveles de zoom. Por lo general, cada nivel de escala válida es la mitad de la escala inmediatamente superior. Es un valor de referencia fijado que se usa para calcular cuantas *tiles* se necesitan para cubrir por completo una caja delimitada dada una escala determinada.

En este caso, los parámetros obligatorios a incluir en una petición, están más restringidos y a pesar de ser parecidos a los de un WMS-OGC, la mayoría son fijos y pueden consultarse realizando una petición *GetCapabilities* al servicio. Partiendo de los datos recibidos, los parámetros de una petición del tipo *GetMap* serían los siguientes:

- *Service*: también este caso, WMS.
- *Versión*: versión del *WMS*.
- *Request*: tipo de petición (en este caso, *GetMap*).
- *Layers*: normalmente restringido a un único conjunto de layers.
- *Styles*: suele ser único y llamado *default*.
- *SRS*: solo hay un sistema de coordenadas permitido, el EPSG:4326.
- *BBox*: rectángulo que define el área geográfica solicitada, si no se corresponde con el área concreta de una de las *tiles*, es servicio devolverá un error.
- *Width* y *Height*: dimensiones en *pixels* de la imagen a generar, generalmente fijados en 256x256.
- *Format*: también suele ser único.

Para facilitar la construcción de las peticiones a estos servicios, los parámetros Layers, Styles, SRS, BBox máxima, Width, Heigth y Format se agrupan en los llamados *TilSet* o conjunto de *tiles* [4] y se añaden las resoluciones de todos los niveles de escala o zooms que permite el servicio. De este modo, basta con seleccionar uno de los *TileSet* ofrecidos, para rellenar automáticamente la mayoría de los parámetros de las peticiones.

Este es un ejemplo de una petición a un servicio de este tipo y su correspondiente respuesta:

http://clamosa.cps.unizar.es:8080/TileCache_IDEZar/WMSTileCache?&TRANSPARENT=false&VERSION=1.1.1&BGCOLOR=0xFFFFFF&SERVICE=WMS&REQUEST=GetMap&STYLES=default&FORMAT=image/jpeg&LAYERS=base&BBOX=-0.9008789062500,41.6601562500000,-0.8789062500000,41.6821289062500&SRS=EPSG:4326&WIDTH=256&HEIGHT=256

Figura E.2. Resultado de la petición de una tile a un servicio WMS-C

Glosario

PFC [1]: Proyecto Fin de Carrera.

OGC [2]: *Open Geospatial Consortium*. Grupo creado en 1994 que agrupa a más de 250 organizaciones públicas y privadas cuyo fin es la definición de estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica. Persigue acuerdos entre las diferentes empresas del sector que posibiliten la interoperación de sus sistemas de geoprocesamiento y facilitar el intercambio de la información geográfica en beneficio de los usuarios. Anteriormente fue conocido como *Open GIS Consortium*. Algunas de las especificaciones más importantes surgidas de OGC son los servicios *WMS* o *WFS* o el formato *GML*.

OSGeo [3]: *Open Source Geospatial Foundation*. Organización no gubernamental sin ánimo de lucro formada en febrero del 2006 cuya misión es el soporte y promoción del desarrollo colaborativo de tecnologías geoespaciales. Provee así soporte financiero, organizativo y legal a la comunidad de código abierto encargada de trabajar en este tipo de desarrollos.

Tile [4]: Tesela, en español. Porción cuadrada que representa una parte de la extensión total de un área geográfica. La unión de un conjunto de ellos correctamente dispuestos puede conformar un mapa, una cobertura o un conjunto de *features*, entre otras cosas.

Smalltalk [5]: *Smalltalk* es un lenguaje de programación que permite realizar tareas de computación mediante la interacción con un entrono de objetos virtuales. Metafóricamente, se puede considerar que un *Smalltalk* es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

NeXTSTEP [6]: *NeXTSTEP* es el sistema operativo orientado a objetos, multitarea que NeXT Computer, Inc. diseñó para correr en los ordenadores *NeXT* (informalmente conocidos como “black boxes”).

GPL [7]: General Public License es una licencia creada por la Free Software Foundation y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

GCC [8]: GNU Compiler Collection (colección de compiladores GNU) es un conjunto de compiladores creados por el proyecto *GNU*. *GCC* es software libre y lo distribuye la *FSF* (Free Software Foundation) bajo la licencia *GPL*. Estos compiladores se consideran estándar para los sistemas operativos derivados de *UNIX*, de código abierto o también de propietarios, como *Mac OS X*.

GNUstep [9]: *GNUstep* es un conjunto de *Frameworks* o bibliotecas orientadas a objetos, aplicaciones y herramientas escritas en el lenguaje *Objective-C*, para el desarrollo de aplicaciones de escritorio.

POI [10]: Point Of Interes, o Punto De Interés (PDI) en español, es una localización puntual especifica que alguien puede encontrar útil o de interés. El término es ampliamente utilizado en cartografía, especialmente en las variantes electrónicas, incluido el SIG y el software de navegación GPS. Un punto de interés GPS especifica, como mínimo, la latitud y longitud del punto de interés, asumiendo un sistema de referencia geodésico determinado. Normalmente se incluye un nombre o una descripción del lugar de interés, y otra información como la altitud o un número de teléfono puede estar asociado al POI. Las aplicaciones GPS suelen utilizar iconos para representar las diferentes categorías de POIs en el mapa de forma gráfica.

iRoutes [11]: Aplicación desarrollada en este proyecto, que pretende ofrecer al usuario un interfaz intuitivo mediante el cual pueda importar un fichero con formato GeoJSON donde se encuentre definida una ruta y un conjunto de POIs, para posteriormente visualizarlo sobre un mapa.

Framework [12]: Estructura de soporte, que permite desarrollar una aplicación sobre él. En general, define la arquitectura básica de la aplicación y provee de servicios que agilizarán y simplificarán el desarrollo del proyecto. Un *Framework* representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. Son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional.

Facade [13]: El patrón Facade se utiliza para proporcionar una interfaz unificada de alto nivel para un conjunto de clases en un subsistema, haciéndolo más fácil de usar. Simplifica el acceso a dicho conjunto de clases, ya que el cliente sólo se comunica con ellas a través de una única interfaz.

Feature [14]: Se puede definir como el elemento básico de información geográfica que describe una entidad geográfica real o abstracta. Cada elemento esta compuesto de una serie de atributos que lo describen cuantitativa y cualitativamente. A su vez cada elemento puede contener recursivamente otras *features*. Ejemplos de *features* son:

- Un segmento de carretera entre dos puntos
- Una autopista compuesta de varios segmentos de carretera
- Un conjunto de isobaras de un mapa de presión.

GML [15]: *Geography Markup Language* (Lenguaje de Marcado Geográfico). Sublenguaje de *XML* descrito como una gramática en *XML Schema* para el modelado, transporte y almacenamiento de información geográfica.

Bibliografía

Become an Xcoder, By Bert Altenberg, Alex Clarke and Philippe Mougin (v1.15) (2008)
http://www.cocoalab.com/BecomeAnXcoder.pdf

From C++ to Objective-C version 2.1, Pierre Chatelier (v2.1) (2009)
http://pierre.chachatelier.fr/programmation/objective-c.php

El lenguaje Objective-C para programadores C++ y Java (2008)
http://www.etnassoft.com/biblioteca/el-lenguaje-objective-c-para-programadores-c-y-java/

The Objective-C Programming Language (2011)
http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

OpenGIS Web Map Server Implementation Specification (v1.3.0) (2006)
http://portal.opengeospatial.org/files/?artifact_id=14416

Algunas Web de interés:

Especificaciones de OpenGis: http://www.opengeospatial.org/

Página oficial de OpenLayers: http://openlayers.org/

Página oficial de GeoSpatiumLab: http://www.geoslab.com

Página oficial de IDEE: http://www.idee.es - http://idee.unizar.es

Especificaciones de OSGeo: http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

Estándar Tile Map Service o WMS Tile Caching: http://wiki.osgeo.org/wiki/WMS_Tile_Caching

Estudio sobre Frameworks para *parsear* archivos JSON:
http://www.cocoanetics.com/2011/03/json-versus-plist-the-ultimate-showdown/

Especificación del formato GeoJSON: http://geojson.org/geojson-spec.html

Página del Framework RouteMe:
http://iphone-dev-tips.alterplay.com/2009/12/routeme-is-alternative-map-control-to.html

Manual de referencia del Framework MapKit:

http://developer.apple.com/library/ios/#documentation/MapKit/Reference/MapKit_Framework_Reference/_index.html

Información y código del Framework JSONKit: https://github.com/johnezang/JSONKit