CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Explotación de servicios de la Infraestructura de Datos Espaciales de España (IDEE) en Dispositivos Móviles

Autor: Alejandro Vera De Juan

Director: Francisco Javier Soriano Camino

A mi abuelo Ramón, allí donde esté.
Espero que se sienta orgulloso de su nieto.

AGRADECIMIENTOS

Me gustaría agradecer a Javier Soriano y Genoveva López el haberme dado la oportunidad de formar parte del Computer Networks and Web Technologies Lab y poder trabajar, tanto en el laboratorio (que ha llegado a convertirse también en el cuartel general de redacción de prácticas), como en unas instalaciones tan modernas como son las del Center for Open Middleware. Ha sido una gran experiencia, personal y profesional.

Por supuesto, debo de nuevo agradecerle a Javier, que ha sido mi tutor, todo el apoyo que me ha prestado durante el desarrollo del proyecto y de esta memoria, así como su aspecto crítico que me ha ayudado a mejorar el resultado de este proyecto. También debo agradecerle que haya decidido tomar un avión sólo para estar presente el día de la exposición del TFG. Gracias.

Debo agradecer al equipo del Conwet en el COM toda su ayuda durante este proyecto. Alex, Carlos, Santi, Álvaro; muchas gracias, me habéis ayudado mucho y habéis conseguido que, tras el cambio del laboratorio al COM, me haya sentido como en casa. También quiero agradecerle a todos los miembros del laboratorio del Conwet su ayuda, especialmente a Sergio por hacerme ver la importancia de las pruebas.

No quiero olvidar agradecerle a mi familia el apoyo que me han dado durante toda mi vida, apoyándome en las decisiones que he tomado. Mis padres, mis abuelos, que me han aguantado y animado cada vez que tenía dificultades y que me han animado a seguir esforzándome en los momentos de éxito.

Imposible no agradecerle a mis amigos de la Facultad (o ETSIINF, aunque para mi siempre será la Facultad) el que hagan que cada vez que me despierte quiera ir a la Facultad y no quedarme en casa, sin vosotros no podría. Muchas gracias Adam, Borja, Camilo, Peter (que ahora mismo está en el extranjero), Wei...

Igualmente a mis amigos del colegio, que consiguen amenizarme los fines de semana y hacen que mantenga mis habilidades lectoras con esos cientos de mensajes diarios de Whatsapp. También a mis compañeros de tenis, que hacen que continuamente me tenga que esforzar para ganar el punto, lo cual me ha ayudado a aprender a no bajar nunca la guardia y esforzarme por mejorar.

Y por último agradecerles a todos los profesores la formación que me han dado. A Sonia Frutos, Rafael Fernández y Miguel Jiménez por conseguir que el mundo de las redes me parezca maravilloso. A Dolores Lodares, Antonio Tabernero, Antonio Giraldo, Elena Castiñeira, porque me han enseñado que las matemáticas pueden ser muy entretenidas. A Pablo Nogueira, un crack explicando Algoritmos y Estructuras de datos. También a Fernando Pérez Costoya que me ha ayudado mucho, tanto en las prácticas de la Facultad, como en dudas que tenía para mis proyectos personales. Y en general, a todos los profesores de la Facultad puesto que de todos he aprendido cosas nuevas.

GRACIAS

Alejandro Vera De Juan

RESUMEN

En los últimos años el número de dispositivos móviles y *smartphones* ha aumentado drásticamente, así como el número de aplicaciones destinadas a estos. Los desarrolladores siempre se han visto frenados en la creación de estas aplicaciones debido a la complejidad que supone la diversidad de sistemas operativos (Android, iOS, Windows Phone, etc), que utilizan lenguajes de programación diferentes, haciendo que, para poder desarrollar una aplicación que funcione en estas plataformas, en verdad haya que implementar una aplicación independiente para cada una de las plataformas.

Para solucionar este problema han surgido *frameworks*, como Appcelerator Titanium, que permiten escribir una sola vez la aplicación y compilarla para las diferentes plataformas móviles objetivo. Sin embargo, estos *frameworks* están aún en estado muy temprano de desarrollo, por lo que no resuelven toda la problemática ni dan una respuesta completa a los desarrolladores.

El objetivo de este Trabajo de Fin de Grado ha sido contribuir a la evolución de estos *frameworks* mediante la creación de un módulo para Appcelerator Titanium que permita construir de manera ágil aplicaciones multiplataforma que hagan uso de visualizadores de información geográfica. Para ello se propone el desarrollo de un módulo de mapa con soporte para capas WMS, rutas y polígonos en WKT, KML y GeoJSON. Se facilitará además que estas aplicaciones puedan acceder a capacidades del hardware como la brújula y el GPS para realizar un seguimiento de la localización, a la vez que se hace uso de la aceleración por el hardware subyacente para mejorar la velocidad y fluidez de la información visualizada en el mapa.

A partir de este módulo se ha creado una aplicación que hace uso de todas sus características y posteriormente se ha migrado a la plataforma Wirecloud4Tablet como componente nativo que puede integrarse con otros componentes web (*widgets*) mediante técnicas de *mashup*. Gracias a esto se ha podido fusionar por un lado todas las ventajas que ofrece Wirecloud para el rápido desarrollo de aplicaciones sin necesidad de tener conocimientos de programación, junto con las ventajas que ofrecen las aplicaciones nativas en cuanto a rendimiento y características extras.

Usando los resultados de este proyecto, se pueden crear de manera ágil aplicaciones composicionales nativas multiplataforma que hagan uso de visualización de información geográfica; es decir, se pueden crear aplicaciones en pocos minutos y sin conocimientos de programación que pueden ejecutar diferentes componentes (como el mapa) de manera nativa en múltiples plataformas. Se facilita también la integración de componentes nativos (como es el mapa desarrollado) con otros componentes web (*widgets*) en un *mashup* que puede visualizarse en dispositivos móviles mediante la plataforma Wirecloud.

ABSTRACT

In recent years the number of mobile devices and smartphones has increased dramatically as well as the number of applications targeted at them. Developers always have been slowed in the creation of these applications due to the complexity caused by the diversity of operating systems (Android, iOS, Windows Phone, etc), each of them using different programming languages, so that, in order to develop an application that works on these platforms, the developer really has to implement a different application for each platform.

To solve this problem frameworks such as Appcelerator Titanium have emerged, allowing developers to write the application once and to compile it for different target mobile platforms. However, these frameworks are still in very early stage of development, so they do not solve all the difficulties nor give a complete solution to the developers.

The objective of this final year dissertation is to contribute to the evolution of these frameworks by creating a module for Appcelerator Titanium that permits to nimbly build multi-platform applications that make use of geographical information visualization. To this end, the development of a map module with support for WMS layers, paths, and polygons in WKT, KML, and GeoJSON is proposed. This module will also facilitate these applications to access hardware capabilities such as GPS and compass to track the location, while it makes use of the underlying hardware acceleration to improve the speed and fluidity of the information displayed on the map.

Based on this module, it has been created an application that makes use of all its features and subsequently it has been migrated to the platform Wirecloud4Tablet as a native component that can be integrated with other web components (widgets) using mashup techniques. As a result, it has been fused on one side all the advantages Wirecloud provides for fast application development without the need of programming skills, along with the advantages of native apps, such as performance and extra features.

Using the results of this project, compositional platform native applications that make use of geographical information visualization can be created in an agile way; ie, in a few minutes and without having programming skills, a developer could create applications that can run different components (like the map) natively on multiple platforms. It also facilitates the integration of native components (like the map) with other web components (widgets) in a mashup that can be displayed on mobile devices through the Wirecloud platform.

Índice de contenido

Índice de ilustraciones

Índice de tablas

1 INTRODUCCIÓN

1.1 Visión general

En los últimos años el número de dispositivos móviles ha aumentado enormemente, sobretodo el de los llamados *smartphones* y el de las tabletas. En una década se ha pasado de unos dispositivos móviles utilizados únicamente para realizar llamadas, a unos *smartphones* con potencia equiparable a la de ordenadores de hace unos años, pantallas de alta resolución y conexiones a Internet de alta velocidad.

Según un estudio realizado por Emarketer [1], un 63,5% de la población mundial tiene actualmente un móvil y hay una tendencia a que este porcentaje aumente, aunque a menor velocidad (Ilustración 1). Además, según ese estudio actualmente el 38,5% de esos usuarios de teléfonos móviles lo son de *smartphones* y se espera que este porcentaje llegue al 48,8% en 2017 suponiendo un incremento de 750 millones de usuarios de *smartphones*, sumando así un total de 2.900 millones de usuarios de *smartphones* en 2017. Además, el uso de Internet en 2012 era del 38,5% en los propietarios de móviles y se espera que sea del 57,8% en 2017.

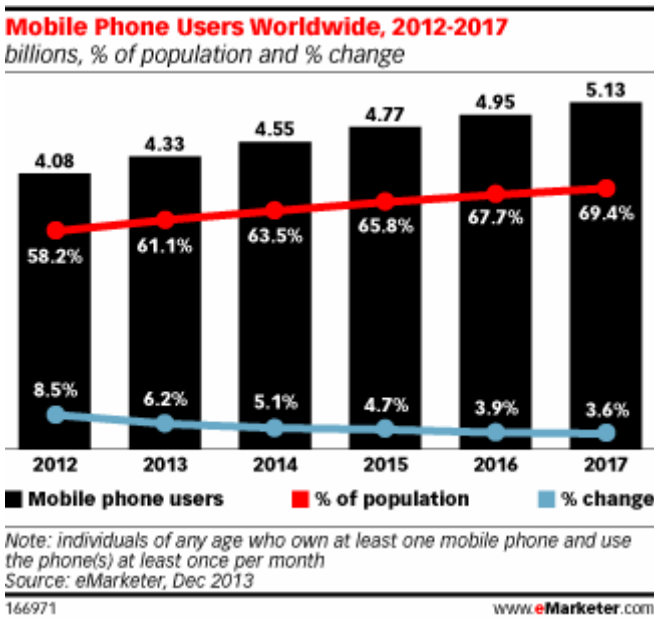


Ilustración 1: Usuarios de teléfonos móviles.
Fuente: Emarketer

Por otra parte, según IDC [23], las *tablets* están experimentando un gran crecimiento en los últimos años [22]. En el año 2013 experimentaron un crecimiento del 51,6%, aunque en 2014 se prevé que se reduzca este crecimiento al 19,4%. En la siguiente tabla se presenta el crecimiento del número de *tablets* diferenciando entre el uso comercial y el de los consumidores.

Año	Comercial	Consumidores
2013 Real	11%	89%
2014 Previsión	14%	86%
2018 Previsión	18%	82%

Tabla 2: Crecimiento de mercado de tablets (Fuente: IDC)

Este gran boom de los *smartphones* ha supuesto un aumento del desarrollo de aplicaciones móviles que sean capaces de explotar todas las posibilidades que estos móviles ofrecen y que permitan a las empresas acceder así a un enorme mercado en constante crecimiento.

Según un estudio realizado por Smartinsights[2], los usuarios de dispositivos móviles pasan el 89% de su tiempo de uso de estos dispositivos en aplicaciones nativas, frente a un 11% en *webs* (Ilustración 2). Por este motivo, muchas organizaciones han lanzado en los últimos años aplicaciones para acceder a sus servicios que amplíen los servicios que daban en sus páginas web a los usuarios de dispositivos móviles.

Sin embargo, este crecimiento de aplicaciones móviles se ha visto dificultado por la variedad de sistemas operativos móviles (Tabla 1). Al no existir un sistema operativo único, sino varios completamente diferentes, los desarrolladores deben elegir para qué sistema operativo realizar su aplicación y, en caso de decidir hacerla para todos, deberá hacer una aplicación desde cero para cada uno de los sistemas operativos, ya que los lenguajes de programación que se usan en cada uno de ellos son diferentes.

% de mercado de los SSOO de smartphones	Q4 '12	2012	Q4 '13	2013
Android	70.3%	68.8%	78.4%	78.9%
Apple iOS	22.0%	19.4%	17.6%	15.5%
Microsoft	2.7%	2.7%	3.2%	3.6%
Others	5.0%	9.1%	0.7%	2.0%

Tabla 1: Tasa de mercado de SSOO móviles (Fuente: Strategy Analytics)

Para facilitar el desarrollo multiplataforma y solucionar estos problemas con los que se encuentran los desarrolladores de A*pps* móviles han surgido *frameworks* como Appcelerator, PhoneGap, MoSync, etc.

Estos *frameworks* se basan en el uso de un lenguaje común de programación (generalmente Javascript) mediante el cual se hace uso de las APIs del *framework* que internamente están programadas en el lenguaje nativo de cada uno de los sistemas operativos móviles pero ofrecen una interfaz en el lenguaje común. A la hora de generar la aplicación, se enlaza automáticamente con las APIs correspondientes para el sistema

operativo para el que se esté compilando la aplicación, a pesar de que toda la programación de la aplicación se haya realizado de manera independiente del sistema operativo. De esta manera, la aplicación se escribe una sola vez y puede funcionar en todos los sistemas operativos que soporte el *framework* en las APIS que se utilicen.

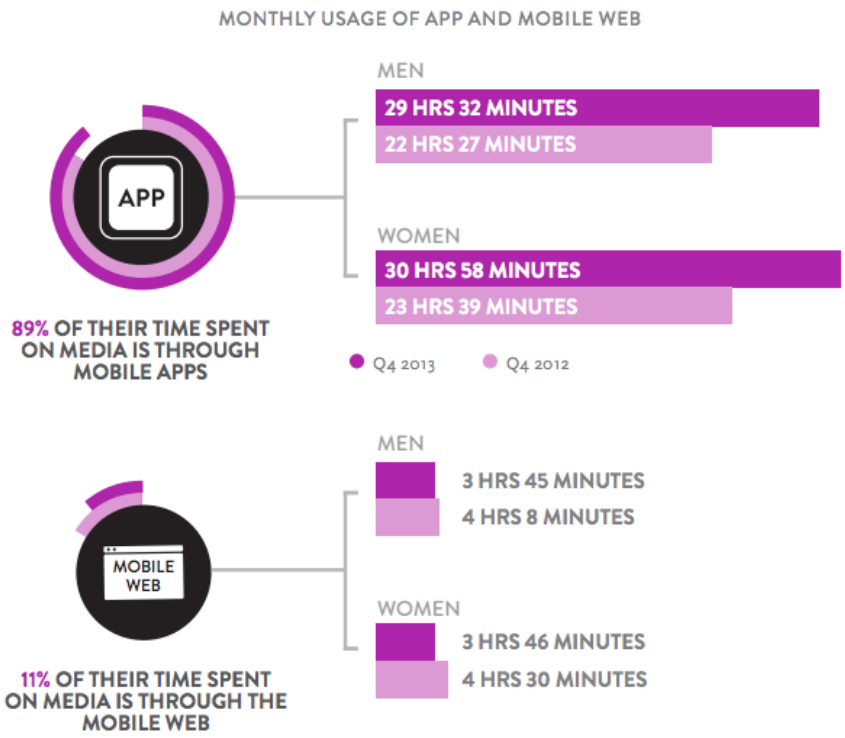


Ilustración 2: Diferencia de uso de apps y webs.

Fuente: Smartinsights

1.2 Objetivos del Trabajo

Este proyecto surge de un marco de colaboración existente entre el Laboratorio CoNWeT y el Centro Nacional de Información Geográfica para facilitar la explotación de servicios de la IDEE de España mediante las tecnologías de *mashup* (plataforma Wirecloud) y extiende esta colaboración para considerar también la explotación de estos servicios en aplicaciones móviles.

Wirecloud es una plataforma que permite la construcción de aplicaciones (*mashups*) rápidamente mediante la conexión de *widgets*. Los *widgets* son pequeñas aplicaciones que contienen puntos de entrada y de salida de datos que les permiten colaborar con otros *widgets* para formar un *mashup*. Mediante el *wiring* se puede conectar, por ejemplo, la salida de un *widget* a la entrada de otro. Así, mediante la combinación de diferentes *widgets* se pueden llegar aplicaciones realmente potentes simplemente arrastrando *widgets* y conectándolos de manera gráfica.

El objetivo de este proyecto consiste en desarrollar un conjunto de herramientas y una

App para facilitar la explotación de estos servicios en plataformas de dispositivos móviles como smartphones y *tablets*.

A continuación se explican los objetivos principales del TFG revisados respecto a la propuesta inicial de trabajo. En el apartado 6 Apéndice A: Plan de trabajo, se encuentran los objetivos de la planificación inicial y la revisada y se justifican los cambios realizados respecto a la planificación inicial. Los objetivos revisados son:

- Desarrollo de un módulo para Appcelerator

  Appcelerator es un *framework* que simplifica el desarrollo de aplicaciones para múltiples plataformas móviles escribiendo el código una única vez. Este código es código Javascript (más concretamente NodeJS) que hace uso de módulos de Titanium escritos en código nativo. En tiempo de compilación, se cargan los módulos correspondientes a la plataforma para la que se está compilando y así, a partir de un mismo código, se pueden obtener aplicaciones nativas para distintos sistemas operativos móviles (aplicaciones multiplataforma).

  En este caso, se busca desarrollar un módulo de mapa que soporte las siguientes características:

  1. Posicionamiento GPS y orientación mediante brújula.
  2. Soporte para capas WMS.
  3. Soporte para visualización de elementos vectoriales WKT, KML o GeoJSON.
  4. Soporte para insertar elementos geolocalizables.

- Desarrollo de una App de mapas

  Se busca desarrollar una aplicación de Appcelerator que haga uso del módulo anteriormente descrito. Esta aplicación debe presentar las siguientes características:

  1. Navegación por el mapa y controles.
  2. Visualización de la capa base y superposición de capas.
  3. Consultas de información de capas (GetFeatureInfo).
  4. Visualizador de leyendas de las capas.
  5. Buscador de lugares (WFS).
  6. Gestionar favoritos.
  7. Visualizador jerárquico de servicios y capas disponibles (búsqueda, selección), con posibilidad de añadir nuevos servicios y capas.

- Integración con la plataforma Wirecloud4Tablet

  Se busca realizar la integración con la plataforma Wirecloud4Tablet, la cual se trata de una plataforma que permite la creación de *mashups* (al igual que Wirecloud) compuestos por *widgets* que pueden ser nativos o escritos en Javascript y que pueden utilizar la funcionalidad que proporcionan los dispositivos móviles como puede ser GPS, brújula, acelerómetro, cámara, etc. Si se encuentra disponible un *widget* nativo para la plataforma del usuario, se cargará de forma totalmente transparente, produciendo una gran mejora en el rendimiento respecto a la versión escrita con tecnologías web.

  El objetivo es que la App nativa pase a ser un *widget* de la plataforma y pueda ser utilizado con otros *widgets* de la plataforma aunque estos no sean en nativo sino escritos en HTML, CSS y Javascript (aplicaciones híbridas).

1.3 Logros obtenidos

Como resultado de este Trabajo de Fin de Grado se han alcanzado los siguientes logros:

- Se ha conseguido realizar un módulo de mapa para poder ser utilizado en Appcelerator Titanium con capacidad para gestionar múltiples capas WMS así como de visualizar ficheros en los formatos WKT, KML y GeoJSON, gestionar desde el hardware el posicionamiento GPS y la brújula, y soportar gran cantidad de POIs (Points of Interest) gracias a la velocidad de las aplicaciones nativas.

  Al haber creado un módulo para Titanium con semejantes características, se ha abierto todo un nuevo abanico de posibilidades para el desarrollo de aplicaciones móviles que requieran de visualización de información geográfica. Un ejemplo de este tipo de aplicaciones es la que se ha desarrollado en este Trabajo de Fin de Grado (ver apartado 3.3 Desarrollo de la aplicación para Appcelerator Titanium).

- Se ha conseguido integrar esta aplicación con Wirecloud4Tablet, creando así un *widget* nativo que pueda sustituir al *widget* de Wirecloud en todos los *mashups* que lo utilicen. De esta manera, se pueden explotar todas las ventajas que ofrece una aplicación nativa a la vez que se mantienen las ventajas que ofrece Wirecloud para facilitar el desarrollo de aplicaciones composicionales de manera rápida y sencilla (en poco minutos y sin necesidad de saber programar gracias a Wirecloud). Lógicamente, los *mashups* que utilicen este componente nativo, sólo podrán visualizarse en dispositivos móviles que tengan instalada además la Base App de Wirecloud (Wirecloud4Tablet).

  Además, este hecho es muy importante para el Instituto Geográfico Nacional (que ya estaba interesado en el desarrollo de aplicaciones *web* de escritorio mediante Wirecloud) ya que sólo dispone actualmente de un visualizador *web* y este avance les permitirá poder empezar a llevar sus servicios a plataformas móviles, las cuales están en auge y demandarán cada vez más este tipo de servicios.

- Combinando los dos logros anteriores, se puede decir que mediante este Trabajo

de Fin de Grado se han desarrollado los elementos que faltaban para poder crear aplicaciones composicionales nativas multiplataforma que hagan uso de visualización de información geográfica; es decir, se pueden crear aplicaciones en pocos minutos y sin conocimientos de programación que se pueden ejecutar de manera nativa (al menos el mapa) en múltiples plataformas (aunque en este TFG solo se haya desarrollado para Android al no haber tiempo para realizarlo para más plataformas).

2 ESTADO DEL ARTE

2.1 Sistemas operativos móviles

En este apartado se analizarán los principales sistemas operativos móviles desde el punto de vista del desarrollador [3], haciendo especial hincapié en Android al ser el objetivo principal de este Trabajo de Fin de Grado.

2.1.1 Android

Android es un sistema operativo móvil desarrollado por Android Inc con un estilo arquitectónico por niveles y multiplataforma. Es la respuesta de la Open Handset Alliance (un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio) a la necesidad de una arquitectura multiplataforma para un ecosistema heterogéneo de dispositivos portables como *tablets* y móviles. Actualmente forma parte de Google, siendo el sistema operativo móvil más extendido con millones de usuarios y desarrolladores. Actualmente la cuota de mercado de Android es aproximadamente del 79% en *smartphones* (Tabla 1).

La arquitectura de Android está basada en niveles, donde cada nivel se apoya en el nivel inferior que le proporciona las funcionalidades necesarias.

La arquitectura de Android [4, 5, 6] se divide en cinco componentes como muestra la Ilustración 3:

- Kernel: Utiliza un kernel basado en una versión modificada del kernel Linux 2.6 que es el que se encarga de la gestión de memoria, procesos, red y ficheros. Actúa como una capa de abstracción entre el hardware y el resto de la arquitectura, permitiendo así que Android se pueda utilizar en una amplia variedad de dispositivos móviles.

- Librerías nativas: Es un conjunto de librerías escritas en C/C++ que pueden ser utilizadas por los desarrolladores a través del *framework*. Incluye librerías como System C library, WebKit y SQLite.

- Runtime de Android: Se compone de una máquina virtual especial de Java denominada Dalvik y ciertas librerías *core* de Java (se han eliminado algunas librerías que no son consideradas necesarias para hacerlo más liviano y mejorar su rendimiento y menor consumo de energía, por lo que no hay compatibilidad total con aplicaciones y librerías hechas para Java).

  La VM Dalvik corre código Java pero no es compatible con la máquina virtual de Oracle. Dalvik utiliza su propio *bytecode* aunque es posible realizar la conversión de archivos *class* de Java a archivos Dex de Dalvik a través de la herramienta *dx* del SDK de Android. Permite lanzar múltiples instancias de maquinas virtuales permitiendo así aislamiento y seguridad en las aplicaciones. Además, Dalvik está optimizada para funcionar en entornos con baja memoria y capacidad de procesamiento.

- Capa de *framework*: Está diseñado para simplificar la reutilización de componentes. Permite a las aplicaciones publicar sus servicios para que otras aplicaciones sean capaces de utilizarlos. Es el API que generalmente usan los desarrolladores y da acceso a la mayoría de funcionalidades básicas del teléfono:
  - Activity Manager: Controla el tiempo de vida de los procesos.
  - Telephony Manager: Controla las llamadas telefónicas y mensajes.
  - Content Provider: Controla la compartición de datos entre aplicaciones.
  - Location Manager: Controla el uso del GPS o de la antena de telefonía para servicios de localización.
  - Resource Manager: Maneja distintos tipos de recursos para las aplicaciones.
- Aplicaciones: Se posicionan encima del resto de capas, teniendo sólo acceso al *framework*. Cualquiera puede crear sus aplicaciones, pudiendo sustituir las aplicaciones que trae de fábrica.

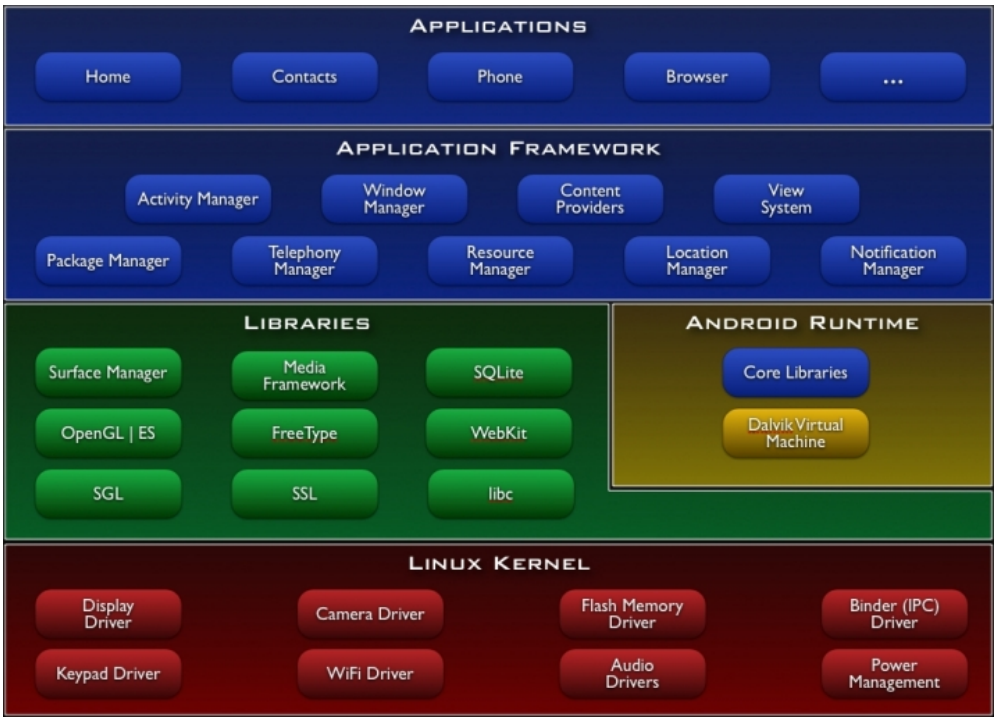


Ilustración 3: Arquitectura de Android

.

Como ya se ha comentado, el desarrollo para Android se hace en lenguaje Java utilizando el SDK (Software Development Kit) de Android (disponible para Windows, Linux y Mac). Normalmente se utiliza como IDE Eclipse con el *plugin* de Android instalado. Además, se puede utilizar código escrito en C y C++ a través del NDK (Native Development Kit).

2.1.2 iOS

Es el sistema operativo móvil de Apple Inc. [8] Originalmente fue diseñado para el iPhone, aunque ahora también se utiliza en el iPod, iPad y en Apple TV. Es un sistema operativo diseñado exclusivamente para ser utilizado en hardware de Apple. Actualmente la cuota de mercado de iOS es aproximadamente del 15,5% en smartphones (Tabla 1).

El SDK está dividido en: Cocoa Touch (*multi-touch*, eventos, localización, cámara), Media (OpenAL, Quartz, OpenGL ES), Core Services (red, SQLite, hilos) y OS X Kernel (TCP/IP, Sockets, sistema de ficheros y seguridad). El IDE de desarrollo es Xcode, el cual está solo disponible para ordenadores Mac y dispone de un emulador y un constructor de interfaces. Además, para poder desarrollar aplicaciones para iOS y publicarlas en el App Store es necesario pagar 99 dólares anuales [7].

El desarrollo de las aplicaciones se hace en Objective-C, el cual es un lenguaje orientado a objetos construido sobre C y que dispone de un recolector de basura. Las aplicaciones, al estar en Objective-C, son compiladas y ejecutadas directamente sobre el hardware, en vez de ejecutarse sobre una máquina virtual, por lo que tienen un mejor rendimiento. A partir de la versión 4, se permitió la multitarea (que antes sólo estaba reservada para aplicaciones por defecto del sistema).

2.1.3 Windows Phone

Es un sistema operativo móvil desarrollado por Microsoft [10]. Surgió como el sucesor de Windows Mobile, a pesar de lo cual es incompatible con éste. La primera versión fue Windows Phone 7, lanzada en Octubre del 2010. La versión actual es Windows Phone 8.1. Actualmente la cuota de mercado de Windows Phone es aproximadamente del 3,6% en smartphones (Tabla 1).

Windows Phone permite multitarea, uso de procesadores multicore (hasta 64 cores) y pantallas de alta resolución.

Las aplicaciones son de "código gestionado" [9], que es un término acuñado por Microsoft para referirse a aplicaciones que son ejecutadas bajo la gestión de un máquina virtual (Common Language Runtime Virtual Machine), generalmente NET Framework (Microsoft) o Mono (Open Source) y pueden estar escritas en C# y Visual Basic .NET. Además, Windows Phone soporta librerías nativas escritas en C y C++, lo cual permite que se puedan portar más fácilmente programas escritos para Windows.

Para el desarrollo de las aplicaciones se utiliza el Windows Phone SDK el cual solo se puede instalar en Windows. El IDE sobre el que se realiza el desarrollo es Visual Studio y dispone de un constructor de interfaces así como un emulador (que requiere la tecnología Hyper-V).

2.1.4 Otros sistemas operativos móviles

A parte de los ya enunciados, existen otros sistemas operativos móviles con menor presencia en el mercado, entre los que se encuentran:

- BlackBerry: en el cual las aplicaciones pueden ser de navegador, aplicaciones Java o un híbrido entre navegador y Java.
- MeeGo: un sistema operativo basado en Linux, lanzado en septiembre de 2011 como resultado de la colaboración entre Intel y Nokia.
- Firefox OS: sistema operativo basado en Linux, recientemente desarrollado por la fundación Mozilla, cuyas aplicaciones son desarrolladas en HTML, CSS y Javascript [11].

2.2 Desarrollo móvil multiplataforma

En el anterior apartado se ponía de manifiesto la gran cantidad de diferentes sistemas operativos, lenguajes y paradigmas con los que hacer las aplicaciones. Actualmente, el principal problema de los desarrolladores de aplicaciones móviles es conseguir un equilibrio entre la penetración de la aplicación en el mercado y el coste de desarrollo de la aplicación (en términos monetarios y también de "*time to market*"). Si los desarrolladores quieren dirigirse a la mayoría de los usuarios de dispositivos móviles deberán hacer aplicaciones independientes para cada una de las plataformas existentes, o bien deberán intentar seguir algún tipo de estrategia multiplataforma [12], siendo las principales las que se describen en las siguientes subsecciones.

2.2.1 Aplicaciones web móviles

Esta solución consiste en desarrollar aplicaciones web y darle la apariencia nativa a través de plataformas o librerías Javascript y CSS. En verdad, este método en verdad no llega a ser multiplataforma, ya que no todos los navegadores son soportados por estas librerías y además no se tiene acceso a las capacidades del hardware de los dispositivos móviles. Las versiones más recientes de los navegadores proporcionan información como la orientación de la pantalla y geolocalización, pero esta información sigue siendo muy limitada respecto a la capacidad que podría tener una aplicación nativa.

2.2.2 Aplicaciones híbridas

Las aplicaciones híbridas surgen para solucionar el problema anteriormente mencionado. Un claro ejemplo de frameworks para construir este tipo de aplicaciones es PhoneGap. Básicamente, estos frameworks generan una aplicación nativa que consiste únicamente en un navegador web. Se le proporciona al desarrollador una API Javascript mediante la que puede leer información de los sensores del dispositivo y ejecutar acciones sobre el hardware en diferentes plataformas de manera nativa.

De esta manera, las aplicaciones híbridas retienen algunas de las ventajas de las aplicaciones nativas, como puede ser la posibilidad de ser distribuidas a través de la tienda de aplicaciones de cada uno de los sistemas operativos; sin embargo, lo que el usuario se está descargando en verdad es una aplicación web, no una nativa. Esto hace que los desarrolladores deban gastar mucho tiempo cuidando el *look & feel,* para dar la sensación al usuarios de que se trata de una aplicación nativa y así no decepcionarles con una mala experiencia de usuario.

2.2.3 Aplicaciones interpretadas

Las aplicaciones interpretadas utilizar elementos de interfaz nativos a la vez que mantienen una lógica de aplicación independiente de la plataforma. Un claro ejemplo de este tipo de framework para el desarrollo de este tipo de aplicaciones es Appceletaror Titanium. Este framework provee un conjunto de APIs Javascript (Node.js para ser más precisos) que permiten el acceso a elementos de interfaz, sensores y la ejecución de acciones de manera nativa.

Esta solución tiene muchas ventajas, pero también tiene desventajas, ya que los desarrolladores van a estar limitados por la funcionalidad que ofrezcan las APIs de la plataforma.

2.2.4 Aplicaciones generadas

La finalidad de este paradigma es la de generar aplicaciones totalmente nativas para cada una de las plataformas a partir de un único código base. Por ahora, esta tecnología está en fase de desarrollo y no hay ninguna plataforma que tenga una versión para producción.

2.3 Tipos de aplicaciones

En este apartado se realiza un análisis de las ventajas y desventajas de las aplicaciones nativas, las aplicaciones web y las aplicaciones híbridas. [16, 17]

Las aplicaciones nativas son aquellas que han sido desarrolladas específicamente para una plataforma en concreto utilizando los lenguajes de esa plataforma. Las aplicaciones web y las híbridas ya han sido descritas previamente.

2.3.1 Aplicaciones nativas

Una de las principales ventajas de las aplicaciones nativas es el mejor rendimiento que tienen respecto al los otros dos tipos de aplicaciones ya que son capaces de acceder al hardware del móvil para obtener una mayor velocidad. Además, permiten utilizarlas sin conexión a internet ya que pueden tener guardados todos los recursos necesarios en el móvil. Por otra parte, al poder ser incluidas en los App Stores de las diferentes plataformas le dan a la aplicación una mayor visibilidad para así poder obtener un mayor número de usuarios.

Sin embargo, este tipo de aplicaciones supone un mayor esfuerzo de desarrollo y mantenimiento debido a tener que desarrollar aplicaciones para cada una de las plataformas (aunque como ya se ha visto en el apartado anterior, hay frameworks para facilitar esta situación). Además, las actualizaciones tienen que ser descargadas por los usuarios, por lo que habrá muchas versiones de la aplicación siendo utilizadas con la consecuente dificultad que puede suponer el manejar varias versiones en el lado del servidor.

2.3.2 Aplicaciones web

Las ventajas de este tipo de aplicaciones es que son bastante fáciles de desarrollar ya que solo hay que adaptar la página web para que funcione para pantallas más pequeñas.

Además, HTML, Javascript y CSS son tecnología ya maduras y hay una gran cantidad de frameworks para facilitar el desarrollo. Además, todas las plataformas móviles tienen navegadores web capaces de visualizar aplicaciones web.

Sin embargo, estas aplicaciones no disponen del gran rendimiento que tienen las aplicaciones nativas (como se puede ver en el *benchmark* realizado por Martin Dobrev [18]), no funcionan sin conexión y además no tienen acceso a los dispositivos del móvil (aunque los navegadores están empezando a crear APIs para dar su funcionalidad, como por ejemplo el API de geolocalización, pero no ofrecen el mismo rendimiento que si fuese acceso desde una aplicación nativa).

2.3.3 Aplicaciones híbridas

Estas aplicaciones están a medio camino entre las dos anteriores. Por eso comparten ventajas ventajas de ellas, como por ejemplo la facilidad de desarrollo (ya que la mayor parte es desarrollo igual que el de una web), tienen acceso a las funcionalidades nativas de la plataforma, en gran medida son multiplataforma (siempre y cuando el framework con el que estés desarrollando ofrezca APIs para todas las plataformas a las que se quiera dirigir la aplicación) y además se pueden distribuir a través de App Stores.

Sin embargo, no todo son ventajas, ya que una vez más, su rendimiento no es comparable con el de una aplicación nativa, ya que en verdad son aplicaciones web que se encuentran embebidas dentro de una aplicación nativa que consiste solo en un navegador web con características adicionales.

2.4 Appcelerator Titanium

Titanium es un framework de código libre que permite la creación de aplicaciones móviles para diferentes plataformas como Android, iOS, Windows Phone, BlackBerry y web a partir de un único código escrito en Javascript. El elemento principal de Titanium es el SDK de Titanium y su IDE, Titanium Studio.

El módulo que se quiere desarrollar será un módulo de este framework. Titanium tiene una gran variedad de módulos desarrollados tanto por Appcelerator como por la comunidad de desarrolladores. En concreto, dispone de un módulo de mapa básico. A partir de este módulo se realizará este trabaja, añadiendo toda la funcionalidad extra que tiene como objetivo este TFG.

2.4.1 Arquitectura

La ejecución de Javascript se realiza a través del motor V8 de Google, el cual ofrece un gran rendimiento. Titanium funciona a través de módulos. Cada módulo tiene una funcionalidad que está programada en lenguaje nativo para cada una de las plataformas para las que tiene soporte ese módulo. Esta funcionalidad es ofrecida a través de una interfaz para poder ser usada desde Javascript por los desarrolladores de aplicaciones. Los módulos deben tener Proxys, que son los encargados de realizar la conversión de las llamadas a métodos y los parámetros con los que fueron realizadas al lenguaje nativo. Esta traducción se hace apoyándose en la API que ofrece Titanium.

Los desarrolladores de aplicaciones sólo tienen que programar su código en Javascript,

utilizando los módulos desarrollados por otras personas y la API proporcionada por Titanium para el control de la interfaz gráfica, eventos, etc, sin preocuparse de los diferentes lenguajes específicos de las plataformas objetivo [21].

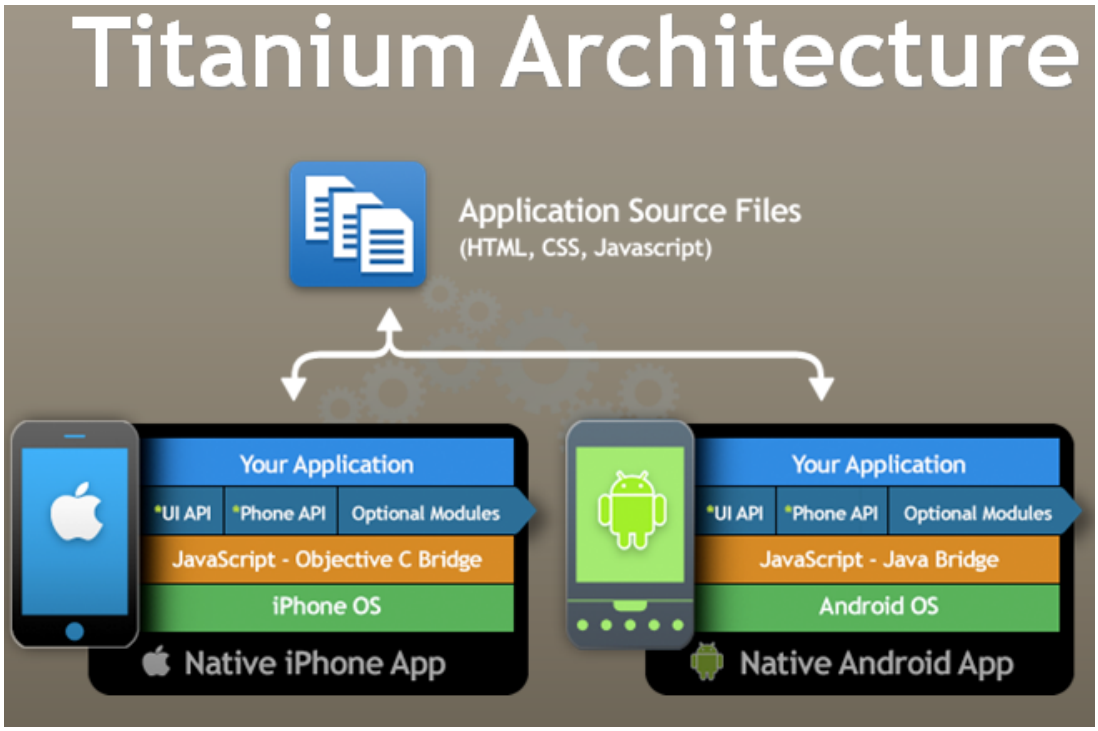


Ilustración 4: Arquitectura de Titanium

A la hora de generar la aplicación, el propio Titanium se encarga de generar una aplicación para cada una de las plataformas que tiene el desarrollador como objetivo. Para ello, realiza en cierto modo una elección en tiempo de compilación del código nativo, para la plataforma en la que se está compilando, de cada uno de los módulos que se utilizan desde la aplicación en Javascript.

De esta manera, se obtienen aplicaciones con un rendimiento muy similar al de las aplicaciones nativas, ya que los módulos están escritos en nativo, lo único que no es nativo es la lógica de la aplicación. Sin embargo, algo que sí es destacable es el mayor tiempo de inicialización que tienen las aplicaciones generadas con Titanium respecto a aplicaciones totalmente nativas, ya que las aplicaciones deben cargar el intérprete de Javascript y todas las librerías necesarias.

2.4.2 Titanium Studio

Appcelerator Titanium ofrece para los desarrolladores un IDE especializado llamado Titanium Studio. Este IDE ofrece toda la funcionalidad necesaria para que los desarrolladores puedan programar sus aplicaciones y desplegarlas en los móviles, facilitando la selección de los módulos a añadir y la edición de los archivos de configuración a través de una interfaz gráfica. Además, permite desplegar las aplicaciones tanto en varios emuladores como en el propio dispositivo, apoyándose para

ello en los SDKs de las correspondientes plataformas.

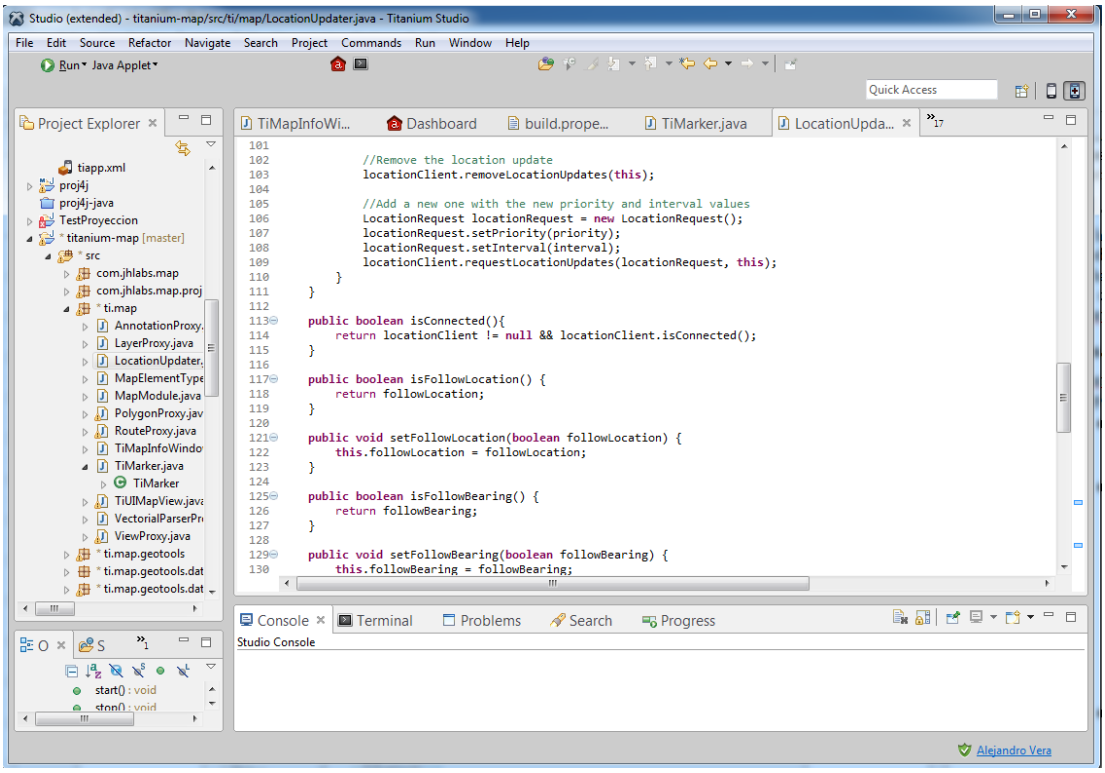


Ilustración 5: Titanium Studio

2.5 Wirecloud

Wirecloud es una plataforma de *mashup* que permite la construcción de aplicaciones rápidamente mediante la conexión de *widgets*, sin que se requieran conocimientos de programación.

2.5.1 Descripción del entorno Wirecloud

La Ilustración 6 describe la Arquitectura de Wirecloud y las relaciones entre sus principales elementos.

Los *widgets* son pequeñas aplicaciones desarrolladas utilizando HTML, CSS y Javascript que contienen puntos de entrada y de salida de datos. Mediante el *wiring* se puede conectar de manera visual la salida de datos de un *widget* a la entrada de datos de otro, creando así un *mashup* de elementos interconectados que colaboran para dar lugar a la aplicación. Estos datos se transfieren de un *widget* a otro en forma de eventos que pueden ser controlados mediante la API Javascript de Wirecloud.

Cada *widget* es desarrollado de forma independiente al resto de *widgets*, pero a su vez está ideado para ser utilizado junto con otros *widgets* con los que interactúa, para formar aplicaciones más complejas mediante *mashup*. Cuando se requiere de una transformación de datos o de la conexión con una determinada fuente de datos, se utilizan operadores y conexiones de *piping* para conectar estos operadores entre sí y con otros *widgets*. Así, mediante la combinación de diferentes *widgets* y operadores se

pueden llegar a crear aplicaciones web realmente potentes simplemente arrastrando *widgets* y operadores y conectándolos de manera gráfica.

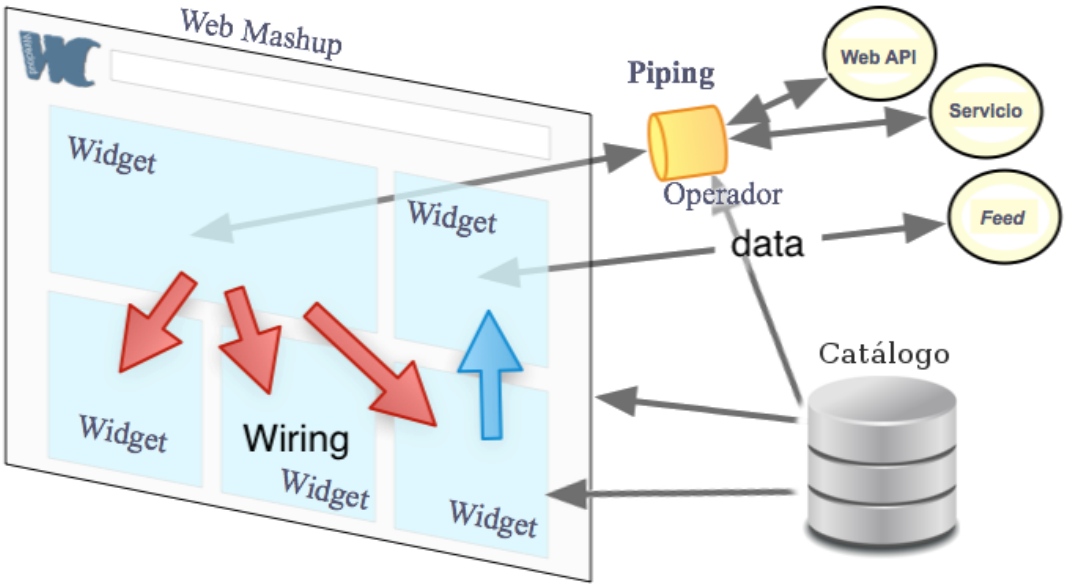


Ilustración 6: Arquitectura de Wirecloud

Los *widgets* y los operadores son creados por técnicos con conocimientos de programación, mientras que los *mashups* pueden ser creados por personas que carezcan de estos conocimientos. Esto permite incentivar la creación de aplicaciones ya que un mayor número de personas podrán realizar las aplicaciones que necesiten mediante la simple composición de *widgets* y *mashups* creados por otras personas y disponibles en el catálogo o *marketplace* de Wirecloud.

Un ejemplo de *mashup* podría ser un *mashup* con tres *widgets* para el control de los repartos de una empresa: un *widget* obtendría la información de los repartidores de la empresa y su posición actual. Este *widget* estaría conectado a un *widget* de visualización (un mapa) y pintaría la posición en tiempo real de cada uno de los repartidores. Después, podría haber un tercer *widget* conectado al mapa de tal manera que cuando se haga *click* en un repartidor en el mapa este tercer *widget* muestre la información sobre ese repartidor y el reparto que está realizando actualmente. Los datos publicados y consumidos por estos tres *widgets* y compartidos mediante *wiring* serían la información de los repartidores, incluida su localización, la información de los repartos y la información del mapa.

Otro ejemplo de mashup, esta vez relacionado con Smart City, se puede ver en la Ilustración 7.

2.5 WIRECLOUD

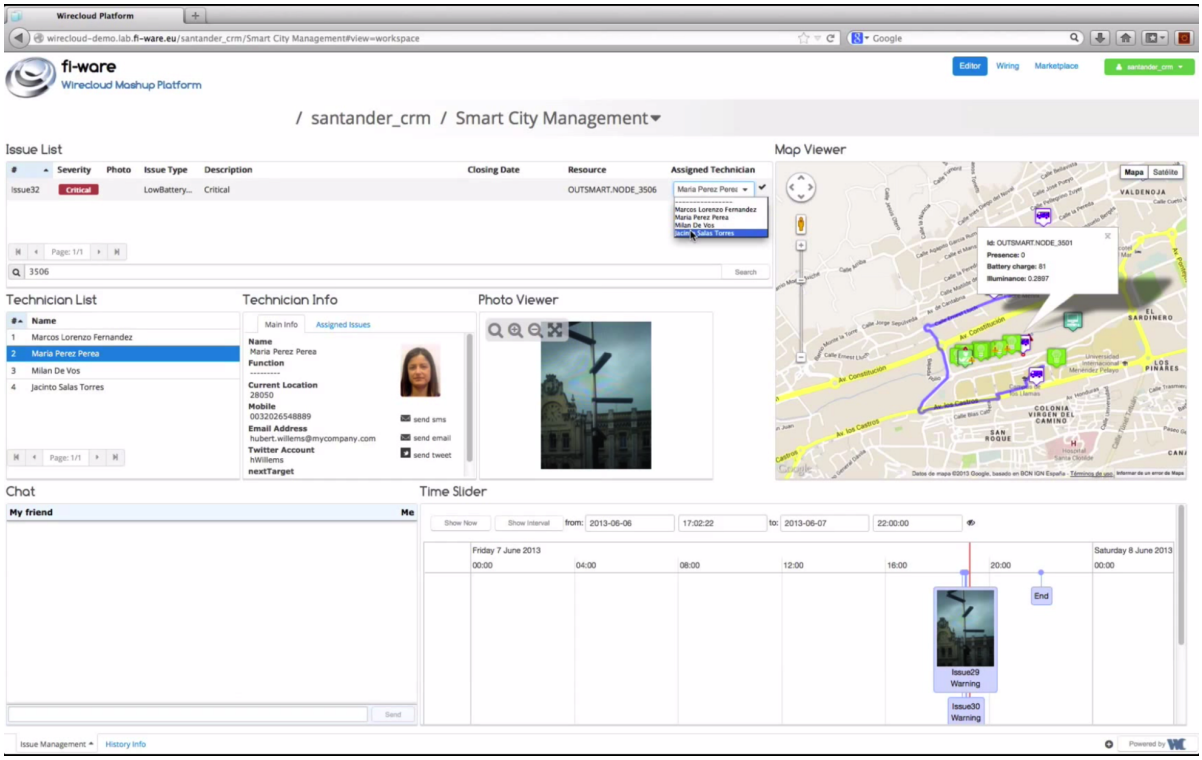


Ilustración 7: Mashup de Smart City Management

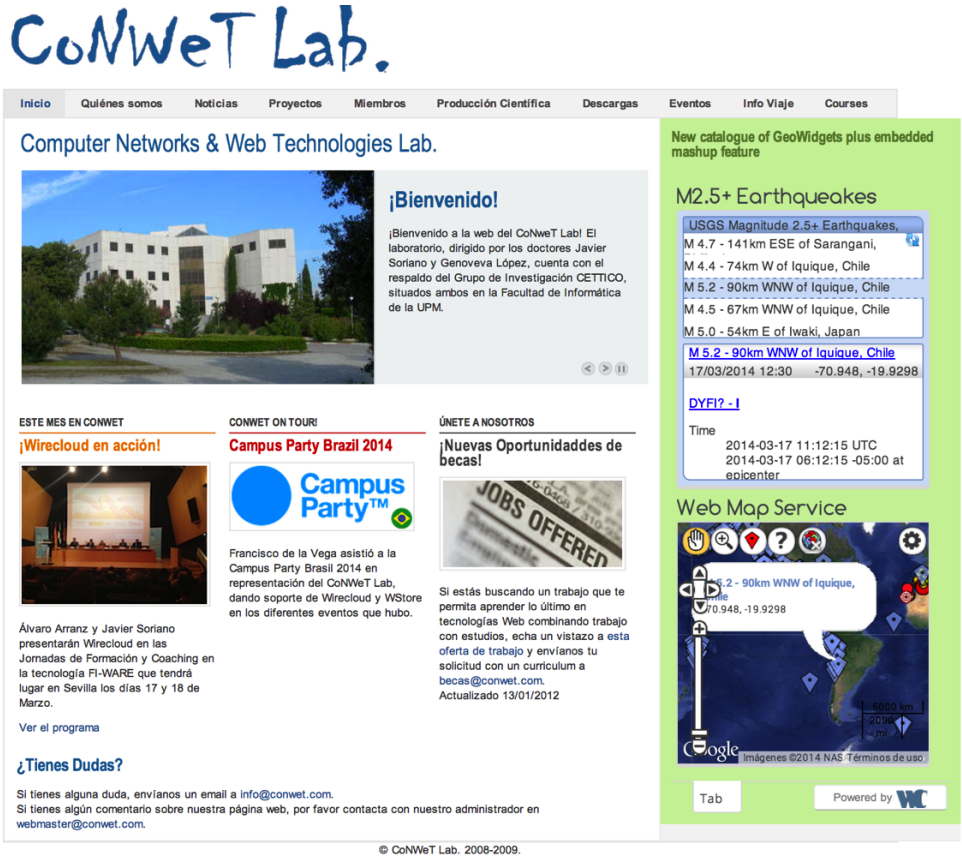


Ilustración 8: Ejemplo de integración con una web

Además de en el catálogo, con propósitos de compartición, los *mashups* pueden ser también publicados en páginas web mediante un código HTML generado por Wirecloud que puede ser añadido a la web en la que se quiera mostrar el *mashup* creado, o simplemente haciendo público el enlace (URL) al *mashup*.

2.5.2 Proceso de construcción de un mashup

En la Ilustración 9, se representa la secuencia de pasos para la construcción de un *mashup*.

2.5 WIRECLOUD

Ilustración 9: Construcción de un mashup en Wirecloud

2.5.3 Wirecloud4tablet

Wirecloud4Tablet es un proyecto enfocado al desarrollo de aplicaciones móviles, el cual permite construir aplicaciones composicionales con elementos nativos y elementos web. Esto es una parte fundamental del desarrollo ya que implica que el desarrollador pueda tener las mejores facetas de ambos contextos; es decir, rendimiento, *look and feel*, compatibilidad, *responsive design* y diversas funcionalidades que presentan tanto las aplicaciones puramente nativas como las aplicaciones puramente web.

La composición de estos elementos además puede tener un sistema de *wiring* o cableado al igual que su plataforma web de la cual hereda la primera parte del nombre (Wirecloud) y así comunicar o enviar diferentes tasas de datos entre cualquiera de esos elementos con poco coste computacional y programable por parte del desarrollador.

Actualmente Wirecloud4tablet solo soporta la visualización de *mashups*, teniendo que realizar toda la creación de estos desde Wirecloud. En la visualización de un *mashup* desde Wirecloud4tablet, cuando se detecta que uno de sus componentes es un *widget* que está disponible de forma nativa, se realiza la sustitución de manera automática, pudiendo así aprovechar todas las capacidad de la plataforma móvil sobre la que se está ejecutando.

El proyecto empezó planteando objetivos en este sentido ya que se utilizó la plataforma Wirecloud como el IDE para poder realizar esas conexiones y los elementos web que podemos visualizar posteriormente en la aplicación. Las futuras líneas del proyecto sugieren la creación de un IDE propio y al menos dejar la compatibilidad a la plataforma nombrada en un módulo o API a utilizar.

2.6 Trabajo relacionado

Se ha realizado una búsqueda en *stores* y en Google de aplicaciones móviles de mapas que tengan soporte para la funcionalidad objetivo de este trabajo (soporte para múltiples capas, visualización de elementos vectoriales, brújula, etc) con el objetivo de obtener ideas sobre la funcionalidad más importante y la forma de organizar todas esas funcionalidades dentro de una aplicación móvil.

A continuación, se indican algunas aplicaciones existentes, que comparten algunas de las características que se espera que tenga la aplicación resultante de este Trabajo de Fin de Grado. Sin embargo, ninguna de ellas cumple totalmente con los objetivos que se persiguen en este trabajo.

2.6.1 Sitna

Esta aplicación, desarrollada por el Gobierno de Navarra, es una aplicación web desarrollada para móviles utilizando tecnología HTML5, CSS3 y JavaScript, así como librerías de software libre, en concreto OpenLayers y jQuery. [13]

Debido a que es una aplicación web, la aplicación es multiplataforma y no requiere instalar nada; se puede acceder desde cualquier dispositivo con conexión a Internet, incluidos los navegadores actuales de equipos de escritorio, si bien el interfaz está optimizado para su utilización desde dispositivos táctiles.

Sin embargo, al no ser una aplicación nativa su rendimiento es mucho menor y al tener que manejar gran cantidad de datos (por ejemplo, una buena cantidad de POIs o muchas capas) responderá mucho más lentamente en un móvil que una aplicación nativa, que puede hacer uso del hardware para acelerarla.

Entre las características que ofrece se encuentran:

- Navegar por el mapa.
- Añadir capas de información (WMS).
- Consultar información de capas.
- Ver leyenda de capas.
- Seleccionar el mapa de fondo.
- Posicionamiento GPS.
- Buscador de lugares (WFS).
- Centrar en coordenadas.
- Guardar marcadores.

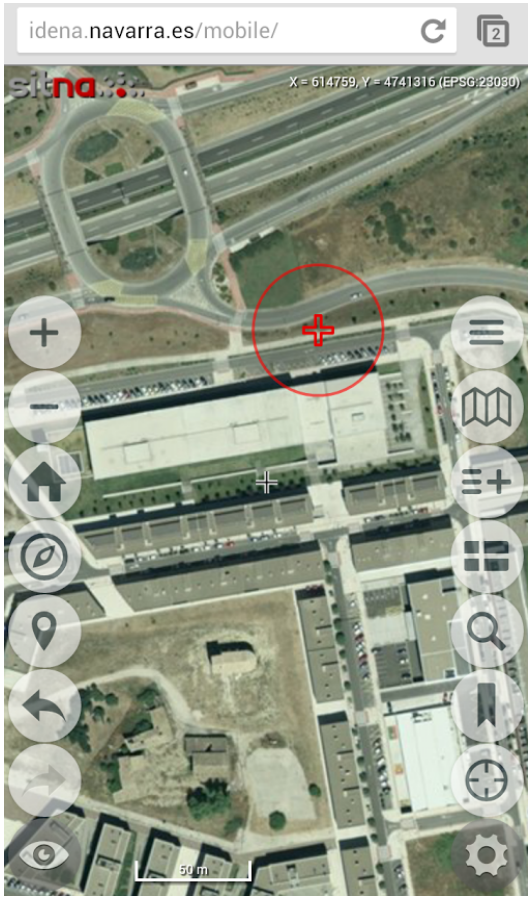


Ilustración 10: Aplicación Sitna en el móvil

Al poseer prácticamente todas las características deseadas para el mapa de este TFG, puede ser una aplicación de gran interés y a tener en cuenta durante el desarrollo del módulo y la aplicación del mapa.

2.6.2 GIS WMS Viewer

Se trata de una aplicación de pago desarrollada por MapGoGIS que se encuentra disponible en Google Play (el App Store oficial de Android). Se trata de una aplicación nativa y las características que ofrece son las siguientes:

- Navegar por el mapa.
- Añadir capas de información (WMS).
- Posicionamiento GPS.
- Medición de distancias y áreas.

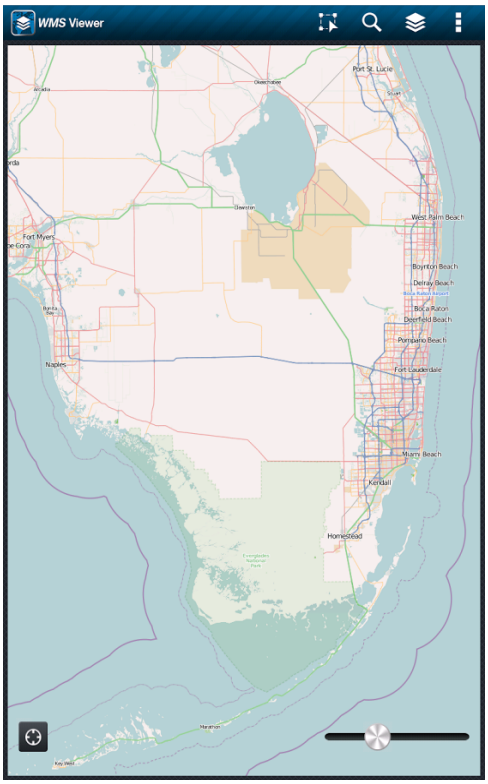


Ilustración 11: GIS WMS Viewer

2.6.3 Oruxmaps

Se trata de una aplicación nativa gratuita muy conocida entre las personas que hacen actividades al aire libre (bicicleta de montaña, senderismo...) que se encuentra disponible en Google Play. Entre las características que ofrece se encuentran:

- Navegar por el mapa.
- Añadir capas de información (WMS).
- Posicionamiento GPS.

- Grabación y visualización de tracks (rutas realizadas) en formato GPX, KML y KMZ.
- Diferentes métodos de navegación por rutas.

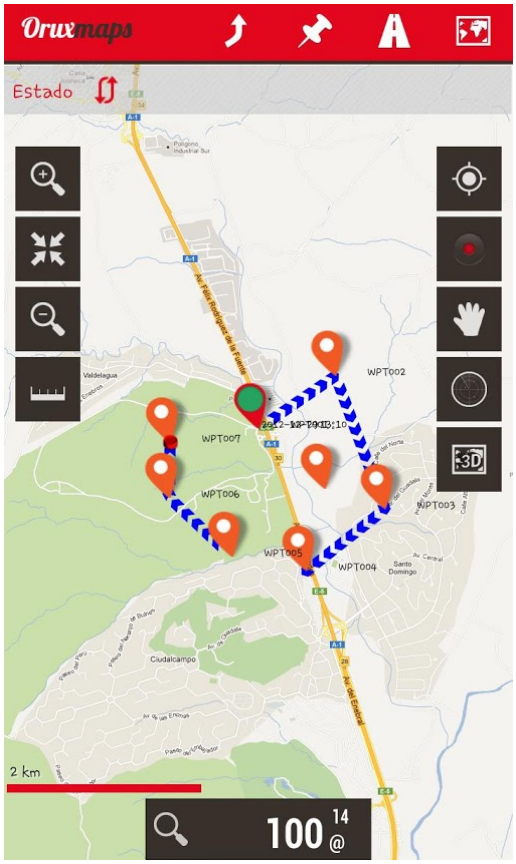


Ilustración 12: Oruxmaps

2.6.4 Catastro España

Se trata de una aplicación nativa gratuita que se encuentra disponible en Google Play. Esta aplicación permite obtener y visualizar la información proporcionada por el catastro.

- Navegar por el mapa.
- Añadir capas de información (WMS) del Catastro, IDEE y Cartociudad.
- Posicionamiento GPS.
- Consulta de información de las capas.

Ilustración 13: Catastro España

2.6.5 Citysurf Globe

Se trata de una aplicación nativa, gratuita y muy potente desarrollada por la empresa PiriReis Bilisim Teknolojileri que se encuentra disponible en Google Play. Hace uso de las APIs de OpenGL para obtener aceleración por hardware (GPU) en la navegación 3D. Algunas de sus características destacables son: [15]

- Navegación 2D y 3D por el mapa.
- Brújula digital (el mapa gira como si fuese una brújula, apuntando siempre al norte).
- Soporte para servicios GIS (WMS, WMTS, WMS-C, TMS, Digitalglobe XYZ).
- Posicionamiento GPS.
- Búsqueda de ruta entre dos puntos.
- Medición de distancias y áreas.
- Consulta de información de las capas.
- Guardar marcadores.
- Soporte para KML y KMZ
- Visualización de modelos 3D.

3 DESARROLLO

3.1 Desarrollo de un módulo para Appcelerator Titanium

Titanium está formado por colecciones de módulos que ofrecen una API de métodos, propiedades y constantes. Hay tres tipos diferentes de módulos [19]:

- Módulos integrados en el espacio de nombres de Titanium, como pueden ser los módulos de *Titanium.UI* y *Titanium.Geolocation*. Estos módulos están siempre disponibles para las aplicaciones de Titanium.

- Módulos empaquetados, que son extensiones a la API de Titanium y pueden ser importados en las aplicaciones Titanium utilizando el método *require.* Estos módulos son importados utilizando un identificador definido por el propio módulo (por ejemplo *Ti.Map*). Pueden contener tanto código nativo como Javascript y pueden ser obtenidos a través del *marketplace* de Appcelerator (o cualquier otro sitio donde se pueda acceder a su código fuente).

  Estos módulos son compilados y empaquetados para una plataforma en concreto (Android, iOS o Web). Durante la fase de empaquetado el código nativo es compilado y el código Javascript es ofuscado. De esta manera, los desarrolladores pueden distribuir su módulo y venderlo sin tener que revelar su código fuente.

- Módulos CommonJS, que pueden ser utilizado dentro de las aplicaciones para organizar y estructurar el código. Son añadidos a la carpeta *Resources* y no son manejados por Titanium.

El módulo que se se ha desarrollado en este TFG corresponde al segundo tipo; es decir, un módulo empaquetado. Este tipo módulos suele estar formado por los siguientes elementos:

- Proxy: es la parte fundamental del módulo ya que es la que expone la API Javascript. Como bien indica su nombre hace de *proxy*; es decir, hace de intermediaria entre Javascript y el código nativo. Están escritos en código nativo y extienden (en Android) de la clase KrollProxy, que ofrece un conjunto de métodos para realizar esa unión entre código nativo y Javascript. Gracias a esta clase, cuando se llama a un método en el objeto Javascript correspondiente a este *proxy*, se ejecuta el método nativo asociado a ese método Javascript. Lo mismo ocurre con las propiedades de los objectos. Cuando se intenta acceder o modificar una propiedad de un objeto Javascript, automáticamente se llama a un *getter* o *setter* en el objeto nativo. Además, se puede puede añadir un *listener* para los cambios en las propiedades para poder tratar los cambios en propiedades para las que no se han definido *setters* (es una segunda manera de hacerlo, que es la utilizada en el módulo *Ti.Map*).

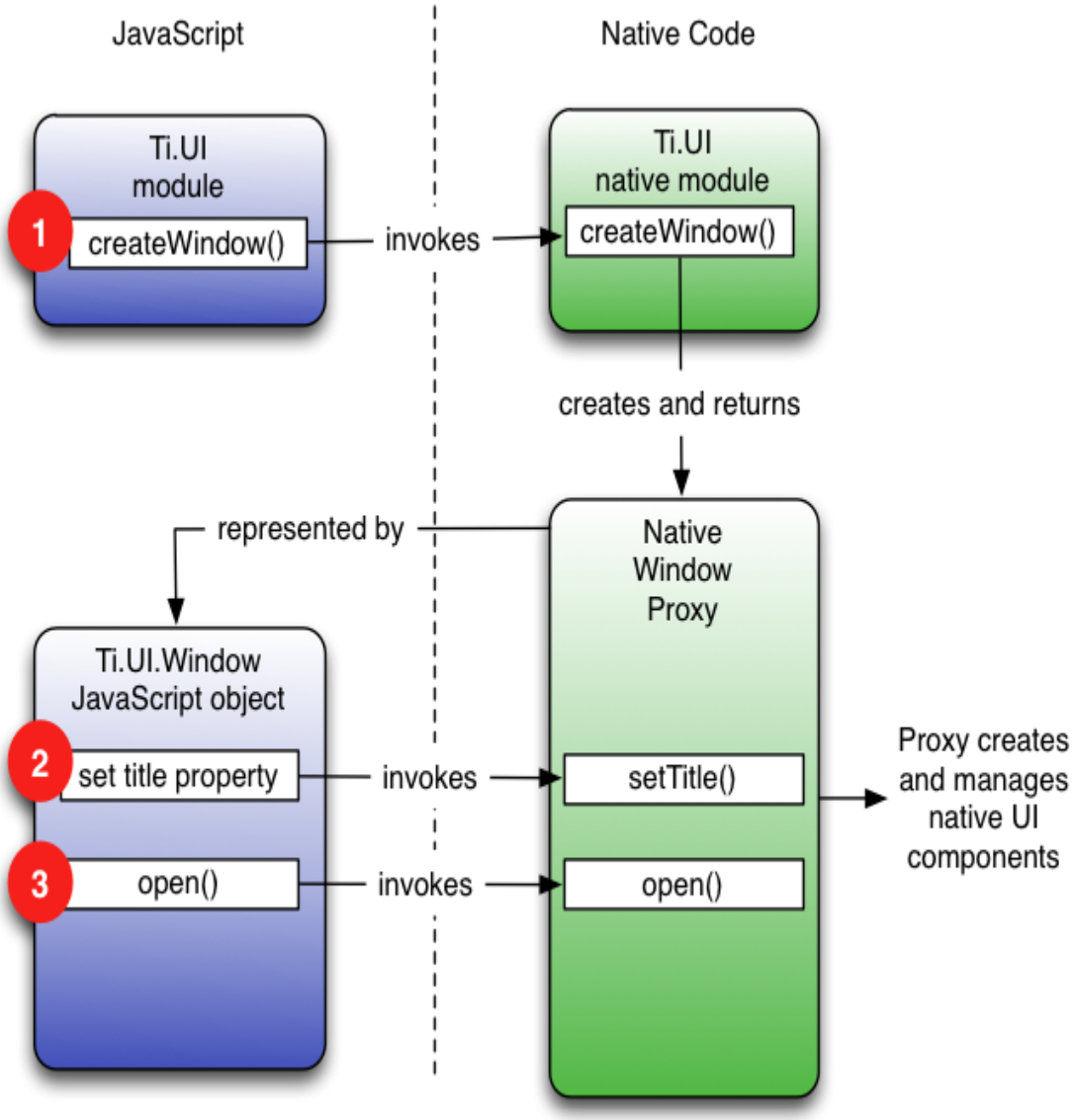


Ilustración 14: Comunicación entre Javascript y proxy nativo

- Module: sólo puede haber uno en el módulo que se está desarrollando. Es el que contiene el identificador utilizado para importar el módulo mediante la sentencia *require*. Se encarga de exponer los métodos para la creación desde Javascript de los objetos de los *proxies*. Por ejemplo, si tenemos un *proxy* llamado *MapProxy*, este objeto tendría un método llamado *createMap* que devolvería un objeto Javascript asociado a ese *proxy*. Todo esto puede ser realizado a través de sentencias para el preprocesador de Java.

- View y ViewProxy: son necesarios para la interfaz gráfica. Un *view proxy* es un *proxy* que extiende de *TiViewProxy* y tiene ciertas características adicionales a los *proxys* del sistema de interfaz gráfica de Titanium. Todo view proxy tiene asociado un objeto *view* que extiende de *TiUIView*. Este *view* puede ser creado y destruido cuando sea necesario dado que, por ejemplo, los controles de una ventana sólo deben estar instanciados cuando esa ventana esté visible.

Cuando se está desarrollando un módulo para Appcelerator Titanium hay que tener en cuenta que Javascript sólo ejecuta sobre un hilo; sin embargo, las plataformas nativas soportan múltiples hilos de ejecución. Además, el control de la interfaz gráfica solo se puede realizar desde el hilo principal por lo que las plataformas ofrecen métodos para poder realizar esta ejecución en el hilo principal. Todo esto supone una dificultad añadida a la hora de programar además de un código más largo y más complejo.

3.2 Desarrollo del módulo de mapa para Appcelerator

Esta es la fase que más tiempo ha llevado de las que se han realizado, teniendo en cuenta que era la primera vez que realizaba aplicaciones para Android y que utilizaba Appcelerator. Ha sido complicado aprender ya que el desarrollo de módulos está menos documentado que el propio uso de Appcelerator. Además, como se partió del módulo básico de mapas de Appcelerator, hubo que investigar cómo estaba organizado el código antes de empezar a añadir funcionalidad a ese módulo.

Por otra parte, se ha realizado un estudio de los diferentes estándares que se iban a utilizar, como pueden ser WMS (Web Map Service), WMTS (Web Map Tile Service), GeoJSON, SVG (Scalable Vector Graphics), WKT (Well Known Text), KML (Keyhole Markup Language)...

3.2.1 Preparación del entorno

El primer paso consistió en la preparación de todo el entorno de trabajo; es decir, la instalación de Titanium Studio, el SDK y NDK de Android así como otros elementos necesarios para la compilación e instalación del módulo.

3.2.1.1 Instalación de Titanium Studio

El instalador de Titanium Studio se puede descargar desde la página web de Appcelerator Titanium. Para ello es necesario disponer de un cuenta, aunque el registro es gratuito.

Una vez instalado, se debe configurar el centro de descargas y añadir un nuevo “Software Site”, que será el correspondiente a la versión de Eclipse del Titanium Studio instalado (en este caso Kepler).

Una vez añadido este nuevo “Software Site”, habrá que seleccionarlo como lugar de búsqueda de software e instalar “Eclipse Java Development Tools” que son necesarias para poder desarrollar un módulo para Android, puesto que su lenguaje nativo es Java.

3.2.1.2 SDK y NDK de Android

Para poder desarrollar un módulo para Android es necesario instalar el SDK de Android. En el caso de Titanium Studio, es muy sencillo hacerlo, ya que ofrece una interfaz gráfica (accesible desde el Dashboard) desde la que se puede seleccionar las versiones de Android que se quieren instalar (ver Ilustración 15). Además, en el caso del mapa, es necesario instalar el NDK el cual es un conjunto de herramientas que permiten embeber código máquina nativo compilado en lenguajes C o C++. Este NDK se puede obtener desde la página de *developpers* de Android.

Una vez hecho esto, hay que acceder al Android SDK Manager (un ejecutable disponible en la raíz de la instalación del Android SDK) y desde allí instalar los Google Play Services (necesarios para el mapa) y el Intel x86 Emulator Accelerator (que permite acelerar la ejecución de las imágenes de dispositivos móviles que utilicen procesadores Intel x86).

3.2.1.3 Driver para depuración por USB

Para poder depurar nuestras aplicaciones en un dispositivo Android a través de USB es necesario instalar un driver. Este driver puede ser el Google USB Driver, disponible desde el Android SDK Manager, que será necesario si el dispositivo móvil es uno de los Android Developer Phones (aquellos comprados desde Google Play). En caso contrario, habrá que instalar un driver OEM (Original Equipment Manufacturers) distribuido por el fabricante del dispositivo. Se puede obtener una lista de los sitios de descarga de estos drivers desde la web de Android (http://developer.android.com/tools/extras/oem-usb.html#Drivers).

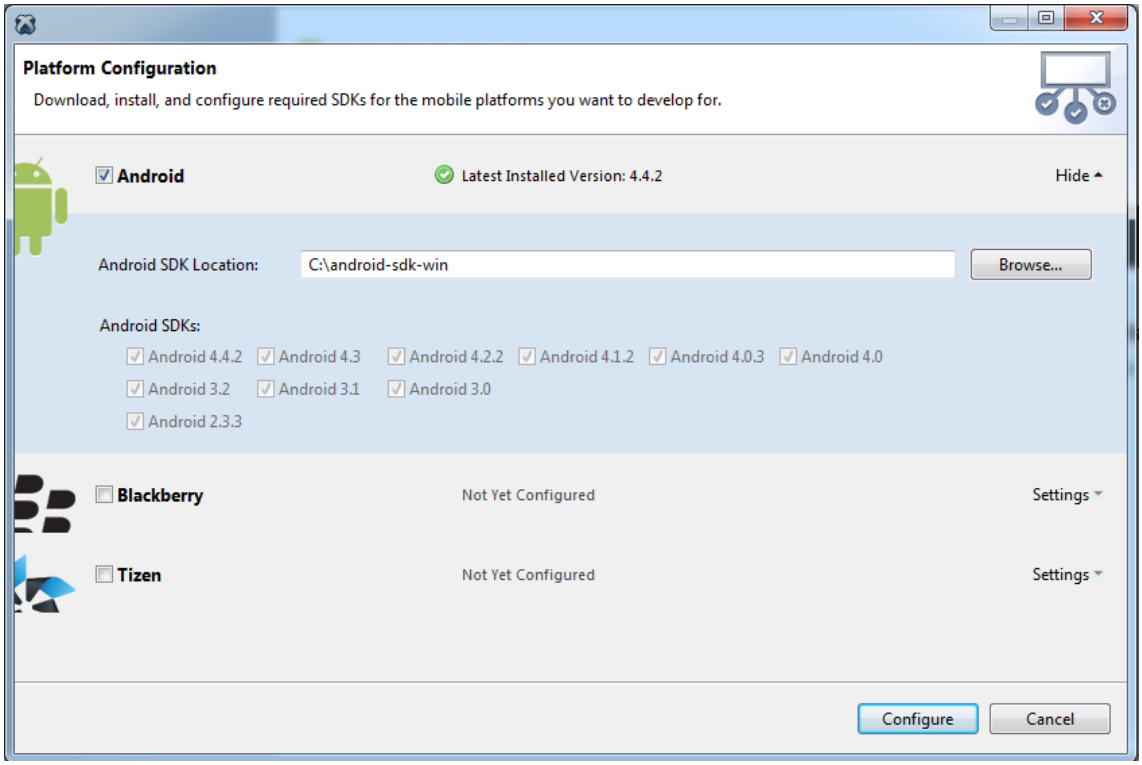


Ilustración 15: Selección de plataformas a instalar

3.2.1.4 Herramientas de compilación: Python y Gperf

El script de compilación del SDK de Android de Titanium necesita a su vez que esté instalado Python 2.7 (muy importante que sea esta versión en concreto, ya que sino no funcionaría debido al renombrado de ciertos métodos en la versión 3.0). Además, es necesaria la herramienta Gperf, la cual en principio solo existía para Linux, pero ha sido portada a Windows por el proyecto GnuWin32.

3.2.1.5 Descarga y configuración del código

Tras haber realizado toda la configuración anterior, hubo que descargar el código del módulo de mapa existente disponible en GitHub (https://github.com/appcelerator-modules/ti.map), en concreto la carpeta de android. Una vez descargada esta carpeta, hubo que importar el proyecto de esa carpeta, actualizar los *path* del *classpath* del proyecto por los del sistema en el que se estaba trabajando y crear un fichero *build.properties* (el cual contiene variables requeridas por el *script* de compilación del SDK de Android de Titanium) en la raíz del proyecto con el contenido que se muestra en la Ilustración 16:

```
titanium.platform=C:\\Users\\Alejandro\\AppData\\Roaming\\Titanium\\mobilesdk\\win32\\3.2.2.GA\\android
android.platform=C:\\android-sdk-win\\platforms\\android-10
google.apis=C:\\android-sdk-win\\add-ons\\addon-google_apis-google-10
android.ndk=C:\\android-ndk-r9c
python.exec=C:\\Python27\\python.exe
```

Ilustración 16: Fichero build.properties

3.2.1.6 Compilación, empaquetado y despliegue del módulo

La compilación y empaquetado del módulo se realizan a través de un script Ant del SDK de Android de Titanium. Para llevarlas a cabo se debe hacer click derecho sobre el archivo *build.xml* del módulo (que llama a su vez al script anteriormente mencionado) y seleccionar Run As → Ant Build… (Ilustración 18) y en la ventana que se abre a continuación elegir como *target "dist"* y pulsar en *Run* (Ilustración 19). El paquete del módulo será generado de esta manera en la carpeta "*dist*" del proyecto.

Para desplegar el módulo en una aplicación se puede hacer desde Help → Install Mobile Module. Se abrirá una ventana donde habrá que indicar el archivo JAR del módulo y el lugar donde desplegar el módulo: Titanium SDK (para que sea accesible desde todas las apps que estemos realizando) o Mobile App Project (para indicar la App en concreto en la que queremos instalar el módulo).

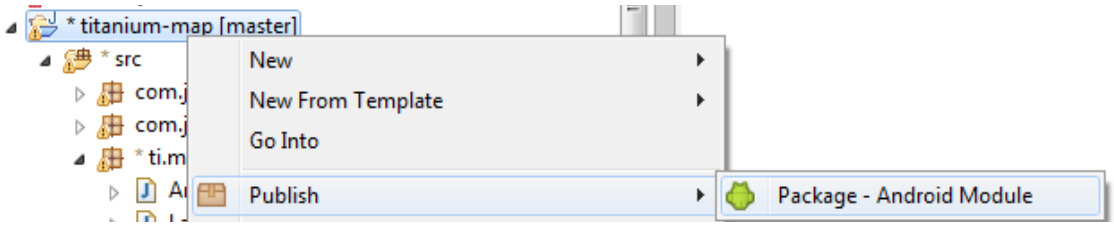


Ilustración 17: Publish de Titanium Studio

Otra manera más sencilla de realizar estos pasos es mediante la funcionalidad *Publish* de Titanium Studio a la cual se puede acceder haciendo *click* derecho sobre el proyecto del módulo (Ilustración 17) y posteriormente indicando dónde instalar el módulo (igual que en el caso de Install Mobile Module).

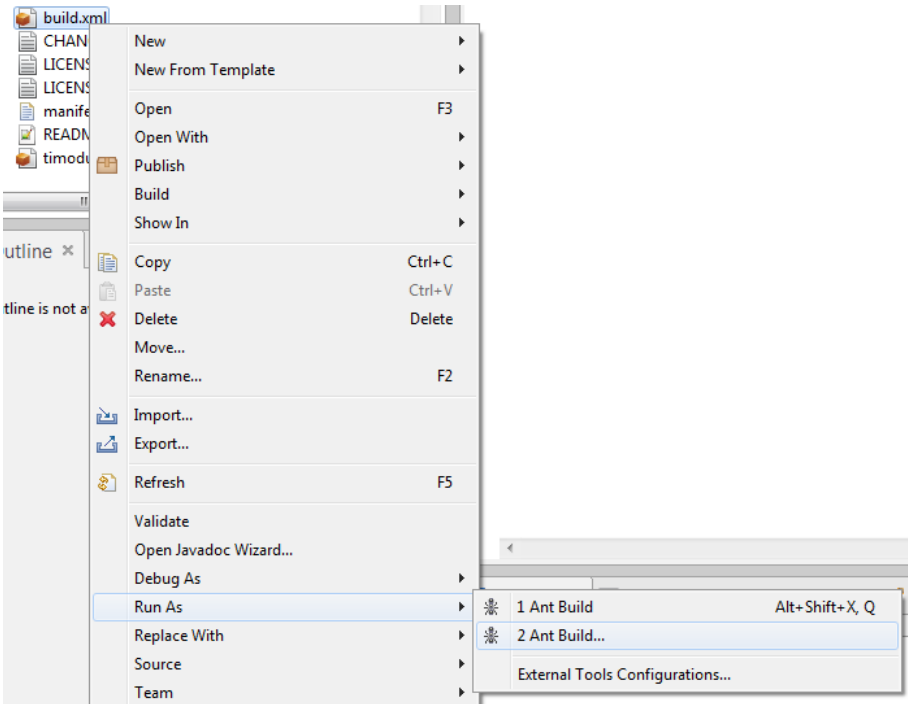


Ilustración 18: Compilación del módulo mediante Ant

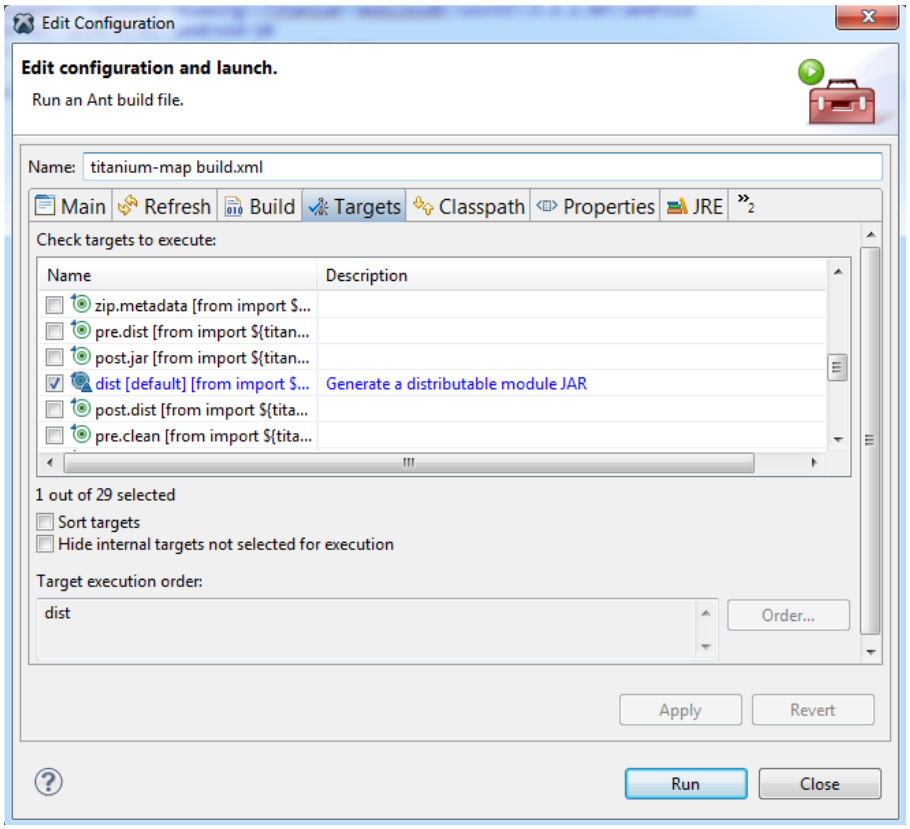


Ilustración 19: Elección del target de Ant

3.2.2 Estructura básica inicial del módulo

El módulo está formado por diferentes carpetas y ficheros, cada uno de los cuales cumple con una finalidad. A continuación se detallan las carpetas y ficheros que componen el módulo (Ilustración 20).

- Platform: se trata de una carpeta donde se deben introducir los ficheros que sean específicos de una arquitectura concreta.
- Libs: en esta carpeta se generan las librerías resultantes de las compilaciones que realiza Android NDK. Es una carpeta que no afecta al desarrollo, ya que es utilizada automáticamente por el *script* de compilación y empaquetado.
- Lib: en esta carpeta se deben añadir las librerías externas que utiliza el módulo.
- Hooks: es una carpeta que contiene unos archivos de instalación y desinstalación vacíos, solo por si el desarrollador quiere crearse sus propios scripts.
- Example: es una carpeta que contiene ficheros con ejemplos de uso del módulo.
- Documentation: como su nombre indica, en esta carpeta se encuentra la documentación del módulo.
- Dist: en esta carpeta se genera el archivo JAR del módulo resultante del proceso de compilación y empaquetado.
- Build: se trata de una carpeta de Titanium Studio para alvergar los archivos *.class* y otros resultados de la compilación. No es de importancia para el desarrollador ya que la compilación y distribución están automatizadas.
- Assets: en esta carpeta se deben introducir todos los ficheros que no sean de código fuente que vayan a ser utilizados por el módulo.
- Src: en esta carpeta es donde se encuentra todo el código fuente del módulo. Inicialmente contenía un único paquete *ti.map* con los siguientes ficheros:
  - MapModule.java: es el Module de este módulo (ver 3.1Desarrollo de un módulo para Appcelerator Titanium). Contiene una gran cantidad de constantes para el módulo así como el ID del este.
  - ViewProxy.java: es el *View Proxy* de *TiUIMapView*; es decir, es el *proxy* de la vista del mapa.
  - TiUIMapView.java: es la *View* del mapa; es decir, la parte visible del mapa con la cual puede interaccionar el usuario y en la que se visualizan las capas. Contiene el objeto GoogleMap con el que se controla el mapa de Google de Android.
  - AnnotationProxy.java: es un *proxy* para las anotaciones las cuales permiten marcar POIs (Points of Interest) gráficamente y proporcionan un *popup* con información cuando se hace "*click*" (se pulsa con el dedo) sobre ellas.

- RouteProxy.java: es un *proxy* para la creación de rutas; es decir, lineas formadas a partir de un conjunto de puntos que se pueden añadir al mapa.
- TiMarker.java: es una clase que simplemente realiza la asociación entre un *AnnotationProxy* y el *Marker* de Google Maps que le corresponde. Se utiliza internamente, no es visible desde fuera del módulo.
- TiMapInfoWindow.java: clase interna que representa el *popup* que se abre al seleccionar un *Annotation*.

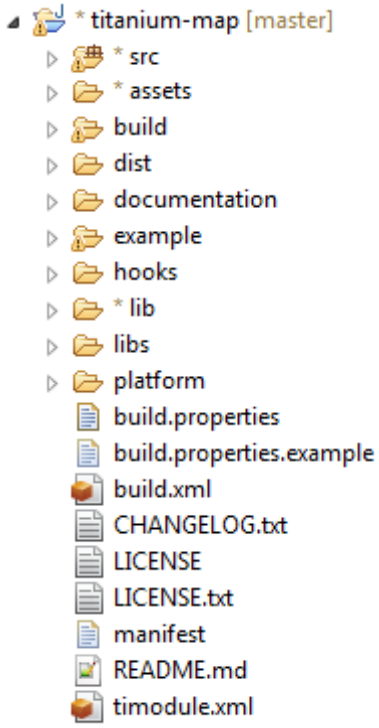


Ilustración 20: Estructura de archivos inicial del módulo

3.2.3 Soporte básico para WMS

La primera tarea por la que se empezó para la mejora del módulo fue el soporte básico para capas WMS.

El mapa de Google de Android no ofrece soporte para WMS; sin embargo, se puede realizar un "*hack*" para poder hacer que visualice las capas WMS. Este *hack* se basa en que Google Map permite añadir objetos *TileOverlay*, que son diferentes capas. Cada uno de estos objetos *TileOverlay* hace uso de una clase que implemente la interfaz *TileProvider* que es la encargada de proporcionar un *Tile* dadas unas coordenadas x,y, y z del mapa de Google.

Gracias a esto, la manera de añadir capas WMS al mapa de Google consistía en añadir un *TileOverlay* que utilizase un *UrlTileProvider*. Este *UrlTileProvider* se trata de una

clase abstracta del mapa de Google que tiene un método por implementar: *getTileUrl*. Este método recibe las coordenadas x,y,z solicitadas por el mapa y debe devolver una URL. Por lo tanto, se creó una clase *WMSTileProvider* que extendía de esta clase *UrlTileProvider* con la lógica necesaria para generar las urls correspondientes al servicio para las coordenadas dadas.

Además, para conseguir convertir una capa en capa base se le puede indicar al mapa de Google que no debe mostrar ninguna capa base a través del método *GoogleMap.setMapType* y la constante MAP_TYPE_NONE. De esta manera, nuestro *overlay* con *z-index* más bajo será el que se convierta en una capa base "virtual" (ya que realmente no es una capa base).

Posteriormente se refactorizó el código para dar soporte a las diferentes versiones de WMS y facilitar la adición futura de soporte para versiones venideras. Para ello se creó una super clase *WMSTileProvider* y clases que heredaban de ella para cada una de las versiones de WMS soportadas (1.1.1 y 1.3.0).

Además, para una mejor organización, todas estas clases se crearon dentro del paquete *ti.map.layer* (no existente inicialmente).

3.2.4 Conversión de proyecciones

Además, se añadió soporte para la conversión de proyecciones y así poder solicitar capas WMS en proyecciones diferentes a la que maneja internamente el mapa de Google. Esto llevó bastante tiempo, pues hubo que encontrar una librería de cambio de proyecciones que funcionase en Android. Inicialmente se intentó utilizar una librería para Java pero, por supuesto, esa librería hacía uso de librerías del core de Java que no se encuentran en el *core* de Dalvik. Además, la librería era tan grande que el proceso de compilación abortaba al generar el archivo DEX ya que el número de referencias a métodos superaba 2^{16}.

La solución que se ideó para soluciones este problema fue generar la aplicación principal sin incluir la librería, y por otro lado compilar la librería y generar múltiples archivos *dex* mediante la opción *multidex* de la herramienta de traducción de *class* a *dex* (*dx*). La aplicación principal debería cargar dinámicamente los archivos *dex* de la librería y llamar, mediante reflexión de Java, a los métodos de interés para la conversión de coordenadas entre proyecciones.

Las primeras pruebas funcionaron y se pudo realizar la llamada a diferentes métodos de la librería; sin embargo, había un método fundamental para la transformación de proyecciones que no se estaba cargando y no se podía acceder mediante la reflexión.

Tras un tiempo de pruebas, llegué a la conclusión de que, o bien *dx* (herramienta de paso de *class* a *dex*) estaba ignorando las *Inner Classes* de Java, o bien el cargador de archivos DEX de Android las estaba ignorando. Por este motivo tuve que abandonar este método, a pesar de todos los avances conseguidos y el tiempo invertido.

Decidí entonces buscar librerías de proyecciones ligeras para Java, que utilizasen pocas librerías de Java. Encontré así una librería llamada Proj4j, que se trataba de un proyecto que parecía algo abandonado y que además todavía no tenía una versión estable. La

librería necesitaba una librería del *core* de Java, por lo que hubo que obtener la librería en cuestión, añadirla al proyecto y para la generación de la aplicación Android hubo que añadir una opción al *dexer* para que ignorase el hecho de que se estaba incluyendo una librería considerada como *core* de Java (la cual por defecto no está permitido por si en el futuro se decide añadir esa librería en el *core* de Dalvik, ya que podría haber problemas de coherencia al haber dos clases con el mismo nombre). Tras esto, se realizaron pruebas de la librería en Android y fueron todas satisfactorias.

3.2.5 Mejora del Tile Provider

Durante el proceso de añadir soporte WMS, hubo que sustituir la clase proporcionada en la API de Google Maps que realiza las peticiones de url de *tiles* de las capas (*UrlTileProvider*), ya que esta clase no tenía ningún *timeout* sobre las conexiones realizadas. Por algún motivo, el servidor del IGN (Instituto Geográfico Nacional) con el que se realizaban las pruebas de capas WMS había veces en las que nunca respondía a las peticiones realizadas. Al ocurrir esto, después de unos minutos de uso del mapa, los threads del pool creado por el mapa de Google para realizar las peticiones HTTP acababan todos bloqueados esperando la respuesta del servidor.

Para solucionar esto, hubo que implementar una clase que implementase la interfaz correspondiente proporcionada en la API de Google (*TileProvider*) que se encargase las conexiones con el servidor (añadiéndole un *timeout*) y la obtención y tratamiento de los *tiles*. Esta clase se llamó *AdvancedTileProvider* y hubo que actualizar la clase abstracta *WMSTileProvider* (creada para dar soporte a WMS) para que extendiese de ella.

Posteriormente, se decidió actualizar el nombre de los archivos de WMS añadiéndoles "Advanced" al comienzo.

3.2.6 Soporte para elementos vectoriales

Después se procedió a añadir soporte para elementos vectoriales. Se analizaron los diferentes estándares y se decidió implementar WKT, KML y GeoJSON. Estos estándares utilizan polígonos, que no estaban soportadas en la versión de partida del módulo del mapa ya que éste sólo soportaba rutas (líneas de múltiples puntos). Hubo que añadir así soporte para polígonos (planos y también con agujeros) en el mapa. Para el parseado de WKT y GeoJSON se pudieron utilizar librerías que realizaban todo el parseado y así facilitaban el acceso a la información de los ficheros, pero para el caso de KML hubo que crear un parseador por medio de la clase *SAXParser* que está incluida dentro del *core* de Dalvik.

3.2.7 Brújula y seguimiento de posición

A continuación, se añadió soporte para fijar el mapa según la posición y la brújula; es decir, que el mapa siga los movimientos que el usuario realiza. Si el usuario gira por ejemplo el móvil 30º, el mapa girará 30º con él, actuando así como una brújula. Esto se pudo realizar a través del *LocationClient* de Android, la cual se trata de una clase que permite la suscripción a Google Location Services para recibir actualizaciones de localización. Para ello hay que indicar durante esa suscripción qué tipo de información se quiere recibir, con qué frecuencia y con qué precisión a través de un objeto *LocationRequest*.

Para facilitar el manejo de esta característica, se creó una clase llamada *LocationUpdater* que se encarga de la gestión de las actualizaciones de posición y orientación en el mapa permitiendo activar o desactivar estas actualizaciones en el mapa.

3.2.8 WMTS

Finalmente, se intentó añadir soporte a WMTS. WMTS tiene un funcionamiento similar al del mapa de Google, funcionando como una pirámide de celdas (Ilustración 21); es decir, para cada nivel de zoom, el mundo se divide en una maya de celdas de NxM. La diferencia es que en el mapa de Google las celdas son siempre cuadradas, mientras que en WMTS son rectangulares y además, el espacio del mundo que abarca el servicio WMTS puede ser diferente al del de Google. Por lo tanto, es necesario calcular qué *tiles* corresponden a un *tile* de Google, descargarlas, colocarlas en el orden correcto, recortar la zona que corresponde exactamente al *tile* y escalar el resultado del recorte.

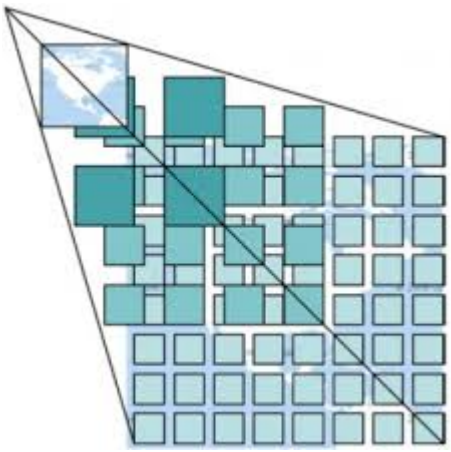


Ilustración 21: Disposición de los tiles en WMTS y Google Map

Para ello realicé un algoritmo que seguía estos pasos; sin embargo, este algoritmo suponía que la distribución del mundo en latitud y longitud en algunos es constante, y no es así, debido a que un grado cerca del polo abarca menos metros que uno cerca del ecuador. Tras perder dos semanas probando todo lo posible para adaptar WMTS al mapa de Google decidí pasar a realizar la aplicación ya que iba bastante retrasado respecto a la planificación inicial.

3.3 Desarrollo de la aplicación para Appcelerator Titanium

Una vez dado por finalizado el módulo, el siguiente paso consistía en la realización de la aplicación de Appceletator Titanium que hiciese uso de ese módulo. Se decidió tomar como referencia la aplicación web Sitna (ver apartado 2.6.1 Sitna).

Para el desarrollo de la aplicación se decidió dividirlo en dos partes: el desarrollo de la funcionalidad y el del estilo de la aplicación.

3.3.1 Desarrollo de la funcionalidad

En esta parte del desarrollo lo más importante era la implementación de las distintas

funcionalidades que debía tener la aplicación sin prestar atención a la apariencia de esta.

3.3.1.1 Organización básica de la interfaz

Se empezó realizando los controles de zoom para así poder aprender el posicionamiento de elementos dentro de la interfaz gráfica y la creación de iconos. En esta etapa se decidió añadir todas las vistas de la interfaz gráfica a la vista del mapa. Al añadirse vistas a otra vista, el último elemento añadido será el que se visualice encima de todos.

Ilustración 22: Vista general de la app

De esta manera, se podían emular ventanas creando vistas del mismo tamaño que la vista del mapa y añadiéndolas a esta. Esta forma de organización de la interfaz gráfica se eligió debido a que para la posterior integración con Wirecloud4Tablet era fundamental no utilizar ventanas, sólo vistas.

3.3.1.2 Ventana de gestión de capas

Después de haber aprendido a crear iconos y posicionar vistas, se procedió a añadir la funcionalidad para la creación y visualización de capas. Esta parte consistía en la creación de una vista que contuviese la lista de capas que se estaban visualizando en ese momento en el mapa y otra lista de las que podrían ser añadidas, además de una ventana que permitiese la creación de nuevas capas.

Se empezó por la creación de nuevas capas (Ilustración 23). El hecho de añadir capas

supone que deba haber algún tipo de persistencia para que cuando se cerrase la aplicación y se volviese a abrir siguiesen apareciendo las capas creadas en la lista de capas que se pueden añadir al mapa.

Por este motivo, se implementó un sistema de persistencia de información que consistía en un objeto global (*Ti.App.Data*). Al ser global se podría acceder desde cualquier parte de la aplicación. Al inicio de la aplicación, este objeto global se carga desde un archivo de la aplicación y cuando se va a cerrar la aplicación se guarda el objeto en ese archivo para no perder los cambios realizados en este objeto. Este objeto tenía como finalidad albergar la información de la capas creadas. Además, se creó otro objeto global esta vez temporal (*Ti.App.TempData*). La finalidad de este objeto es la de poder almacenar información temporal como pueden ser las capas que se están visualizando en el mapa.

Por otra parte, para crear una capa era necesario obtener la información del servicio en el que se encuentra la capa para poder mostrar la lista de capas y proyecciones de ese servicio. Para poder lograr esto era necesario tener soporte para acceso al *GetCapabilities* (operación mediante la que se obtiene toda la información de un servicio WMS) de los servicios, lo cual se decidió realizar mediante la librería Javascript OpenLayers; así como poder realizar peticiones HTTP.

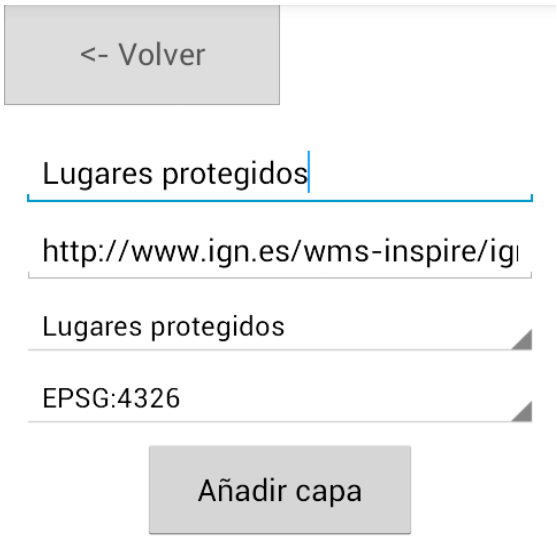


Ilustración 23: Creación de nueva capa

El motivo por el que se eligió OpenLayers es que se trata de la librería de referencia para consulta y visualización de información geográfica en Javascript. El hecho de estar disponible para lenguaje Javascript hacía suponer que iba a ser fácil utilizarla en el código de la aplicación. Sin embargo, no fue así debido a que internamente OpenLayers hace uso de ciertas características del DOM (Document Object Model) de los navegadores, que no están disponibles en las aplicaciones de Titanium (ya que carecen de DOM). Para solucionarlo, se barajaron dos opciones: realizar un *mock* del DOM o utilizar *WebViews*.

Finalmente se eligió utilizar *WebViews*, que son vistas que funcionan igual que una

pestaña de un navegador, por lo que sí disponen de un DOM. Esta *WebView* lo que debía hacer es contener un código HTML en la que se cargase la librería. Además, hubo que implementar un pequeño sistema de envío de eventos entre el código Javascript de la *WebView* y el código Javascript de la aplicación, ya que son totalmente independientes. De esta manera, si se quería ejecutar un método de OpenLayers desde la aplicación, se enviaba un evento a la *WebView* con el método a ejecutar y sus parámetros. La *WebView* ejecutaba ese método y lanzaba un evento en la aplicación con el resultado de la ejecución de ese método.

Por otra parte, el uso de las *WebView* supuso otro problema, ya que las *WebViews* siguen la Same Origin Policy, la cual se trata de una política de seguridad que impide a los *scripts* de una página web realizar peticiones HTTP a dominios diferentes del de la propia página. Por este motivo, la *WebView* era incapaz de obtener información de los servicios WMS, ya que el dominio de la *WebView* era la propia aplicación. Para solucionar esto, se utilizó la misma aproximación que para el caso de la ejecución de OpenLayers pero en sentido contrario. En este caso, cuando la *WebView* quería hacer una petición HTTP, enviaba un evento a la aplicación que realizaba la petición HTTP sin las restricciones del Same Origin Policy y después enviaba un evento de vuelta a la *WebView* con el resultado.

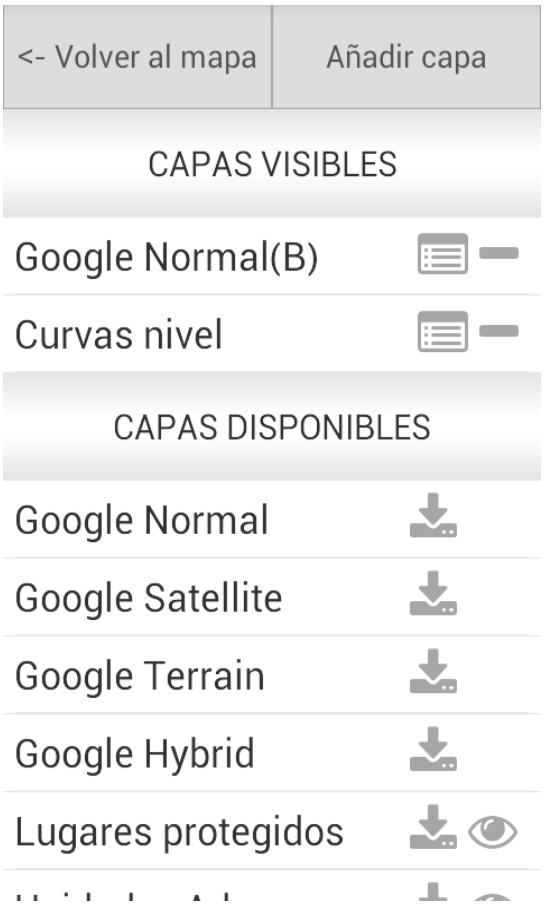


Ilustración 24: Organizador de capas

Una vez se tuvo la funcionalidad de creación de capas, se creó la ventana en la que se organizan. Esta ventana está formada por dos listas: una de capas visibles en el mapa y

otra de capas que se pueden añadir al mapa. Desde la lista de capas visibles se puede consultar la leyenda de la capa o eliminarla de entre las capas visibles. En la lista de capas que se pueden añadir se encuentra un icono para añadirla como capa base (sustituyendo la capa base actual) o para añadirla como *overlay*.

3.3.1.3 Botones de brújula y posicionamiento GPS

Se añadieron los botones de posicionamiento GPS y brújula. La brújula hace que el mapa se mueva igual que lo haría una brújula, mostrando el mapa con la misma orientación que el dispositivo.

El botón de posicionamiento GPS hace que el mapa se vaya moviendo con los cambios de posición del dispositivo, manteniendo el mapa centrado en la posición actual del dispositivo móvil.

La implementación de ambos botones fue sencilla debido a que la mayor parte del trabajo la realizaba el método ofrecido por el módulo de mapa.

3.3.1.4 Ventana de lugares favoritos

Esta ventana consta de una lista de lugares favoritos y un botón que lleva a una ventana de creación de lugares favoritos. Primero se realizó esta ventana de creación de lugares favoritos (Ilustración 25), la cual contenía únicamente un campo de texto para el nombre del lugar favorito. Cuando el usuario haga un *click* continuo sobre un punto del mapa, se abrirá un menú con la opción de añadir ese lugar como favorito. En tal caso, automáticamente se guarda la posición de ese punto y se añade el favorito a la lista de favoritos.

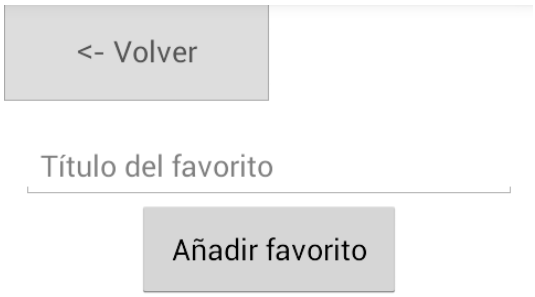


Ilustración 25: Creación de favorito

Desde la lista de favoritos (Ilustración 26), se puede eliminar el favorito o ir al favorito, lo cual hace que el mapa se mueva a las coordenadas y altura del favorito.

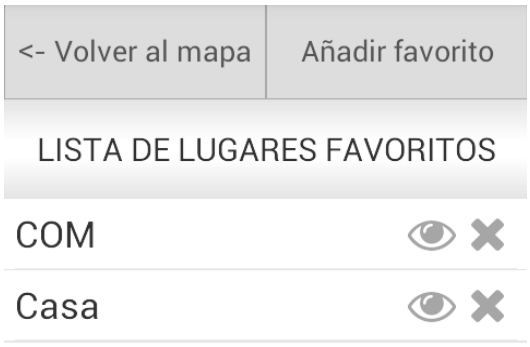


Ilustración 26: Lista de favoritos

3.3.1.5 Ventana de vistas favoritas

Esta ventana es muy similar a la de lugares favoritos, con la diferencia de que en este caso se guardan tanto la localización como la altura del mapa en el momento en que se crearon.

3.3.1.6 Botón de consulta de información de capas

Al activar este botón y mantener el dedo pulsando sobre un punto del mapa, se realiza una consulta de información de esa localización (*GetFeatureInfo*) de todas las capas *queryables*.

La implementación de esta característica fue algo compleja debido a que al servicio WMS, cuando se realiza la operación *GetFeatureInfo*, no se le pasan las coordenadas de las que se quiere obtener la información (que sería lo más sencillo para la implementación en este caso). En su lugar, lo que hay que pasarle el *bounding box* del *tile* (las coordenadas que engloban el *tile*), el tamaño del *tile* y el punto en píxeles dentro de ese *tile* del cual se quiere obtener la información. Eso complicaba la implementación ya que del mapa solo se pueden obtener las coordenadas sobre las que se han hecho *click*.

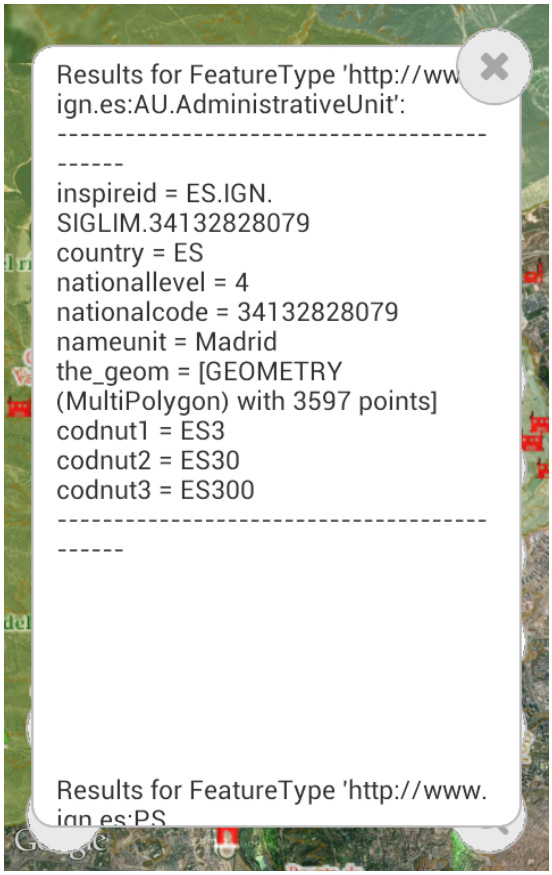


Ilustración 27: Consulta de información de capas

Para solucionar este problema lo que se hizo fue crear una petición "falsa" al servidor. No era necesario darle al servidor la información del *tile* sobre el que de verdad se está haciendo *click*, sino que se le puede dar de un *tile* equivalente. Por ello, la solución consistió en crear un *tile* imaginario de tamaño 1000x1000, considerar como *bounding box* de ese *tile* el *bounding box* que se estaba visualizando en el mapa en ese momento e interpolar el píxel a partir del *bounding box* y las coordenadas del punto sobre el que se hizo *click*. De esta manera se podía realizar una petición *GetFeatureInfo* que devolviese la misma información que si hubiésemos sabido la información del *tile* sobre el que realmente se estaba realizando la consulta.

Una vez hecho esto, ya sólo quedaba añadir a una ventana el texto plano devuelto por cada una de las solicitudes *GetFeatureInfo* (ya que se pueden estar visualizando varias capas *queryables* a la vez).

3.3.1.7 Botón de búsqueda de calles

Mediante este botón se puede visualizar una barra de búsqueda donde se puede realizar una búsqueda en tiempo real de calles (con forme se va escribiendo, se van mostrando nombres de calles que concuerdan con los datos introducidos hasta el momento). Una vez encontrada la calle, al hacer *click* en ella el mapa cambiará su localización a la de la calle seleccionada.

La implementación de esta búsqueda se realizó utilizando el servicio WFS de vial de Cartociudad (servicio del Instituto Geográfico Nacional).

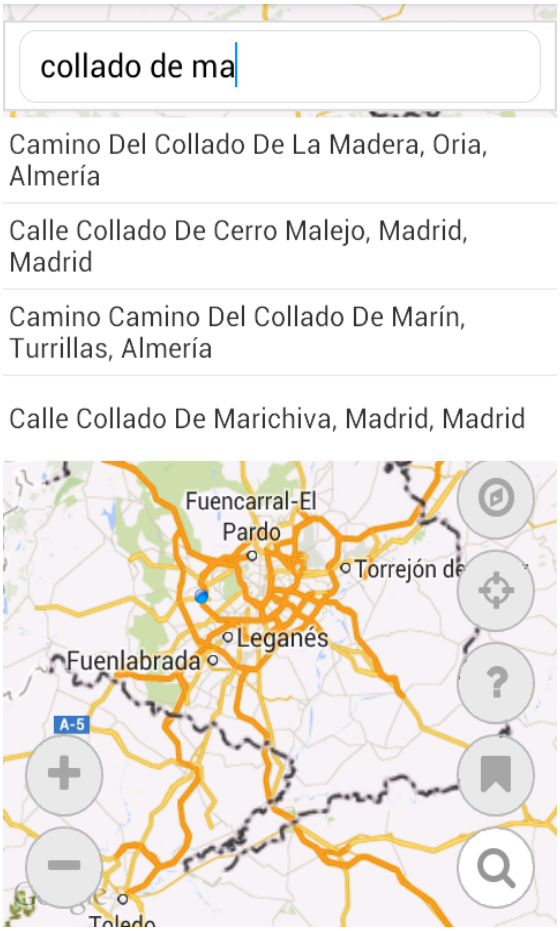


Ilustración 28: Búsqueda de calles

3.3.2 Desarrollo del estilo de la aplicación

Una vez desarrollada la funcionalidad de la aplicación, se procedió a mejorar la apariencia de esta. Appcelerator Titanium no permite separar la lógica de la aplicación de la presentación (como se consigue con CSS). Esto hace que el código sea poco legible, dado que al crear cualquier vista hay que indicar en su creación todo el estilo que tendrá pudiendo ocupar decenas de líneas simplemente la creación de una vista. Además, esto dificultaba la reutilización de código y los futuros cambios en el estilo de la aplicación.

Estaba claro que este era un problema que había que intentar mitigar. Para conseguirlo se creó un fichero de estilo común y, en caso de necesitarlo, uno por cada funcionalidad (Ilustración 29) si hay algún estilo que se quiera aplicar que no esté o deba estar en el archivo de estilos común. Estos ficheros contendrán un objeto con diferentes estilos (que a su vez son objetos). Por ejemplo, el archivo de estilo común contendrá el estilo *button* para los botones de la interfaz.

La idea de estos ficheros es poder cargarlos desde la lógica de la aplicación y utilizarlos

como parámetro en la creación de las vistas. Además, se creó una función *mergeObject(obj1, obj2)* (que añade a *obj1* el contenido de *obj2* sobreescribiendo los elementos de *obj1* que coincidan con los de *obj2*) para permitir la herencia entre objetos de manera que se pueda indicar el posicionamiento u otros atributos particulares de esa vista en la creación de la vista.

```
var theme = require('ui/style/common');

var pageView = Ti.UI.createView(Ti.App.mergeObject(theme.pageContainer, {
    layout: 'vertical'
}));

var goBackButton = Titanium.UI.createButton(Ti.App.mergeObject(theme.topMenuButton, {
    title: '<- Volver',
    top: 0,
    left: 0,
}));

var favoriteTitleTextField = Ti.UI.createTextField(Ti.App.mergeObject(theme.textInput, {
    top: 20,
    left: 10,
    right: 10,
    hintText: "Título del favorito"
}));

var addFavoriteButton = Titanium.UI.createButton(Ti.App.mergeObject(theme.button, {
    title: 'Añadir favorito',
    top: 0,
    left: '25%',
    right: '25%',
}));
```

Ilustración 29: Estilos en la aplicación

Gracias a esto, el estilo de la interfaz se puede modificar de manera mucho más simple y, mediante el cambio de un solo archivo, se pueden modificar las vistas de toda la interfaz.

3.4 Integración con Wirecloud4Tablet

3.4.1 Mashup a desarrollar

La última parte del Trabajo de Fin de Grado consistió en el desarrollo de un *mashup* (aplicación composicional formada por diferentes *widgets* u operadores) que demostrase la integración dentro de Wirecloud4Tablet (W4T) de la aplicación anteriormente desarrollada.

Wirecloud4Tablet es una *App* que se está desarrollando actualmente dentro del grupo de trabajo en el que he estado realizando mi TFG. Esta aplicación actúa como la versión móvil de la plataforma web Wirecloud, permitiendo la mezcla entre *widgets* web no nativos (de la plataforma Wirecloud) y *widgets* nativos para el dispositivo móvil que se esté utilizando.

Para demostrar estas capacidades haciendo uso de la aplicación de mapa desarrollada, se ha decidido realizar un *mashup* que esté formado por un *widget* de mapa nativo y un operador no nativo. El *mashup* es capaz de obtener información de paradas de autobús

mediante la consulta de una API REST de Santander y las envía al *widget* nativo de mapa donde son visualizadas. De está manera, se está demostrando la unión entre *widgets* nativos y no nativos.

3.4.2 Creación del mashup desde Wirecloud

El primer paso consistió en la creación del *mashup* en Wirecloud. Como ya se ha dicho anteriormente, Wirecloud4Tablet es Wirecloud para dispositivos móviles, sin embargo, es necesario crear primero el *mashup* desde la versión web.

Este *mashup* se debe crear utilizando *widgets* de Wirecloud (no nativos). Para ello, primero hay que subirlos al *marketplace* en caso de que no estén disponibles ya. En este caso, solo se ha desarrollado un *widget* (Street Search) que se encarga de realizar una búsqueda de calles en tiempo real a través del servicio WFS de Cartocidad del IGN (Instituto Geográfico Nacional). Los demás *widgets* utilizados (Map Viewer, que es el visualizador; y source-test, que envía una serie de puntos correspondientes a paradas de autobús de Santander) no han sido desarrollados como parte de este TFG, sino que ya existían previamente como parte del conjunto de *widgets* ofrecido por la plataforma Wirecloud y que ya están disponibles en su *marketplace* (Ilustración 30).

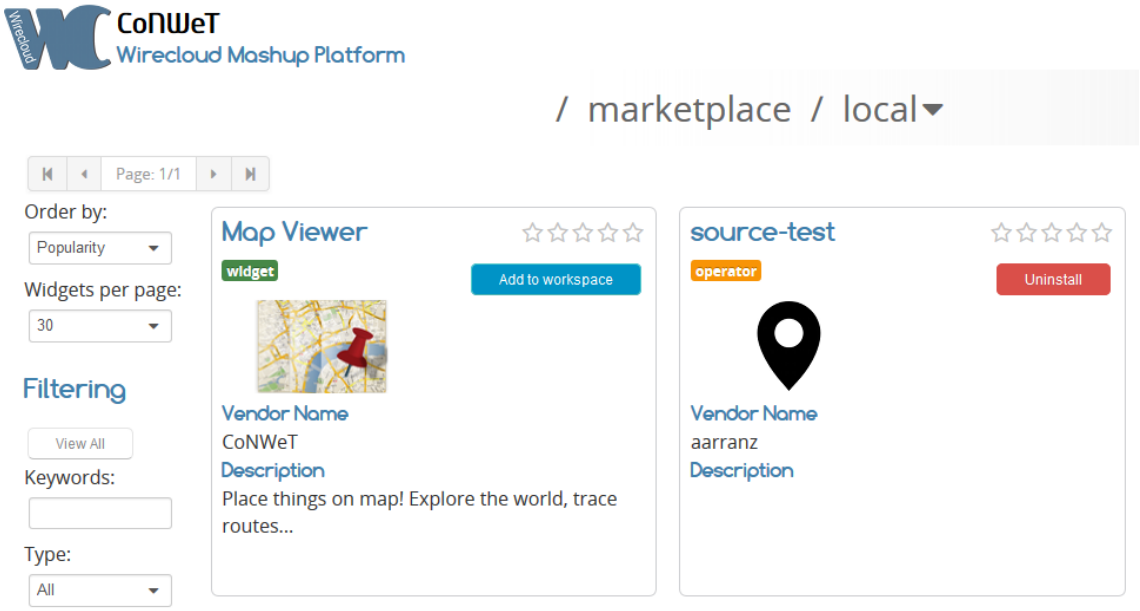


Ilustración 30: Marketplace con el widget y operador necesarios

Una vez disponibles en el *marketplace*, se añade el *widget* de mapa (*Map Viewer*) y el de búsqueda de calles (Street Search) al Workspace (el operador no se añade debido a que no es visible).

Después se realiza el *wiring* mediante el cual se conecta la salida del operador (que envía los POIs de las paradas) a la entrada correspondiente en el *widget* de mapa (Ilustración 31). Igualmente, se conecta el *widget* de búsqueda de calles al mapa.

Tras esto, el operador empezará a funcionar y enviará los POIs al mapa, pudiendo visualizar el resultado en el editor del Workspace (Ilustración 32). Además, el usuario

podrá buscar una calle y cuando haga *click* en ella el mapa automáticamente centrará su posición sobre la calle elegida y la marcará con un icono rojo (Ilustración 34).

3.4.3 Uso del mashup desde Wirecloud4Tablet

Wirecloud4Tablet se basa en la sustitución de *widgets* no nativos por *widgets* nativos que contiene dentro de si mismo. Para poder integrar la aplicación de mapa dentro de Wirecloud4Tablet hubo que desarrollar un *widget* nativo que se añadió dentro de Wirecloud4Tablet.

Este *widget* nativo consistía en el mismo código que la aplicación desarrollada para Appcelerator Titanium, con la diferencia de que toda la aplicación debía estar contenida dentro de una vista en vez de una ventana para que así Wirecloud4Tablet pueda añadirla a su propia ventana. Además, hubo que añadirle las mismas entradas y salidas de datos que el *widget* de mapa que se quería sustituir declarando manejadores para cada una de ellas. Tras esto, hubo que compilar de nuevo Wirecloud4Tablet con el nuevo *widget*.

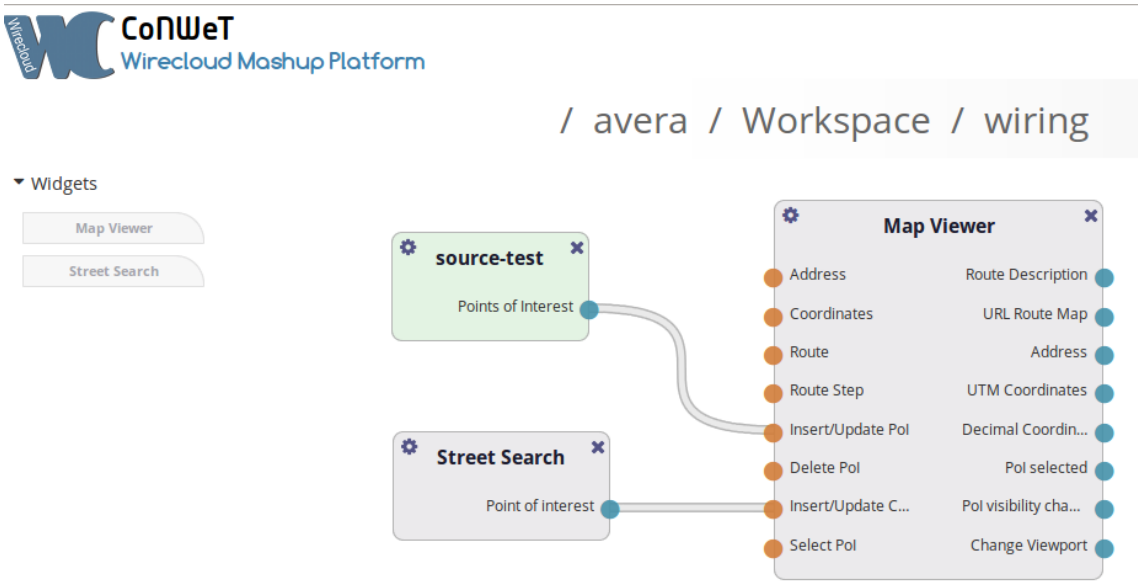


Ilustración 31: Wiring entre los widgets y el operador

Cuando Wirecloud4Tablet carga un *mashup* examina los *widgets* y operadores que lo componen para comprobar si tiene un sustituto nativo para ellos. En caso afirmativo, carga directamente el *widget* nativo que tiene guardado dentro de su propio código. En caso contrario, descarga el *widget* no nativo y lo introduce dentro de una WebView para que pueda ejecutarse como si de un navegador se tratase.

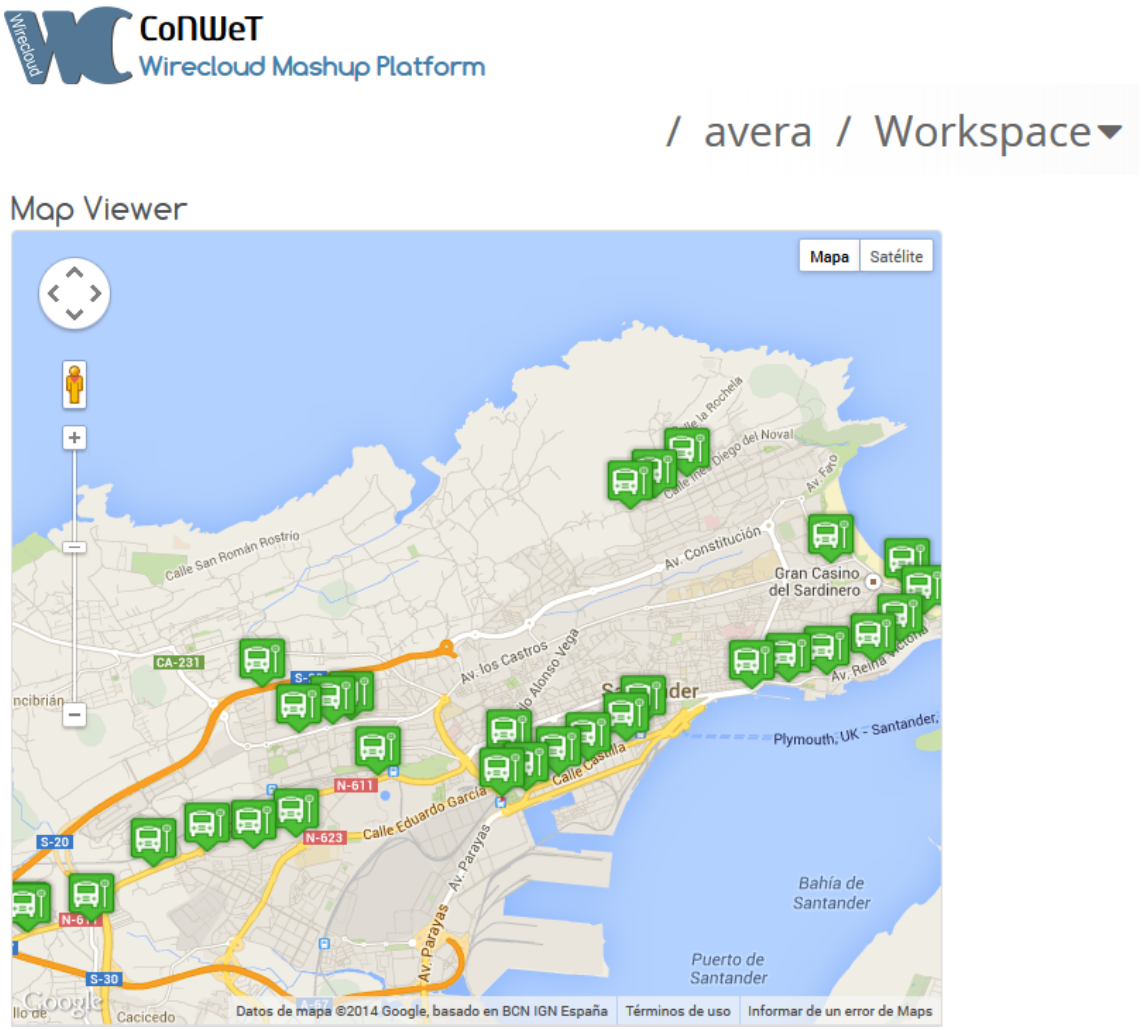


Ilustración 32: Vista del editor del Workspace de Wirecloud

Ilustración 33: Login de Wirecloud4Tablet

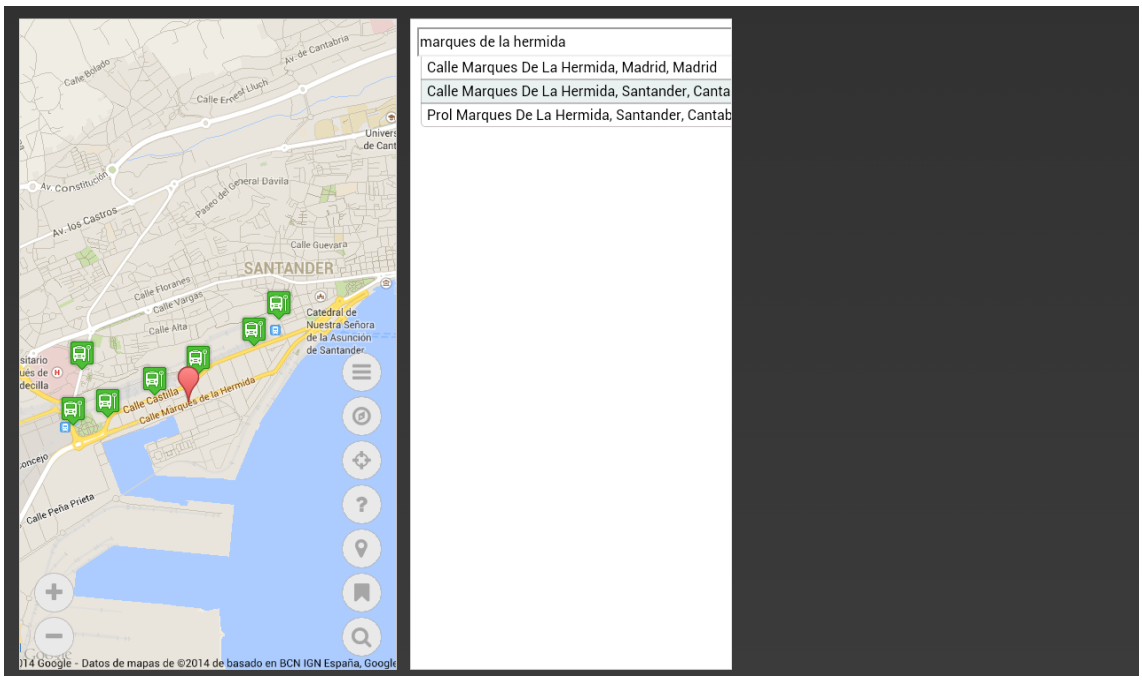


Ilustración 34: Vista del workspace de Wirecloud4Tablet

3.5 Ciclo de vida

3.5.1 Metodología

Para este proyecto se ha utilizado un ciclo de vida siguiendo metodologías ágiles, en concreto la metodología SCRUM.

Esta metodología consiste en dividir el desarrollo en diferentes etapas denominadas "*sprints*" al final de las cuales siempre tiene que haber un resultado útil. Cada uno de los *sprints* se planifica antes de empezarlo en el "*sprint planning*". Durante este *sprint planning* se crean unos casos llamados historias de usuario en las que se explican cosas que se quieran hacer y por qué se quieren hacer. Cada una de ellas tiene una dificultad y un prioridad en base a las cuales se decide cuales se van a llevar a cabo en ese sprint y cuales no da tiempo a hacer.

Al final del sprint se puede saber cuántas se han completado para poder mejorar el cálculo de la velocidad de trabajo. Además, al principio de cada sesión de trabajo se realiza un "*Daily Scrum*" en el que se hace un pequeño resumen de lo que se hizo el día anterior y se ve lo que se va a hacer ese día. En este caso, dado que no se ha trabajado en equipo no son necesarios los *Daily Scrum*.

3.5.2 Plan de pruebas

En todo proyecto software es fundamental definir un plan de pruebas para poder asegurar el correcto funcionamiento del software durante, y después, del periodo de desarrollo.

En este proyecto, debido a la escasez de tiempo y a la naturaleza del software desarrollado se ha preferido desarrollar un conjunto de pruebas manuales. Hay que tener en cuenta que todo el desarrollo se ha realizado en torno a un mapa, el cual es un software básicamente visual, lo cual dificulta la automatización de las pruebas. Para evitar gastar una gran cantidad de tiempo realizando la automatización de pruebas, se ha preferido optar por unas pruebas manuales. Sin embargo, en caso de que en el futuro se continúe con el desarrollo del módulo o de la aplicación, será necesario realizar una automatización de las pruebas para conseguir que el software sea mantenible.

Los tipos de pruebas que se han considerado para el software desarrollado en este TFG son las siguientes:

- Pruebas unitarias: se definieron pruebas para cada uno de los componentes de cada una de las vistas de la aplicación y cada una de los nuevos métodos del módulo. Para el caso de los componentes de la aplicación, las pruebas definían de manera textual el comportamiento y apariencia de cada uno de los componentes, por ejemplo, que esté centrado, que ocupe toda la pantalla, que al hacer *click* sobre él se marque, etc. Para el caso del módulo, se crearon diferentes entradas y se especificaron las acciones que se deberían observar en el mapa tras su ejecución.
- Pruebas de integración: se definieron pruebas de integración para cada una de las vistas de la aplicación, conteniendo las pruebas unitarias de los componentes de esa vista.
- Pruebas de sistema: se definió un conjunto de pruebas para comprobar el correcto funcionamiento de la aplicación en su totalidad. Básicamente consistían en historias de usuario, en las que el usuario debe realizar unas acciones de alto nivel (por ejemplo, añadir una capa del IGN, luego buscar su calle y consultar información sobre la capa añadida). De esta manera, se puede comprobar que la navegación entre distintas ventanas de la aplicación funcione correctamente.

Las pruebas de integración, y por consecuente, las pruebas unitarias, se llevaban a cabo para la vista afectada cada vez que se consideraba finalizado un cambio mayor en cada una de las vistas de la aplicación. Lo mismo se realizaba en el caso de los métodos del módulo, realizando las pruebas unitarias de los métodos del módulo afectados.

Las pruebas de sistema se realizaban tras acumular una cantidad aceptable de cambios en diferentes ventanas de la aplicación.

Por supuesto, lo óptimo habría sido realizar el conjunto de pruebas en su totalidad cada vez que se realizase un cambio, pero al tenerse que realizar manualmente esto resultaba inviable.

3.5.3 Gestión de configuración

Para poder llevar el control de versiones del software desarrollado, así como de servir de sistema de copias de seguridad, se han utilizado dos repositorios (para el módulo de mapa y la aplicación) creados en el GitLab del Conwet Lab. Los *commits* se realizaron en inglés para tener en cuenta la posible futura migración y publicación del repositorio en GitHub.

3.5.4 Gestión de incidencias y tickets

Se utilizó la funcionalidad de *tickets* de GitLab para poder así llevar una mejor organización de los problemas encontrados en las pruebas de la aplicación y del módulo.

3.5.5 Licencias

Todavía no se ha decidido la licencia con la que se liberará el código desarrollado, pudiendo ser diferente la licencia utilizada en el módulo de la utilizada en la aplicación.

A continuación se detallan las licencias del código y librerías utilizadas:

- Módulo inicial de mapa: Apache Public License, Version 2.0.
- Proj4j: Apache Public License, Version 2.0.
- LWKT (Francesco Cutruzzula): LGPL license
- Android GeoJSON: The MIT License
- Appcelerator Titanium: Apache Public License, Version 2.0.

3.5.6 Código

Como ya se ha dicho anteriormente, todavía no se ha decidido la licencia con la que publicar el software por lo que actualmente el código se encuentra en un repositorio privado.

Una vez se tome la decisión, la información del proyecto y el enlace al repositorio en el que se haya liberado bajo la licencia acordada se publicará en el siguiente enlace: http://conwet.fi.upm.es/es/projects.

4 CONCLUSIONES

4.1 Resultados

Este Trabajo de Fin de Grado ha resultado en la implementación de un módulo de mapa para Appcelerator Titanium capaz de gestionar múltiples capas de servicios WMS, visualizar ficheros KML, WKT y GeoJSON, realizar un seguimiento de la posición del usuario accediendo a características hardware como el GPS y la brújula y de hacer uso de la aceleración gráfica que ofrece OpenGL para la visualización del mapa de manera más fluida, a la vez que es capaz de soportar grandes cantidades de datos (por ejemplo, muchos POIs; es decir, puntos de interés).

Por otra parte, se ha desarrollado una aplicación para Appcelerator Titanium que hace uso de este módulo y, basándose en él, ofrece características de más alto nivel como búsqueda de calles, creación y visualización de vistas y lugares favoritos, seguimiento de localización, etc.

Esta aplicación se ha integrado además con Wirecloud4tablet, creando así un *widget* nativo que pueda sustituir a la versión *web* (HTML) del *widget* de Wirecloud en todos los *mashups* que lo utilicen. De esta manera, se pueden explotar todas las ventajas que ofrece una aplicación nativa a la vez que se mantienen las ventajas que ofrece Wirecloud para facilitar el desarrollo de aplicaciones composicionales mediante *mashup*.

Además, este módulo va a poder ser integrado (en un futuro, ya no como parte de este Trabajo de Fin de Grado) dentro del *framework* que se está desarrollando actualmente en el Center for Open Middleware, sobre el que está construido, por ejemplo, Wirecloud4tablet. El módulo pasaría a ser una API más de las que componen ese framework.

El haber desarrollado un *widget* nativo para Wirecloud4tablet puede suponer un hecho importante para el Instituto Geográfico Nacional (que ya está interesado en el desarrollo de aplicaciones mediante Wirecloud) ya que sólo dispone actualmente de un visualizador web y este avance les permitirá poder empezar a llevar sus servicios a plataformas móviles, las cuales están actualmente en auge, ampliando de esta manera el alcance de sus servicios.

En resumen, se puede decir que mediante este Trabajo de Fin de Grado se han desarrollado los elementos que faltaban para poder crear aplicaciones composicionales nativas multiplataforma que hagan uso de visualización de información geográfica; es decir, se pueden crear aplicaciones en pocos minutos y sin conocimientos de programación, que se pueden ejecutar de manera nativa (al menos el mapa) en múltiples plataformas (aunque en este TFG sólo se haya desarrollado para Android al no haber tiempo para realizarlo para más plataformas).

4.2 Conclusiones personales

Este proyecto ha servido para completar mi formación académica como estudiante,

desde la perspectiva de un proyecto software real, que puede ser usado fuera del ámbito académico.

Este proyecto me ha enriquecido intelectualmente. Me ha permitido conocer en un alto grado de profundidad diferentes plataformas, tecnologías y estándares. Entre las plataformas que he aprendido se pueden destacar Android y Appcelerator Titanium, las cuales, en mi opinión, son un conocimiento muy importante debido a que las aplicaciones móviles están cobrando cada día mayor importancia.

Además, también he tenido que tomar una gran cantidad de decisiones al estar trabajando sobre un entorno de gran incertidumbre, lo cual me ha obligado a explotar posibles formas de dar solución a problemas aún no resueltos (hasta donde sé), antes de encontrar una solución funcional para ellos.

Por otro lado, también he adquirido conocimientos sobre los diferentes estándares utilizados en geolocalización y visualización (WMS, WFS, WMTS, ...) lo cual puede ser muy interesante de cara al currículum, al tratarse de unos estándares que muy poca gente del mundo del desarrollo de software conoce.

Además, fruto de mi trabajo ligado al TFG como becario en el proyecto GeoWidgets de la UPM con el CNIG, durante el desarrollo del TFG tuve la oportunidad de presentar una ponencia a las Jornadas Español en Tecnologías GIS abiertas que diseminaba algunos resultados preliminares del trabajo:

J. Sánchez, J. Soriano, A. Vera, B. Illescas, A.F Rodríguez, *Nuevo entorno de explotación y visualización de datos geoespaciales basado en la plataforma Wirecloud,* 8as Jornadas de GIS Libre, Girona, España, 26-28 de marzo, 2014

4.3 Líneas futuras

Una vez finalizado este Trabajo de Fin de Grado se abren diferentes alternativas para continuarlo:

- Añadir soporte para iOS y Windows Phone, dado que el módulo de mapa que se ha desarrollado solo es para Android.
- Aumentar los estándares que soporta el módulo de mapa. Se podría intentar añadir WMTS y WMS-C para las capas. También se podría añadir polígonos SVG al soporte de elementos vectoriales.
- Desarrollo de *mashups* que puedan utilizar las capacidades nativas del *widget* desarrollado para la plataforma Wirecloud4tablet.
- Integración del módulo como parte de la API del framework sobre el que está construido Wirecloud4tablet.

De hecho, los resultados de este proyecto ya han hecho que algunas líneas hayan sido incluidas en el *roadmap* real de Wirecloud4Tablet y de Yaast (el framework en el que está trabajando el Center foo Open Middleware).

5 REFERENCIAS

[1] eMarketer. Smartphone Users Worldwide Will Total 1.75 Billion in 2014.
http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536 – Última vez accedido: 05/05/2014

[2] Danyl Bosomworth. Mobile Marketing Statistics 2014.
http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ – Última vez accedido: 05/05/2014

[3] Jordi Linares. State of the art of Mobile development technologies.
http://mobile.alc.upv.es/mobweb/local/lahti/materiales/slides_mobweb_stateofartMobile.pdf – Última vez accedido: 05/05/2014

[4] Wikipedia. Dalvik.
http://es.wikipedia.org/wiki/Dalvik – Última vez accedido: 05/05/2014

[5] Stefan Brähler. Analysis of the Android Architecture.
http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf – Última vez accedido: 05/05/2014

[6] Android-app-market. Android Architecture – The Key Concepts of Android OS
http://www.android-app-market.com/android-architecture.html – Última vez accedido: 05/05/2014

[7] Apple. Which Developer Program is for you.
https://developer.apple.com/programs/which-program/ – Última vez accedido: 05/05/2014

[8] Wikipedia. IOS.
http://es.wikipedia.org/wiki/IOS – Última vez accedido: 05/05/2014

[9] Wikipedia. Managed code.
http://en.wikipedia.org/wiki/Managed_code – Última vez accedido: 05/05/2014

[10] Wikipedia. Windows Phone.
http://en.wikipedia.org/wiki/Windows_Phone – Última vez accedido: 05/05/2014

[11] Mozilla. Firefox OS.
https://developer.mozilla.org/es/docs/Mozilla/Firefox_OS – Última vez accedido: 05/05/2014

[12] Heiko Behrens. Cross-Platform App Development for iPhone, Android & Co. — A

Comparison I Presented at MobileTechCon 2010.
http://heikobehrens.net/2010/10/11/cross-platform-app-development-for-iphone-android-co-%E2%80%94-a-comparison-i-presented-at-mobiletechcon-2010/ – Última vez accedido: 05/05/2014

[13] Gobierno de Navarra. Sitna.
http://sitna.navarra.es/geoportal/recursos/mobile.aspx – Última vez accedido: 05/05/2014

[14] Google Play. GIS WMS Viewer.
https://play.google.com/store/apps/details?id=air.WMSViewer.MapGoGIS.app&hl=es – Última vez accedido: 05/05/2014

[15] Google Play. Citysurf Globe.
https://play.google.com/store/apps/details?id=com.citysurf.globe – Última vez accedido: 05/05/2014

[16] Venkata Kiran. Native vs Mobile Web vs Hybrid applications.
http://www.javacodegeeks.com/2013/12/native-vs-mobile-web-vs-hybrid-applications.html – Última vez accedido: 05/05/2014

[17] Nomad Mobile Guides. Differences between mobile websites and native apps.
http://nomadmobileguides.com/making-an-app/frequently-asked-questions/difference-between-mobile-websites-and-native-apps/ – Última vez accedido: 05/05/2014

[18] Martin Dobrev. HTML Maps vs Native Maps on Android.
https://www.youtube.com/watch?v=ze9V7SImBc4&feature=youtu.be – Última vez accedido: 05/05/2014

[19] Appcelerator. Titanium Module Concepts.
http://docs.appcelerator.com/titanium/latest/#!/guide/Titanium_Module_Concepts – Última vez accedido: 05/05/2014

[20] Wikipedia. Same Origin Policy.
http://en.wikipedia.org/wiki/Same-origin_policy – Última vez accedido: 05/05/2014

[21] Frank Ableson. Titanium 1.0: The “App for Building Apps”?
http://www.linux-mag.com/id/7719/ – Última vez accedido: 05/05/2014

[22] Strategy Analytics. Android Captured 79% Share of Global Smartphone Shipments in 2013.
http://blogs.strategyanalytics.com/WSS/post/2014/01/29/Android-Captured-79-Share-of-Global-Smartphone-Shipments-in-2013.aspx – Última vez accedido: 05/05/2014

[23] IDC. Worldwide Tablet Growth Forecast to Slow as New and Replacement Purchases in Mature Markets Begin to Level Off, According to IDC. http://www.idc.com/getdoc.jsp?containerId=prUS24716914 – Última vez accedido: 06/05/2014

6 APÉNDICE A: PLAN DE TRABAJO

6.1 Plan inicial

A continuación se detalla la primera planificación que se realizó antes de empezar con el TFG. Al haberse realizado antes de empezar a trabajar en el TFG la planificación no fue muy precisa y posteriormente hubo que hacer recortes en los objetivos del TFG una vez se tuvo una idea de la velocidad de desarrollo.

6.1.1 Lista de objetivos del trabajo

- Desarrollo de un módulo para Appcelerator

  Se quiere desarrollar un módulo de mapa que soporte las siguientes características:

  - Posicionamiento GPS y orientación mediante brújula.
  - Soporte para capas WMS y WMTS.
  - Visualización de elementos vectoriales SVG, WKT, KML o GeoJSON.
  - Soporte para insertar elementos geolocalizables.
  - Visualizador jerárquico de servicios y capas disponibles (búsqueda, selección), con posibilidad de añadir nuevos servicios y capas.
  - Soporte para mapas "*offline*".

- Desarrollo de una App base de mapas

  Se quiere desarrollar una aplicación de Appcelerator que haga uso del módulo anteriormente descrito. Esta aplicación debe presentar las siguientes características:

  - Navegación por el mapa y controles.
  - Visualización de la capa base y superposición de capas.
  - Consultas de información de capas (GetFeatureInfo).
  - Visualizador de leyendas de las capas.
  - Buscador de lugares (WFS).
  - Gestionar favoritos.

- Funcionalidad del lado del servidor

  Se quiere considerar el uso de *cloud* para poder agilizar tareas como la búsqueda de datos cacheándolos, obtener el listado de servicios y capas disponibles, funcionalidad de auto-completado, etc.

- Integración con la plataforma Wirecloud4Tablet

  Se quiere realizar la integración con la plataforma Wirecloud4Tablet, la cual se trata de una plataforma que permite la creación de *mashups* (al igual que

Wirecloud) compuestos por *widgets* que pueden ser nativos o escritos en Javascript y que pueden utilizar la funcionalidad que proporcionan los dispositivos móviles como puede ser GPS, brújula, acelerómetro, cámara, etc. Si se encuentra disponible un *widget* nativo para la plataforma del usuario, se cargará de forma totalmente transparente, produciendo una gran mejora en el rendimiento respecto a la versión escrita con tecnologías web.

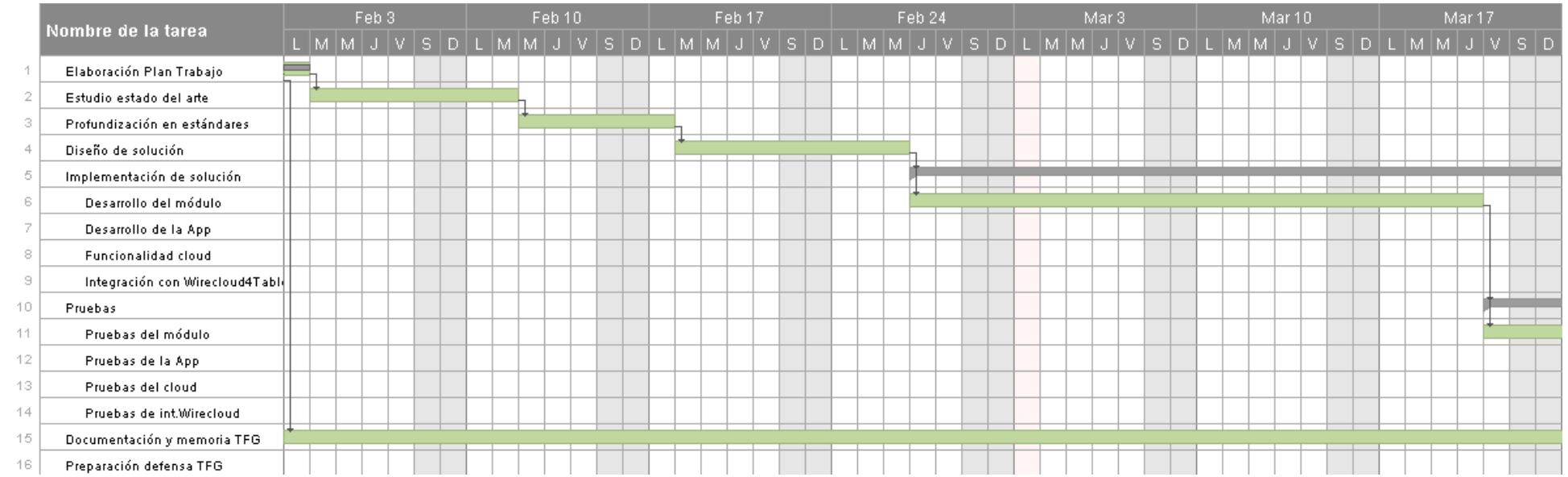
El objetivo del TFG es que la *App* nativa pase a ser un *widget* de la plataforma y pueda ser utilizado con otros *widgets* de la plataforma aunque estos no sean en nativo sino escritos en HTML, CSS y Javascript.

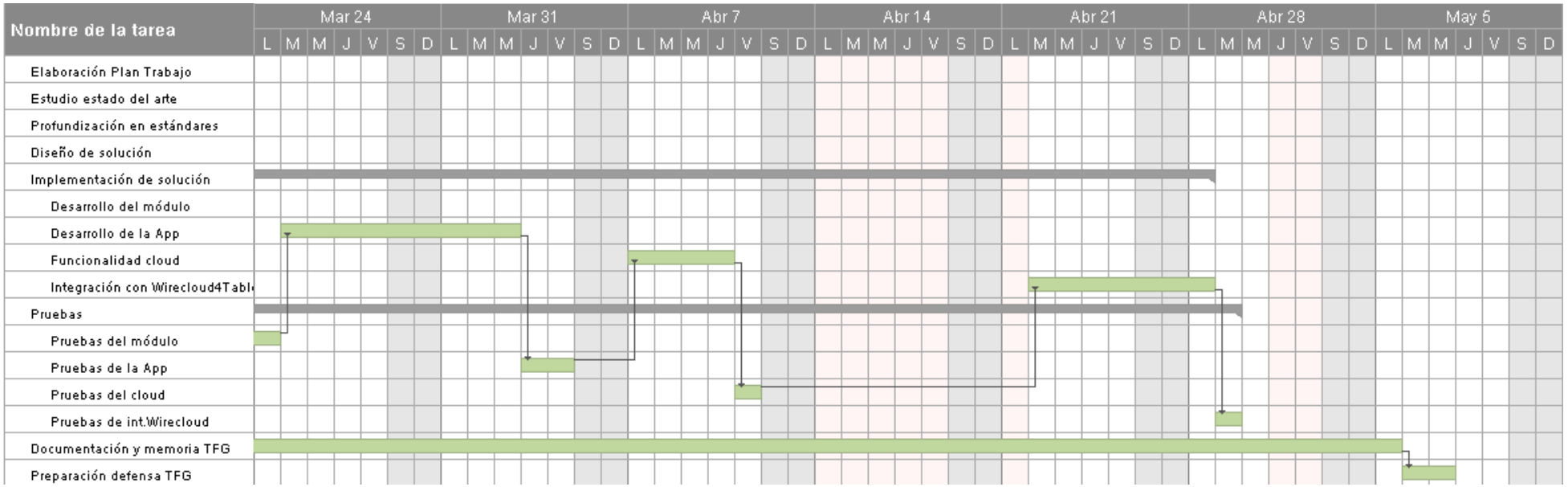
6.1.2 Lista de tareas

Para alcanzar los objetivos indicados en el apartado anterior así como los objetivos del TFG será necesario realizar las siguientes tareas:

- Elaboración junto con el tutor del plan de trabajo a seguir (4 horas).
- Estudio del estado del arte: análisis comparativo y documentación de las aproximaciones y soluciones existentes (30 horas).
- Profundización en estándares (e.g. WMS, WMTS, WFS), sistemas operativos móviles (e.g. iOS, Android), lenguajes de programación (e.g. Objective C), frameworks (e.g. Titanium), herramientas y SDKs (e.g. ArcGIS, MapBox, OpenLayers) que puedan resultar útiles para el desarrollo del proyecto (20 horas).
- Diseño de la solución (35 horas)
- Implementación de la solución (145 horas), que a su vez se subdivide en:
  - Desarrollo del módulo Appcelerator (75 horas).
  - Desarrollo de la App de mapas (35 horas).
  - Funcionalidad en el lado servidor (20 horas).
  - Integración con la plataforma Wirecloud4Tablet (25 horas).
- Pruebas unitarias y de integración (30 horas).
- Elaboración de la documentación del proyecto y de la memoria del TFG (50 horas).
- Preparación de la defensa del trabajo (10 horas).

6.1.3 Diagrama de Gantt





6.2 Revisión de la planificación inicial

Tras varios meses de desarrollo, se ha decidido revisar la planificación inicial para así poder hacer frente a los problemas surgidos durante la implementación así como a los errores en la estimación del tiempo por cada tarea y la carga de trabajo de los objetivos.

Los cambios que ha sufrido la lista de objetivos de la planificación inicial son los siguientes:

- Tras un análisis de SVG, se ha decidido no soportar su visualización en el mapa ya que no se trata de un estándar que sea utilizado para la definición de rutas en mapas, sino más bien para la creación de gráficos de alta calidad. Por lo tanto, se ha decidido no soportar este estándar, pero mantener WKT, KML y GeoJSON que sí que son más habituales.
- Tras dos semanas intentando dar soporte a WMTS, se ha decidido no soportarlo debido a la gran dificultad que estaba suponiendo y el hecho de que se trata de una estándar muy poco utilizado (el estándar más antiguo, WMT, tiene mucha mayor adopción a pesar de ser menos eficiente).
- Se ha decidido eliminar el soporte de mapas "*offline*" ya que se considera que no aporta nada al proyecto y que además los dispositivos móviles no suelen tener capacidad suficiente para poder albergar todo un mapa (ya que un mapa puede estar compuesto por miles de imágenes).
- Se ha decidido eliminar toda la funcionalidad *cloud* ya que se ha considerado que no tenía mucha relación con el proyecto ya que esta funcionalidad básicamente sería el despliegue de un *proxy* caché y de un repositorio de iconos. Debido a la falta de tiempo, esta es una funcionalidad claramente prescindible.
- Tras un análisis conjunto con los desarrolladores de la plataforma de Wirecloud4Tablet se ha reducido la estimación de tiempo en la integración con dicha plataforma debido a que si la aplicación se desarrolla desde un principio siguiendo unas directrices se podrá ahorrar mucho tiempo en la posterior integración.

Debido a los cambios en la lista de objetivos, también ha habido cambios en la lista de tareas:

- El tiempo de desarrollo del módulo, ya finalizado, ha sido de 145 horas (frente a las 75 planificadas) debido a la gran complejidad y las dificultades encontradas.
- Se ha aumentado el tiempo de desarrollo de la aplicación en 35 horas más, sumando un total de 70 horas, debido al tiempo de aprendizaje, ya que nunca antes he desarrollado una aplicación con Titanium.
- Se ha eliminado el tiempo del desarrollo y pruebas de la funcionalidad *cloud* y se ha reducido el tiempo estimado de integración con Wirecloud4Tablet.
- Tras todos estos cambios, el tiempo estimado para la realización del TFG ha pasado a ser de 404 horas, frente a las 324 de la planificación inicial.

Hay que tener en cuenta que esta estimación se ha realizado suponiendo que sea posible utilizar la librería OpenLayers en el desarrollo de la aplicación para Appcelerator, lo cual facilitaría mucho el desarrollo pero es algo que todavía no ha sido probado. En caso de que no fuese posible utilizarla, el tiempo de desarrollo de la aplicación aumentaría drásticamente y habría que hacer recortes de funcionalidad en la aplicación.

6.2.1 Lista de objetivos del trabajo revisada

- Desarrollo de un módulo para Appcelerator

  Se quiere desarrollar un módulo de mapa que soporte las siguientes características:

  - Posicionamiento GPS y orientación mediante brújula.
  - Soporte para capas WMS.
  - Visualización de elementos vectoriales WKT, KML o GeoJSON.
  - Soporte para insertar elementos geolocalizables.

- Desarrollo de una App de mapas

  Se quiere desarrollar una aplicación de Appcelerator que haga uso del módulo anteriormente descrito. Esta aplicación debe presentar las siguientes características:

  - Navegación por el mapa y controles.
  - Visualización de la capa base y superposición de capas.
  - Consultas de información de capas (GetFeatureInfo).
  - Visualizador de leyendas de las capas.
  - Buscador de lugares (WFS).
  - Gestionar favoritos.
  - Visualizador jerárquico de servicios y capas disponibles (búsqueda, selección), con posibilidad de añadir nuevos servicios y capas.

- Integración con la plataforma Wirecloud4Tablet

  Se quiere realizar la integración con la plataforma Wirecloud4Tablet, la cual se trata de una plataforma que permite la creación de *mashups* (al igual que Wirecloud) compuestos por *widgets* que pueden ser nativos o escritos en Javascript y que pueden utilizar la funcionalidad que proporcionan los dispositivos móviles como puede ser GPS, brújula, acelerómetro, cámara, etc. Si se encuentra disponible un *widget* nativo para la plataforma del usuario, se cargará de forma totalmente transparente, produciendo una gran mejora en el rendimiento respecto a la versión escrita con tecnologías web.

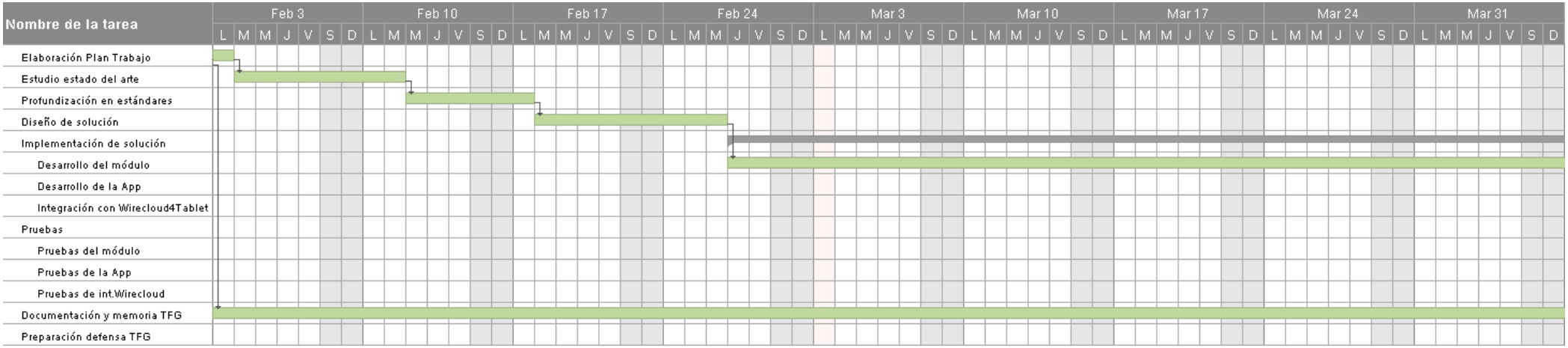
  El objetivo del TFG es que la *App* nativa pase a ser un *widget* de la plataforma y pueda ser utilizado con otros *widgets* de la plataforma aunque estos no sean en nativo sino escritos en HTML, CSS y Javascript.

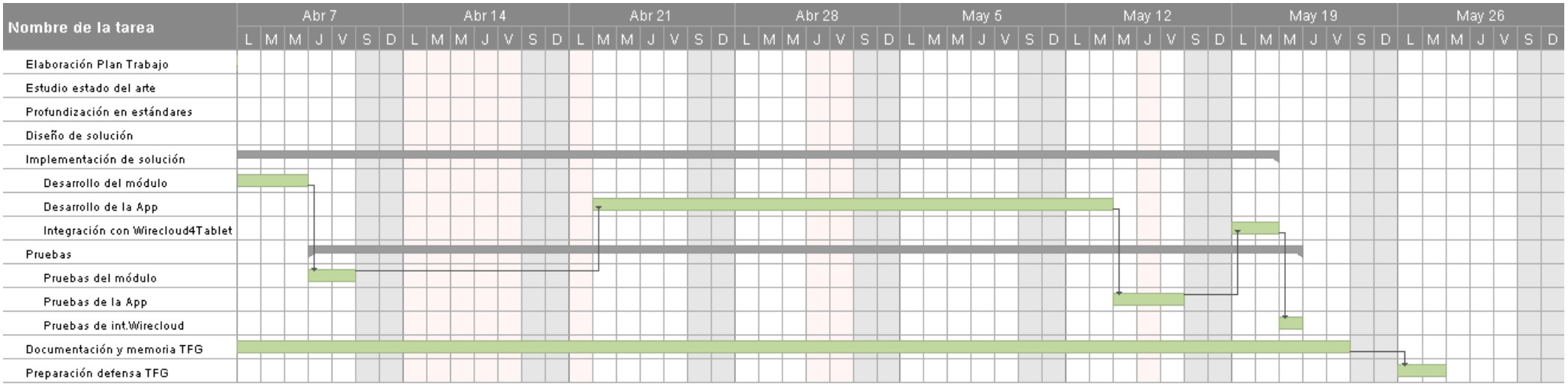
6.2.2 Lista de tareas revisada

Para alcanzar los objetivos indicados en el apartado anterior así como los objetivos del TFG será necesario realizar las siguientes tareas:

- Elaboración junto con el tutor del plan de trabajo a seguir (4 horas).
- Estudio del estado del arte: análisis comparativo y documentación de las aproximaciones y soluciones existentes (30 horas).
- Profundización en estándares (e.g. WMS, WMTS, WFS), sistemas operativos móviles (e.g. iOS, Android), lenguajes de programación (e.g. Objective C), frameworks (e.g. Titanium), herramientas y SDKs (e.g. ArcGIS, MapBox, OpenLayers) que puedan resultar útiles para el desarrollo del proyecto (20 horas).
- Diseño de la solución (35 horas)
- Implementación de la solución (230 horas), que a su vez se subdivide en:
  - Desarrollo del módulo Appcelerator (145 horas).
  - Desarrollo de la App de mapas (70 horas).
  - Integración con la plataforma Wirecloud4Tablet (15 horas).
- Pruebas unitarias y de integración (25 horas).
- Elaboración de la documentación del proyecto y de la memoria del TFG (50 horas).
- Preparación de la defensa del trabajo (10 horas).

6.2.3 Diagrama de Gantt revisado





6.3 Resultado final

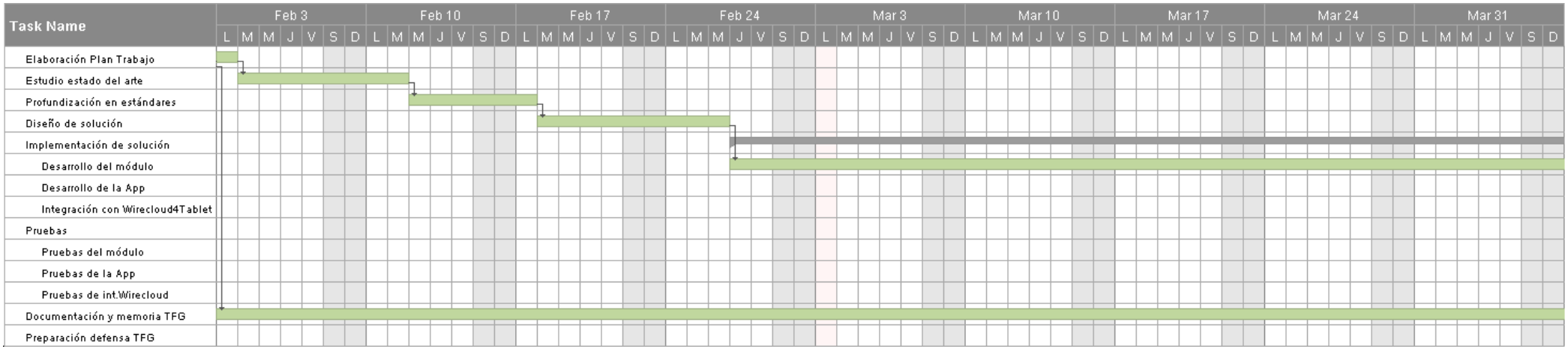
Finalmente se han tardado 25 horas más de las planificadas en la revisión de la planificación que se hizo. Estas desviaciones son lógicas debidas a que nunca antes se había realizado una aplicación para Appcelerator ni se había tenido que integrar una aplicación como un *widget* para Wirecloud4tablet (aunque en el caso de la integración no hubo desviaciones), por lo que es difícil calcular con precisión el tiempo que llevará.

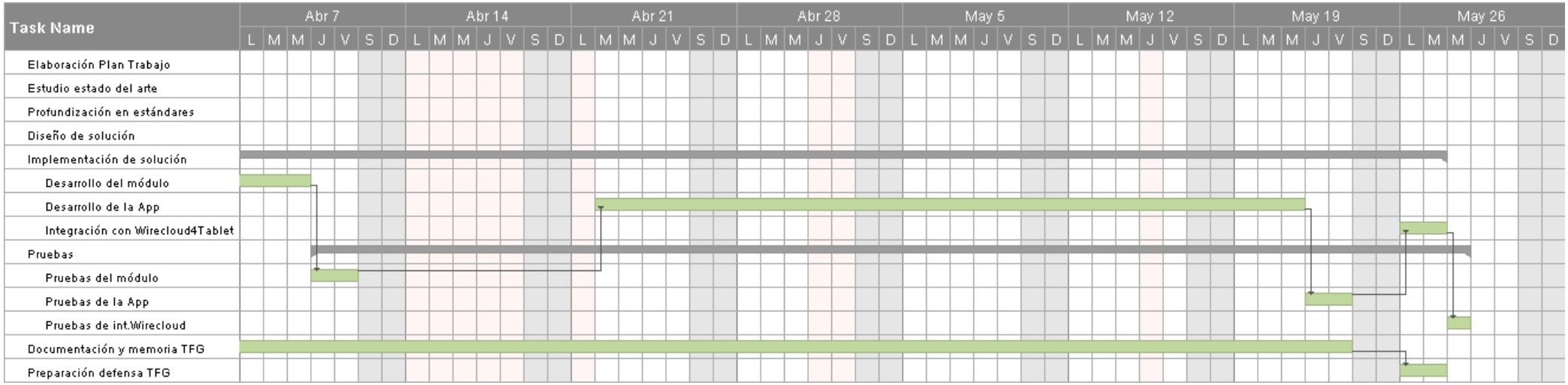
6.3.1 Tiempos finales de tareas

A continuación se indican los tiempos que han supuesto de manera real cada una de las partes del TFG:

- Elaboración junto con el tutor del plan de trabajo a seguir (4 horas).
- Estudio del estado del arte: análisis comparativo y documentación de las aproximaciones y soluciones existentes (30 horas).
- Profundización en estándares (e.g. WMS, WMTS, WFS), sistemas operativos móviles (e.g. iOS, Android), lenguajes de programación (e.g. Objective C), frameworks (e.g. Titanium), herramientas y SDKs (e.g. ArcGIS, MapBox, OpenLayers) que puedan resultar útiles para el desarrollo del proyecto (20 horas).
- Diseño de la solución (35 horas)
- Implementación de la solución (230 horas), que a su vez se subdivide en:
  - Desarrollo del módulo Appcelerator (145 horas).
  - Desarrollo de la *App* de mapas (95 horas).
  - Integración con la plataforma Wirecloud4Tablet (15 horas).
- Pruebas unitarias y de integración (25 horas).
- Elaboración de la documentación del proyecto y de la memoria del TFG (50 horas).
- Preparación de la defensa del trabajo (se realizará tras la entrega de esta memoria, se estiman 10 horas).

6.3.2 Diagrama de Gantt final





Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Fri Jun 06 22:29:20 CEST 2014
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)