



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Desarrollo e implementación de software de localización en tiempo real para dispositivos móviles

Proyecto Final de Carrera

Ingeniería Informática

Autor: Jorge García Cebriá

Director: Juan Vicente Capella Hernández

23/05/2012

Resumen

En el presente proyecto se mostrará el desarrollo, diseño e implementación de un software de localización en tiempo real para dispositivos móviles. El proyecto permite una comunicación entre diferentes dispositivos móviles, independientemente de su sistema operativo, para facilitar la localización de cualquier contacto de la agenda telefónica. Para ello, ha sido necesario el aprendizaje de diferentes tecnologías relacionadas con el desarrollo de las distintas plataformas móviles, el desarrollo de sockets seguros, y la implementación de un sistema de gestión de contenidos.

Palabras clave: móviles, socket, iOS, Android, localización.



Tabla de contenidos

1.	Introducción.....	9
1.1.	Objetivos iniciales	10
1.2.	Motivación	11
1.3.	Estrategias	12
1.4.	Planificación.....	13
2.	Análisis y tecnologías.....	16
2.1.	Análisis	16
2.1.1.	Requisitos no funcionales.....	16
2.1.2.	Requisitos funcionales	17
2.2.	Plataformas y tecnologías utilizadas.....	18
3.	Diseño e implementación	23
3.1.	Especificación de casos de uso.....	25
3.1.1.	Casos de uso relacionados con las acciones del usuario.	25
3.1.2.	Casos de uso relacionados con la aplicación	26
3.1.3.	Casos de uso servidor web	28
3.2.	Diseño e implementación del entorno Drupal.....	30
3.2.1.	Diseño	32
3.2.2.	Implementación	36
3.2.3.	Módulo Services	40
3.3.	Diseño e implementación software móvil.....	43
3.3.1.	Registro del usuario	45
3.3.2.	Pantalla contactos	47
3.3.3.	Interfaz	49
3.3.4.	Localización y orientación	55
3.3.5.	Conexiones de red.....	61
3.4.	Diseño e implementación servidor Echo	65
3.4.1.	Diseño del servidor echo	66
3.4.2.	Implementación del servidor echo	68
3.4.3.	Implementación móvil de conexión al servidor.	70

4. Manual de usuario	75
5. Conclusiones	80
6. Mejoras para el futuro	82
7. Bibliografía	85



1. Introducción

Las nuevas plataformas móviles, los denominados smartphones, son uno de los dispositivos con más aceptación y éxito en la actualidad. Las posibilidades que nos pueden ofrecer un ordenador, un lector de libros electrónicos o un reproductor de música, por poner solo unos ejemplos, las podemos tener actualmente en el mismo dispositivo. Disponen de conexión a Internet y suelen ser ofrecidos a los usuarios a través de las operadoras telefónicas, por lo cual el número de usuarios que disponen de estos sistemas va en aumento año tras año. Actualmente el mercado de smartphones en España es del 35% de los teléfonos móviles totales. Solo Singapur, Canadá, Hong Kong y Suecia superan este porcentaje (Infographics, 2012)

Dentro de este porcentaje, dos sistemas operativos copan el mayor porcentaje del mercado: Android de Google, y iOS de Apple, obteniendo ambos casi la totalidad del mercado español. (Data.es, 2012)

Sin embargo, en términos globales, Android e iOS ocupan más del 75% del mercado de smartphones, siendo los dos mayores exponentes de este ecosistema. Y las ganancias se reparten casi exclusivamente entre ambos con casi un 90% de los beneficios. (mobilemix, 2011).

Parte de los beneficios provienen de las aplicaciones disponibles para descarga. Aplicaciones de mensajería instantánea, videojuegos, redes sociales... Casi un 60% de estos usuarios dedican su tiempo con su Smartphone en torno a las redes sociales (Facebook, Twitter...) (Infographics, 2012)

Así pues, hemos visto como los usuarios disponen de smartphones, cómo consumen aplicaciones y cómo las compañías obtienen beneficios.

Muchas de estas aplicaciones exitosas se basan en un concepto: *“estar siempre disponible”* o *“estar siempre conectado”*. Sin embargo, no existen aplicaciones que lleven ese concepto a la localización de contactos. Existen aplicaciones para enviar mensajes, compartir fotos o compartir tu ubicación, pero no de localizar exactamente al usuario, de cómo llegar hasta él y que cualquier cambio de posición o desplazamiento sea notificado. Y es esa la motivación que llevó al desarrollo del presente proyecto: un sistema de localización de contactos en tiempo real.

1.1. Objetivos iniciales

Los objetivos propuestos en el desarrollo del presente proyecto son principalmente el aprendizaje en el uso de nuevas plataformas móviles, así como la integración de estas con el exterior. Realizar un proyecto y planificar su desarrollo, así como de las diferentes infraestructuras o servicios que serían necesarios realizar. A su vez, otro objetivo importante es el manejo y especialización en el desarrollo de las dos plataformas móviles con mayor éxito del mercado, las cuales son *Android* y *iOS*, propiedad de Google y Apple respectivamente. Estos conocimientos serán utilizados para comparar, siempre desde el punto de vista del desarrollador, las virtudes e inconvenientes de ambas plataformas y sus respectivos kits de desarrollo. Saber desarrollar para ambas plataformas, incluyendo el uso de servicios de red, geolocalización y gráficos, además de planificar correctamente un proyecto, es de vital importancia para una buena formación dentro de este sector.

1.2. Motivación

Si bien hay muchos clientes de comunicación (*Whatsapp, Viber, Tango...*) no existen tantos clientes de localización, y los que existen son muy básicos. Tanto *Google Latitude* (Google) como *Find my Friends* (Apple), las aplicaciones más conocidas en estos casos, no son clientes de comunicación en tiempo real. Son aplicaciones que notifican a los usuarios de una posición en un determinado momento, sin actualizaciones automáticas de posición ni indicaciones de cómo llegar hasta tu contacto. Es por ello que se usan los propios clientes de mensajería ya mencionados cuando existe la necesidad de localizar a cualquier contacto, ya que los servicios de Google y Apple no cumplen esa necesidad al 100%.

Esta aplicación permite ambas cosas. La comunicación en tiempo real a los usuarios de cualquier cambio de posición y a su vez, una indicación simple de cómo llegar o qué distancia falta hasta llegar a tu amigo. Y todo ello conservando la premisa de ser multiplataforma: es una aplicación desarrollada para los dos sistemas móviles de más éxito: *iOS* y *Android*.



1.3. Estrategias

Dadas las motivaciones ya expuestas, se decidió por concretar la política y estrategia a seguir. Se optó por desarrollar una aplicación con una premisa clara: simplicidad. Los usuarios de esta aplicación deberán poder usar todas las funcionalidades y posibilidades ofrecidas de una manera simple y directa. Pero para ello había que decidir previamente las tecnologías a usar.

Básicamente había que decidir tres claves básicas: que sistemas operativos usar, cómo y dónde almacenar los datos, y cómo permitir la comunicación entre usuarios.

En el primer caso y dado que uno de los enfoques del actual proyecto es la multiplataformidad y así demostrar las similitudes, diferencias, virtudes y defectos de cada sistema, se optó por desarrollar la aplicación con dos sistemas clave: *Android* e *iOS*. Si bien existen más sistemas operativos móviles, algunos de mucho éxito como *Blackberry* o *Symbian*, o en notable crecimiento como *Windows Phone*, se optó por los dos más comunes, tanto por posibilidades de mercado, como por facilidades de uso. *Android* ofrece un entorno de programación basado en Java, mientras que *iOS* ofrece lo propio en *Objective-C*, un lenguaje con ciertas similitudes a C y *Smalltalk*.

En cuanto al sistema de gestión de base de datos, necesitamos un sistema para mantener los usuarios y así poder mantener una gestión de éstos. Es por ello que se decidió utilizar *MySQL*. Para facilitar el mantenimiento y usabilidad de todo el sistema de gestión, se decidió integrar un CMS (*Content Management System*). Un CMS, o sistema de gestión de contenidos, es un gestor que permite crear una estructura de soporte para la creación y administración de contenidos, principalmente en páginas web. Se optó por usar *Drupal*, dado que es un sistema libre, con licencia GNU/GPL, escrito en PHP y desarrollado y mantenido por una activa comunidad de usuarios.

Por último, el sistema de comunicación entre usuarios. Actualmente y debido a las operadoras, la comunicación p2p¹ no es permitida y requiere unos cambios a nivel de conexión del cliente que puede no estar dispuesto a asumir. Es por ello que la creación de un nodo intermedio, que se encargue de repetir los mensajes, fue considerado como la opción más plausible. A este nodo lo llamaremos servidor de repetición. Desarrollado a nivel de sockets, debería permitir la conexión de diferentes clientes, asociarlos entre ellos y reenviar cada mensaje de localización recibido a su correspondiente destinatario. El destinatario, que sería el teléfono móvil, sería el encargado de interpretar la información recibida. Aunque otros lenguajes como Python ofrecen librerías de comunicación de sockets ya establecidas, se decidió utilizar C++ por el conocimiento del lenguaje y a su vez, la posible inclusión de seguridad SSL.

¹ Peer-to-peer: conexión directa entre clientes

1.4. Planificación

1.4.1. Fases del proyecto

El proyecto se divide en cuatro grandes fases que se detallan a continuación:

Fase 1 – Plan de trabajo (PAC 1)

En esta fase se han realizado las siguientes actividades:

- Analizar e instalar el software necesario para la documentación y realización del proyecto:
- Apple iOS SDK²: iOS5
- Google Android SDK 11
- Eclipse Classic 3.7.
- Netbeans IDE 7.1
- Drupal 6.22
- Analizar y estudiar proyectos.
- Elección de proyecto a desarrollar.
- Desarrollar el documento de plan de trabajo.

Fase 2 – Análisis y diseño

En esta fase se han realizado las siguientes actividades:

- Profundizar en las tecnologías escogidas para el desarrollo del proyecto.
- Especificar aplicación.
- Especificación nombre del proyecto temporal.

Esta tarea ha sido una de las más importantes del proyecto. Además se ha definido la estructura del proyecto así como los distintos diagramas.

Fase 3 – Implementación

En esta fase se han realizado las siguientes actividades:

- Crear sistema de base de datos
- Desarrollar aplicación para iOS
- Desarrollar aplicación para Android
- Desarrollar servidor de repetición
- Realizar pruebas y corregir errores.
- Realizar documentación.
- Especificación nombre del proyecto y aplicación.

Fase 4 – Memoria y presentación virtual

² SDK: Software Development Kit. Conjunto de herramientas de desarrollo de software que permite al programador crear aplicaciones para un sistema concreto.

En esta fase se realizarán las siguientes actividades:

- Confeccionar la memoria del proyecto.

2. Análisis y tecnologías

En el presente capítulo detallaremos un análisis de la realización del proyecto, así como de todas las tecnologías necesarias para llevarlo a cabo.

2.1. Análisis

Este apartado detalla el análisis inicial realizado sobre todas las necesidades y requisitos que finalmente será desarrollado en este proyecto.

Requisitos

Existen dos tipos de requisitos:

- **Requisitos funcionales:** son aquellos que consisten en una característica requerida del sistema que expresa una capacidad de acción del mismo, generalmente expresada en una declaración en forma verbal.
- **Requisitos no funcionales:** son aquellos que consisten en una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

2.1.1. Requisitos no funcionales

- **Sistema operativo móvil:** La aplicación móvil solo funciona bajo las plataformas Android 2.2 o superior y iOS versión 4.0 o superior.
- **Hardware móvil:** La aplicación móvil solo funciona en aquellos terminales que dispongan de conexión a Internet y GPS, además de magnetómetro (brújula). La falta de cámara trasera y/o la ausencia de giroscopio, si bien no es requerida, puede perjudicar notablemente la experiencia de usuario.
- **Servidor web *Drupal*:** El servidor web requiere de un mínimo de 15MB libres de disco duro, además de soporte a *PHP* 4.4.0 o superior, *Apache/Microsoft IIS* y *MySQL* 4.1 o superior.
- **El servidor de repetición** requiere un entorno *C/C++* para su compilación y posterior ejecución.

- La aplicación móvil para iOS se ha desarrollado para *Xcode* 4.2.1 bajo el SDK 5.0.
- La aplicación móvil para Android se ha desarrollado bajo *Eclipse Classic* 3.7.1, usando el SDK 11 de Android.

2.1.2. Requisitos funcionales

La aplicación cuenta con las siguientes funcionalidades:

- Conexión con el servidor Drupal
Mediante la conexión al servidor web podremos sincronizar nuestros contactos, registrarnos o cambiar preferencias sobre nuestra localización.
- Cambio de skin³
La aplicación permite cambiar entre dos skins bien diferenciados. Uno de ellos se ha desarrollado bajo OpenGL ES 1.1 y el otro con las herramientas nativas de cada SO (CoreGraphics en iOS y Canvas en Android).
- Intercambio de mensajes entre usuarios
La aplicación recibe y envía mensajes de localización a los usuarios, que son reconocidos e interpretados según la necesidad.
- Redes sociales
La aplicación permite compartir en *Facebook* y *Twitter* una imagen recreada por *Google Maps* del trayecto realizado en la localización del usuario.

³ Skin: piel en inglés, en el contexto del software se refiere a una apariencia gráfica modificable.

2.2. Plataformas y tecnologías utilizadas

Dado que hay tres partes clave dentro del proyecto (software móvil, BBDD y software servidor), vamos a indicar por separado el desglose usado:

Software móvil:

Por una parte es evidente que para desarrollar para plataformas distintas, es necesario unos SDK distintos. Por lo tanto, ha sido necesario disponer tanto del SDK de Android, ofrecido por Google de manera gratuita, como el SDK de iOS, ofrecido por Apple también gratis. Como inconveniente, mencionar que el SDK de iOS solo está disponible para Mac.

Apple ofrece junto al SDK, un entorno de desarrollo denominado XCode. XCode es el entorno más comúnmente usado, aunque no el único, para el desarrollo de aplicaciones en iOS y MacOSX. Por otra parte, Google no ofrece ningún entorno de desarrollo, aunque se recomienda usar Eclipse, el cual ha sido la opción utilizada.

En cuanto a los lenguajes de programación son dos: *Objective-C* para iOS y Java para Android.

- **Objective-C:** Es un lenguaje orientado a objetos, clasificado como *superset* de C, lo cual quiere decir que cualquier programa en C válido también lo es en *Objective-C* pero además añade una sintaxis y semántica similar a *Smalltalk*. *Objective-C* fue creado en 1986 por Brad Cox y Tom Love de StepStone. Sin embargo NeXT compró la licencia en 1988, y tras la compra de ésta por parte de Apple, se ha convertido en lenguaje de programación usado en todas sus plataformas (OSX y iOS).
- **Java:** Java es un lenguaje orientado a objetos con una sintaxis similar a C/C++ pero con un modelo de objetos simplificado y elimina herramientas de bajo nivel. Las aplicaciones Java están compiladas en un bytecode, el cual es interpretado en tiempo de ejecución. Java fue creado por Sun Microsystems a principios de los 90.

Existen frameworks multiplataformas, como *PhoneGap* o *Appcelerator*, que bajo el uso de lenguajes ajenos como HTML y Javascript, permiten desarrollar un solo proyecto compatible para varias plataformas, entre las que se incluyen iOS y Android. Sin embargo, existían diversas razones para no utilizar esta opción.

La primera de ellas es que desarrollando nativamente, aprovechas más los recursos del sistema, clave en terminales móviles de gama baja o media.

La segunda y no menos importante, es que una de las finalidades del proyecto era establecer un conocimiento en cuanto a las características de desarrollo para ambas plataformas.

Ambas plataformas ofrecen un SDK completo que permiten al desarrollador no tener que conocer frameworks o API⁴s externas. Sin embargo, en el presente proyecto se añadió un breve uso de OpenGL ES (*Embedded Systems*), con lo cual fue necesario un breve conocimiento de dicha API. Si bien Java y Cocoa Touch⁵ pueden ofrecer ligeras diferencias en la forma de tratar diversos aspectos de la API, la mayor parte del código y funcionalidad permanece idéntica en ambas plataformas. Hay que mencionar que si bien OpenGL ES 2.0 es compatible con los dispositivos móviles del actual proyecto, se optó por usar OpenGL ES 1.1, al ser más similar a los conocimientos adquiridos durante la carrera en la asignatura de IGU (Interfaz Gráfica de Usuario).

En cuanto al aspecto hardware, los terminales compatibles para dicho proyecto deberán ofrecer soporte a giroscopio, GPS, brújula y cámara trasera.

- El giroscopio nos permite conocer el grado de inclinación del terminal para adecuar los gráficos en pantalla.
- El magnetómetro o brújula, en cambio, nos permite conocer la orientación del usuario respecto al eje magnético de la tierra.
- El GPS es la clave de todo el proyecto, ya que es necesario disponer de un método para obtener la localización de los usuarios. Resulta indiferente si se utiliza GPS por satélites o A-GPS (**Assisted Global Positioning System**).
- La cámara trasera permitirá añadir cierto aspecto de realidad aumentada al superponer los objetos sobre la visualización de ésta.

Ya por último, los terminales deben ser compatibles con las notificaciones *push*⁶. Las notificaciones *push* nos permitirán alertar a los usuarios cuando una persona quiere contactar con ellos, sin necesidad de que la aplicación esté abierta. Las notificaciones *push* fueron añadidas por Apple en iOS3.0, mientras que Google las introdujo en su versión 2.2 de Android (*Froyo*). Es por ello que cualquier terminal con un software anterior a estos no será compatible con el proyecto.

⁴ Application Programming Interface: Conjunto de métodos ofrecidos por otro software como una capa de abstracción.

⁵ Cocoa Touch: Conjunto de frameworks que permiten el desarrollo de aplicaciones nativas para iOS.

⁶ Push : tecnología de comunicación donde la petición de transacción se origina en el servidor.

Bases de datos y CMS

Tal y como hemos indicado previamente en el apartado **¡Error! No se encuentra el origen de la referencia.**, necesitamos un sistema de control de usuarios, para permitir acceder, modificar o borrarlos de una manera directa y simple sin necesidad de acceder directamente a la base de datos. Existen diversos sistemas de gestión de contenido, como *Joomla*, *Drupal* o *WordPress*. Sin embargo decidimos usar *Drupal* por modularidad y conocimientos previos ya adquiridos durante proyectos anteriores. *Drupal* funciona en cualquier plataforma que soporta tanto un servidor web capaz de ejecutar PHP (como Apache o lighttpd) y una base de datos (como *MySQL* o *Microsoft SQL Server*) para almacenar el contenido. Por lo tanto, hemos usado el propio *MacOSX* usado para el desarrollo del software móvil como plataforma para el desarrollo del CMS. Posteriormente se trasladó el desarrollo ya finalizado a un servidor externo.

Drupal requiere de PHP 4.4.0 o superior. La mayoría de entornos de desarrollo soportan PHP, y en nuestro caso particular hemos usado Netbeans.

En cuanto a la base de datos, se optó por *MySQL*, que es la opción utilizada por *Drupal* por defecto. *MySQL* es usada en muchos proyectos, incluido gran parte de los realizados previamente a cargo personal antes del presente proyecto. Además existe mucha documentación existente en Internet que facilita su gestión y mantenimiento. Para gestionar la base de datos de manera directa, usamos o bien los comandos del propio *mysql*, o bien *MySQLAdmin*.

En el entorno local, bajo Mac, usamos MAMP, que es un conjunto de programas para *MacOSX* que incluye *MySQL*, PHP y Apache.

Servidor de repetición

Como hemos dicho en el apartado **¡Error! No se encuentra el origen de la referencia.**, necesitamos un modo de distribuir los mensajes entre los clientes, ya que la conexión directa no es posible actualmente a través de redes de datos. Es por ello que se necesitó un sistema accesible desde el exterior (servidor público) aunque para las tareas de testeo y depuración se usó un sistema local. La aplicación encargada de repetir los mensajes se realizó en C++, bajo el entorno de *Netbeans*, y es necesario para su ejecución cualquier sistema que compile bajo gcc. Bajo entorno local se usó MacOSX para testear, y se lanzó públicamente bajo un entorno Linux corriendo en Ubuntu 11.10.



3. Diseño e implementación

En el presente capítulo vamos a explicar cómo se ha confeccionado todo el proyecto, detallando cada sistema usado y como se han interconectado. Separaremos el presente capítulo en tres sub-apartados, correspondientes a las diferentes implementaciones del software móvil, el software en *Drupal* y el servidor de repetición.

Para empezar, decidimos el nombre del proyecto. Durante un largo plazo de desarrollo, se usó el término **Compass** (brújula en inglés) para hacer referencia al proyecto, debido a la similitud con ésta. Sin embargo, y obviamente dado que este término no hacía ninguna referencia real a la posibilidad del proyecto y tras varias jornadas de decisión, se optó por usar el nombre: **“Meet Me App”**, como juego de palabras de la expresión en inglés *“meet me up”*, que puede traducirse como “encuéntrame”.

Así pues, a lo largo del proyecto veremos referencias constantes a la palabra *compass*, presente en nombre de ficheros o métodos, pero la aplicación final es conocida como **“Meet Me App”**

Una vez aclarado este punto, vamos a explicar brevemente como están interconectados bajo un simple diagrama.

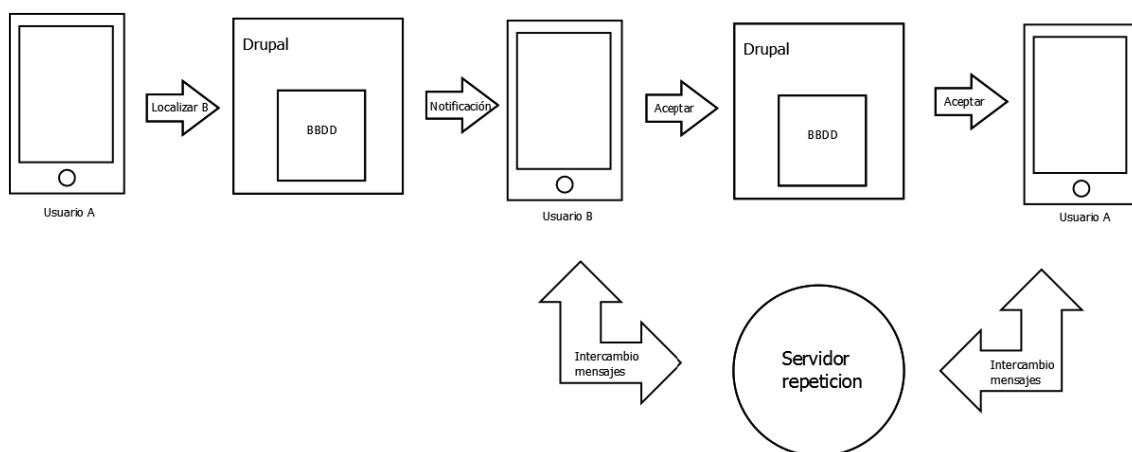


Ilustración 1 Diagrama general de funcionamiento del proyecto

En el esquema podemos observar el funcionamiento principal. Cuando un usuario, al que denominaremos *“Usuario A”* decide contactar con otro usuario, al cual



llamaremos “*Usuario B*” se suceden una serie de eventos que detallamos a continuación.

1. **Localizar B:** Cuando el “*Usuario A*” selecciona a un “*Usuario B*”, se envía una consulta hacia el servidor *Drupal*, que alberga la base de datos. En esta consulta se indica el número de teléfono del “*Usuario B*”. El “*Usuario A*” se queda esperando una respuesta.
2. **Notificar B:** Se realiza una consulta SQL en la base de datos para obtener el *deviceID*⁷ asociado a dicho número de teléfono. Este *deviceID* es necesario para enviar notificaciones *push* a los terminales. Así pues, se envía una notificación *push* al “*Usuario B*”.
3. **Aceptar:** La notificación *push* llega al “*Usuario B*” que al abrir la aplicación se le indicará que el “*Usuario A*” quiere contactar con él. Si acepta la petición, se notifica al servidor *Drupal* de ello y el “*Usuario B*” se conecta al servidor de repetición.
4. **Notificar A:** El servidor *Drupal* envía una notificación *push* al “*Usuario A*” indicándole que el “*Usuario B*” ha aceptado la petición. A partir de esta notificación, el “*Usuario A*” se conecta al servidor de repetición.
5. **Intercambio de mensajes:** Ambos usuarios están conectados al servidor de repetición. Cada uno de los usuarios escribirá su posición en el servidor y leerá del servidor la posición del otro usuario.

Una vez visto el esquema general de la aplicación vamos a detallar los diferentes diagramas de casos de uso.

⁷ DeviceId: es un identificador único del terminal. Es generado por Google/Apple según el SO.

3.1. Especificación de casos de uso

En este apartado vamos a especificar los casos de uso del usuario, de la aplicación y del servidor web.

3.1.1. Casos de uso relacionados con las acciones del usuario.

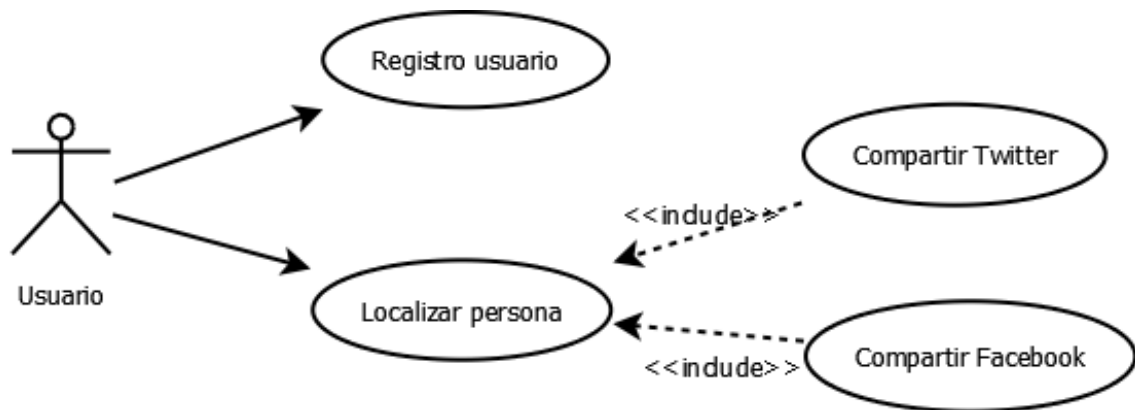


Ilustración 2 Caso de uso de las acciones del usuario

Registro usuario

Resumen de la funcionalidad: El usuario se registra en la web y se le permite contactar con sus contactos.

Actores: Usuario

Casos de uso relacionados: Ninguno

Precondición: Ninguno.

Postcondición: El usuario es insertado.

Proceso normal principal:

1. El usuario introduce sus datos de registro.
2. El usuario envía los credenciales a la web.
3. La web devuelve si el registro ha sido válido.

Alternativas de proceso y excepciones:

- 1a. La aplicación devuelve un error si el usuario no introduce sus datos o estos no son válidos.
- 2a. La aplicación devuelve un error si el usuario no dispone de conexión.

Localizar persona

Resumen de la funcionalidad: El usuario envía una solicitud de localización a un contacto determinado a través del servidor web *Drupal*.

Actores: Usuario

Casos de uso relacionados: Notificar usuario.

Precondición: El usuario está registrado.

Postcondición: El usuario entra en la pantalla de búsqueda.

Proceso normal principal:

1. El Usuario selecciona un usuario de su lista de contactos para localizar.
2. La aplicación envía al servidor web los datos de ambos usuarios.
3. El servidor web comprueba que ambos contactos sean válidos y estén disponibles.
4. El servidor web envía una notificación push al usuario destino.

Alternativas de proceso y excepciones:

- 1a. La aplicación deniega la solicitud si el usuario no dispone de conexión a Internet.
- 3a. El servidor web devuelve un error si el usuario destino está ocupado.

3.1.2. Casos de uso relacionados con la aplicación

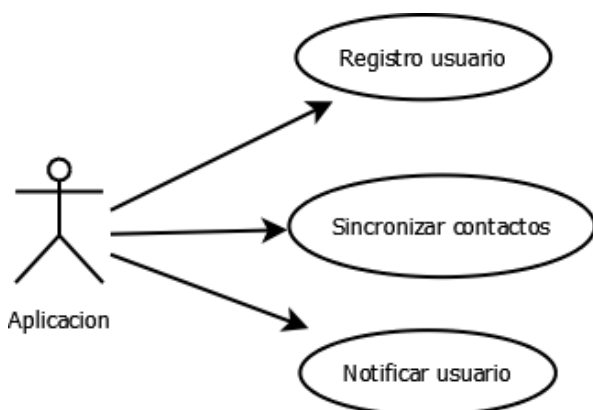


Ilustración 3 Casos de uso de la aplicación

Registro usuario

Resumen de la funcionalidad: La aplicación entra en modo registro cuando el usuario no está registrado.

Actores: Aplicación

Precondición: Ninguna

Postcondición: Ninguna

Proceso normal principal:

- 1.- La aplicación comprueba si existen datos guardados del usuario
- 2.- La aplicación muestra la pantalla de registro.

Sincronizar contactos

Resumen de la funcionalidad: La aplicación envía los contactos del usuario al servidor.

Actores: Aplicación

Precondición: El usuario está registrado

Postcondición: Ninguna

- 1.- La aplicación envía al servidor web todos los contactos del usuario.
- 2.- El servidor web filtra aquellos que estén registrados y los devuelve.
- 3.- La aplicación actualiza sus datos con los datos recibidos.

Alternativa de proceso y excepciones:

- 1a.- El caso de uso termina si el usuario no dispone de conexión a Internet.

Notificar al usuario

Resumen de la funcionalidad: La aplicación muestra una alerta cuando alguien ha solicitado contactar con el usuario actual.

Precondición: El usuario dispone de conexión a Internet y recibe una notificación push.

Postcondición: Ninguna

Proceso normal principal

- 1.- La aplicación comprueba el mensaje de la notificación push.
- 2.- La aplicación muestra una alerta indicándole al usuario que alguien quiere contactar.

Alternativas de proceso y excepciones:

2a. La aplicación cambia la interfaz de la alerta en caso de que el usuario esté en la pantalla de búsqueda por otro usuario.

2a. La notificación es descartada en caso de que el usuario esté en la pantalla de búsqueda esperando conectar con el usuario que envió la notificación.

3.1.3. Casos de uso servidor web

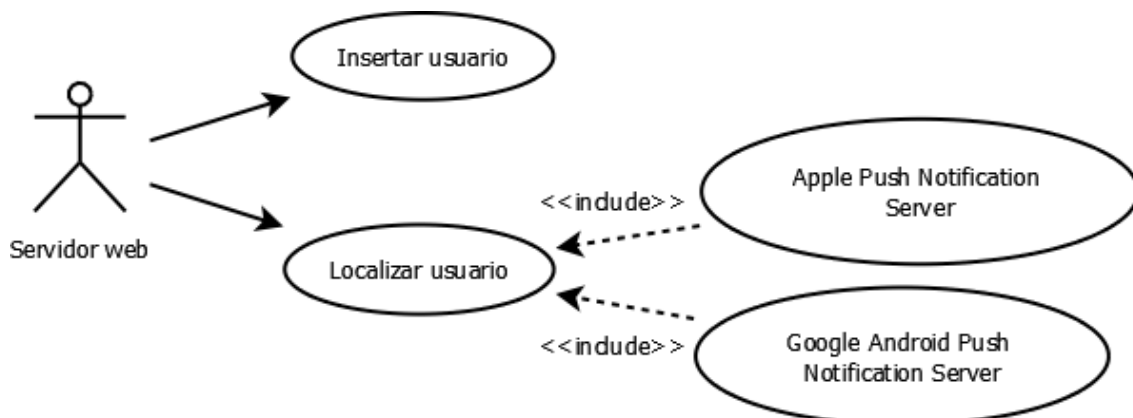


Ilustración 4 Casos de uso del servidor web

Insertar usuario

Resumen de la funcionalidad: La base de datos inserta o reemplaza la tupla de un usuario.

Autor: Servidor web

Precondición: El servidor web recibe una solicitud de registro.

Postcondición: Ninguna

Proceso normal principal:

- 1.- El servidor web comprueba los datos del usuario.
- 2.- El servidor web inserta o actualiza su base de datos.

Alternativas de proceso y excepciones:

- 1a. La aplicación devuelve error si faltan datos.

Localizar usuario

Resumen de la funcionalidad: Se actualiza el estado de los usuarios y se envían notificaciones push a través de servidores externos

Autor: Servidor web

Precondición: El servidor web recibe una solicitud de localización

Postcondición: Ninguna

Proceso normal principal:

- 2.- El servidor web comprueba que ambos usuarios estén disponibles
- 3.- El servidor web actualiza sus estados indicándolos como ocupados.
- 4.- El servidor web envía una notificación push a través del servicio requerido

Alternativas de proceso y excepciones:

2a.- El servidor web devuelve error si faltan datos
2a.- El servidor web devuelve error si al menos uno de los usuarios está ocupado.

Apple Push Notification Server

Resumen de la funcionalidad: Se envía una solicitud de envío de notificación a través de los servidores de Apple.

Autor: Servidor web

Precondición: Se ha recibido una localización cuyo destino es un terminal de Apple.

Postcondición: Ninguna

Proceso normal principal:

- 1.- Se obtiene el nombre del registro asociado al teléfono destino.
- 2.- Se encriptan los datos
- 3.- Se envía la petición.

Alternativas de proceso y excepciones:

- 1a.- La solicitud es cancelada si el teléfono destino no existe.

Google Push Notification Server

Resumen de la funcionalidad: Se envía una solicitud de envío de notificación a través de los servidores de Google.

Autor: Servidor web

Precondición: Se ha recibido una localización cuyo destino es un terminal de Android.

Postcondición: Ninguna

Proceso normal principal:

- 1.- Se obtiene el nombre del registro asociado al teléfono destino.
- 2.- Se encriptan los datos
- 3.- Se envía la petición.

Alternativas de proceso y excepciones:

- 1a.- La solicitud es cancelada si el teléfono destino no existe.

3.2. Diseño e implementación del entorno

Drupal

Antes de empezar el desarrollo del software móvil, es necesario disponer de una base de datos y de una serie de métodos que nos permitan realizar todas las interacciones entre el móvil y la base de datos online.

Si bien es posible ofrecer la base de datos sin más, hemos preferido hacer uso del CMS *Drupal*, para facilitar el testeo y depuración del sistema. *Drupal* es un gestor de contenido modular y muy configurable, además de dependiente casi por completo de una base de datos *Drupal* es de código abierto, con licencia *GNU/GPL* y puede descargarse desde su web.

NOTA: A fecha del proyecto existen dos ramas de *Drupal*, la 6.x y la 7.x. La versión más reciente incluye muchas novedades y características que no son útiles para nuestro proyecto, por lo que hemos usado la versión 6.x al ser la más extendida, probada y estable.

El proceso de instalación es muy sencillo y muy metódico, por lo que se omite en la presente documentación. Se pueden encontrar diversos tutoriales de inicio a *Drupal* tanto en su web como en centenares de sitios especializados.

Drupal gestiona sus contenidos a través de objetos denominados “nodos”. Estos nodos ofrecen una serie de propiedades, como identificador, fecha de creación, permisos, descripción...contenidos relacionados principalmente con la publicación de noticias, artículos, encuestas...en la web. Así pues, no nos ofrece actualmente una funcionalidad para nosotros. Necesitamos una manera de **almacenar los usuarios en la base de datos** que *Drupal* nos ha creado y una manera de **comunicarnos con dichos usuarios**.

Dado que no disponemos de una funcionalidad propia de *Drupal* que nos sirva de utilidad en la tarea de la gestión de usuarios, creamos nuestro propio código, al cual se le denomina “módulo” en *Drupal*, y que nosotros llamaremos **compass**. Estos módulos son software que extiende las funcionalidades y/o características de *Drupal*.

Este módulo, el cual es desarrollado en PHP, debe contener tanto la creación de nuestras propias tablas, así como de la funcionalidad de estas.

Antes de continuar con el diseño e implementación de nuestro módulo conviene mencionar que *Drupal* no ofrece por defecto servicios al exterior. Está pensado para ofrecer contenido a las propias páginas web sobre las cuales trabaja. Pero nosotros queremos que *Drupal* nos ofrezca contenido a los terminales móviles. Por suerte, *Drupal* es de código abierto y muy personalizable.

Drupal está formado por dos partes básicas: *DrupalCore*, el cual es el corazón de *Drupal* y ofrece lo básico, y los módulos externos. Los módulos permiten aumentar las funcionalidades de *Drupal* y muchas veces son contribuidos por la comunidad. Son de código abierto, de libre distribución y generalmente gratuitos. Se instalan de

una manera muy sencilla y están disponibles desde la página web:
<http://drupal.org/project/modules>

Nosotros hemos realizado ambas tareas: aumentar la funcionalidad de Drupal obteniendo módulos ya existentes, y a su vez, realizar los nuestros propios.

Para obtener funcionalidad con el exterior, necesitaremos el módulo *Services* <http://drupal.org/project/services> . Este módulo nos permitirá integrar nuestra aplicación en Drupal mediante llamadas realizadas por diversos protocolos, como REST, XMLRPC o JSON. Nosotros utilizamos XMLRPC aunque se detallará con detenimiento en el apartado Módulo *Services*.

Una vez aclarado este punto vamos a continuar con el desarrollo de nuestro módulo que trabajará directamente sobre el módulo *Services* mencionado.

3.2.1. Diseño

Cada módulo debe contener los siguientes ficheros:

- **install:** Debe definir la creación y destrucción, así como actualizaciones, del esquema de la base de datos relacionado con la instalación, desinstalación o actualización del módulo.
- **info:** Debe definir el nombre del módulo, una descripción y las dependencias del módulo
- **module:** Define la funcionalidad y métodos del módulo.

Fichero info

El fichero info es un simple texto plano en el cual se almacenan metadatos relacionados con temas y módulos. En él se describe tanto el nombre del módulo que será visible en la administración vía web de Drupal, como una descripción de éste, además de requisitos y otras funcionalidades.

Si bien rellenarlo correctamente no es obligatorio, sí es recomendable de cara a la documentación de cualquier proyecto que se realice.

Los campos que nosotros vamos a rellenar son los siguientes:

name (Obligatorio)

Es el nombre del módulo. En nuestro caso será *compass*.

description (Obligatorio)

Una descripción de las tareas realizadas por el módulo

core (Obligatorio)

Indica la versión de Drupal compatible con el módulo. En la fecha de creación del proyecto existen dos ramas, la 6.x (Drupal 6) y la 7.x (Drupal 7). Nosotros como hemos indicado previamente, usamos Drupal 6.

dependencies (Opcional)

Indica aquellos paquetes de los cuales depende el módulo. De esta manera, en caso de activar el actual módulo pero alguno de los paquetes dependientes no lo está, Drupal notifica al usuario.

package (Opcional)

Indicamos sobre qué rama dentro de la administración web de Drupal debe “colgar” el módulo.

Así pues, rellenamos los campos de la siguiente manera

```
name = "Compass Service"
description = "Services module for compass web management application"
package = "Services - services"
dependencies[] = services
core = "6.x"
```


Fichero install

En el fichero *install* se especifica aquellos procesos o requerimientos necesarios por un módulo. El fichero *install* se ejecuta la primera vez que un módulo es activado y se suele usar principalmente para crear tablas en la base de datos, o para actualizar las mismas cuando una nueva versión del módulo lo requiere. Podemos resumirlo como el **diseño de nuestra aplicación web**.

Así pues, vamos a definir en el fichero *install* el actual esquema de la base de datos, que podemos visualizar en el siguiente diagrama

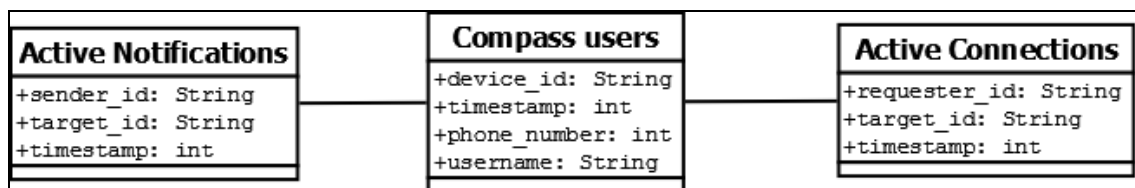


Ilustración 5 Diagrama de la base de datos

Es necesario un control de los usuarios de la aplicación, tanto por labores de gestión de usuarios como por imposición. Una parte indispensable de la aplicación es informar a los usuarios cuando alguien quiere localizarles. Dado que los usuarios no pueden comunicarse entre ellos directamente, *Drupal* actúa como mensajero entre ellos. Es por ello que necesitamos saber remitente y destinatario.

La tabla de usuarios contendrá tuplas referentes a información propia de cada uno. La información que guardamos es la siguiente:

- **Número de teléfono:** El número de teléfono completo (con prefijo) nos servirá como identificador único de cada usuario.
- **Identificador del terminal:** Es vital para poder enviar notificaciones push.
- **Nombre:** Simple formalismo.
- **Tipo de SO:** Para saber sobre qué servicio enviamos las notificaciones push (recordemos iOS y Android).
- **Timestamp:** Fecha de creación de la tupla en formato tiempo universal de Unix.

Podemos definir la tabla en PHP de la siguiente manera

```
function compass_service_schema() {
    $schema = array();
    $schema['compass_users'] = array(
        'description' => 'This table stores the user data for the APP to notify users
when new nodes are added.',
        'fields' => array(
            'deviceID' => array(
                'type' => 'varchar',
                'length' => 255,
                'not null' => TRUE,
                'description' => 'The identificator of the deviceID.'
            ),
            'timestamp' => array(
                'type' => 'int',
                'unsigned' => TRUE,
                'not null' => TRUE,
```

```
'description' => 'Date of creation.',
),
'phone_number' => array(
  'type' => 'varchar',
  'length' => 80,
  'not null' => TRUE,
  'description' => 'User phone number'
),
'user_name' => array(
  'type' => 'varchar',
  'length' => 80,
  'not null' => TRUE,
  'description' => 'User name. Used to send senders\' name through push
notifications',
),
),
'primary key' => array('deviceID')
);
```



Si bien hay que destacar que la implementación de ambos servidores *push* es similar, sí hay diferencias respecto a sus características. Así pues, el *deviceID* de un dispositivo y aplicación bajo iOS es un número de 32 bytes que puede ser representado por 64 dígitos hexadecimales. En *Android*, sin embargo, el *deviceID* puede contener letras y dígitos hasta un total de 255. Además el *deviceID* en *Android* no es fijo y Google puede actualizar el *deviceID* asociado a la dupla móvil/aplicación. Es por ello que es recomendable solicitar el *deviceID* periódicamente. Además es obligatorio tener acceso a *Google Play*⁸ y por lo tanto un cuenta de Google registrada.

Por otra parte, necesitaremos dos tablas más, una que nos indique las notificaciones pendientes que tenga cada usuario, y otra con las conexiones activas de cada usuario. Llamando a la primera tabla como *active_notif*, podemos describir su contenido de la siguiente manera

- *sender_id*: El *deviceID* del usuario que ha enviado la notificación.
- *target_id*: El *deviceID* del usuario que recibe la notificación.
- *timestamp*: La fecha de envío de la notificación en formato universal de Unix.

Esta tabla la usamos cada vez que el usuario quiere comprobar si tiene notificaciones de localización pendientes.

```
$schema['active_notifications'] = array(
  'description' => 'This table stores info for active notifications so the
notification target can access info from the sender.',
  'fields' => array(
```

⁸ Google Play: El servicio de distribución de aplicaciones de Android. Anteriormente conocido como Android Market.

```

'sender_id' => array(
  'type' => 'varchar',
  'length' => 255,
  'not null' => TRUE,
  'description' => 'Sender Device id.'
),
'target_id' => array(
  'type' => 'varchar',
  'length' => 255,
  'not null' => TRUE,
  'description' => 'Target device id.'
),
'timestamp' => array(
  'type' => 'int',
  'unsigned' => TRUE,
  'not null' => TRUE,
  'description' => 'sending timestamp.',
),
),
'primary key' => array('sender_id')
);

```

Por último, la tabla *active_conn* podemos resumirla de la siguiente manera:

- *requester_id*: El *deviceID* del usuario que inició la conexión.
- *target_id*: El *deviceID* del usuario que aceptó la conexión.
- *timestamp*: La fecha de conexión entre ambos usuarios en formato tiempo de universal de Unix.

Esta tabla nos es de utilidad para determinar si un usuario está siendo localizado (o localizando a otro) para determinar al resto de usuarios si pueden localizarle.

```

$schema['active_connections'] = array(
  'description' => 'This table stores info for active connections so we can
query for availability.',
  'fields' => array(
    'requester_id' => array(
      'type' => 'varchar',
      'length' => 255,
      'not null' => TRUE,
      'description' => 'Requester Device id.'
    ),
    'target_id' => array(
      'type' => 'varchar',
      'length' => 255,
      'not null' => TRUE,
      'description' => 'Target device id.'
    ),
    'timestamp' => array(
      'type' => 'int',
      'unsigned' => TRUE,
      'not null' => TRUE,
      'description' => 'connection creation timestamp.',
    ),
  ),
  'primary key' => array('requester_id')
);

```



3.2.2. Implementación

La implementación de nuestro modulo se especifica en el fichero *module*. En él indicaremos mediante lenguaje en PHP, qué métodos existen, qué argumentos reciben y la funcionalidad en sí de cada método.

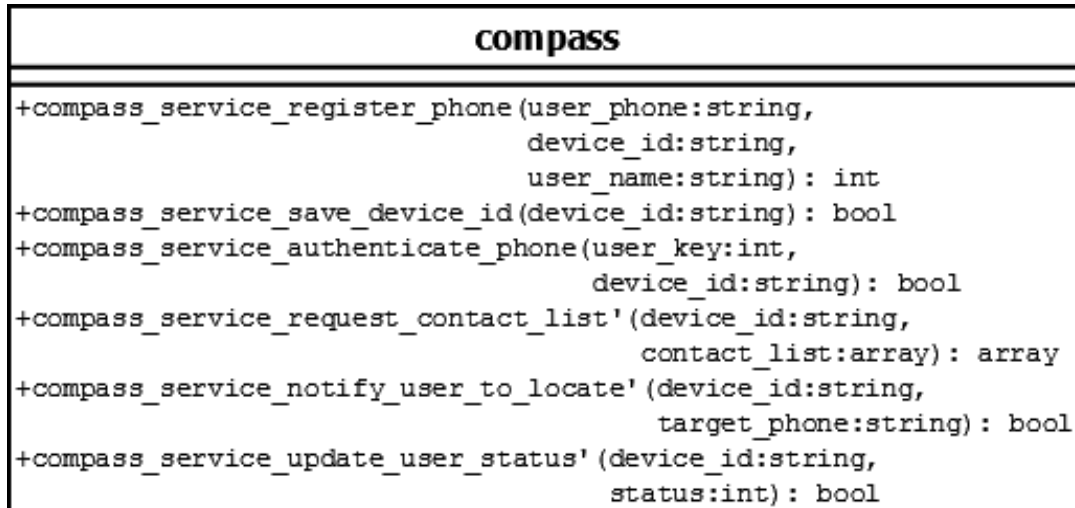


Ilustración 6 Diagrama UML de clases implementadas

Para ello vamos a definir previamente los diagramas de flujo relacionados con los casos de uso vistos anteriormente:

Registro de usuario

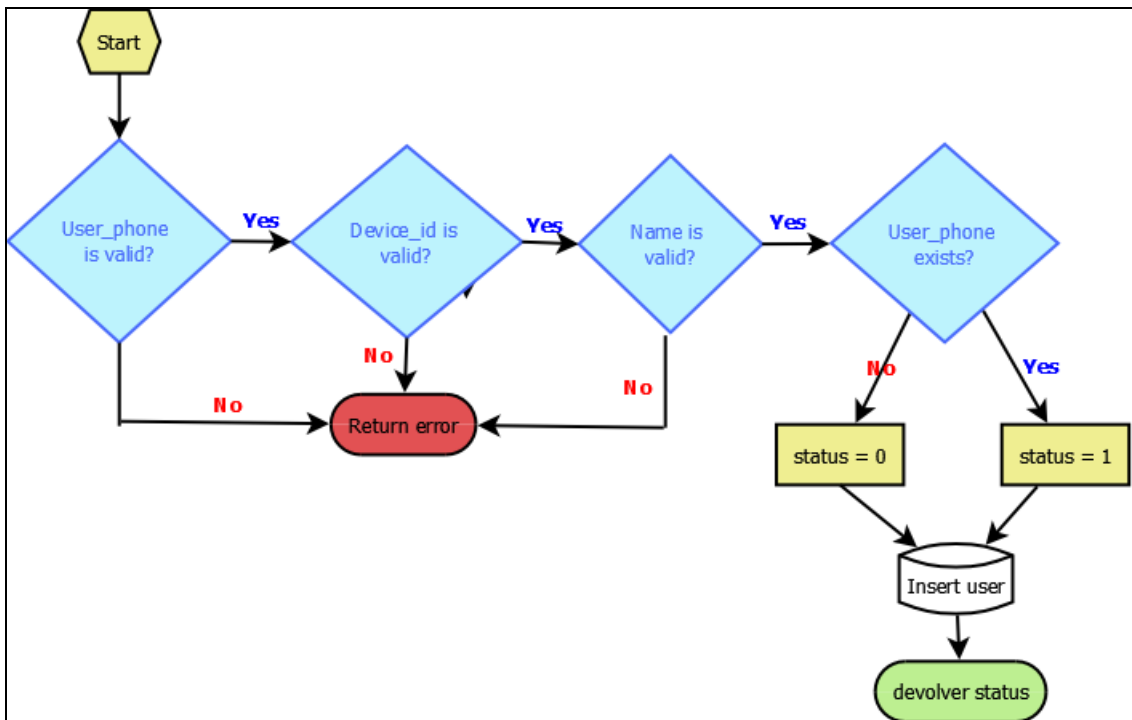


Ilustración 7 El proceso de registro del usuario en el servidor web

La lógica detrás del diagrama es la siguiente. Una vez la solicitud de registro llega al servidor de Drupal, se comprueba que no falten datos o que no sean válidos. Diferenciamos el valor devuelto según sea un registro nuevo o no. Esta situación es añadida debido a que el usuario puede borrar la aplicación o sus datos asociados, por lo que cuando ejecute la aplicación por primera vez aparecerá la pantalla de registro (al no haber datos guardados localmente), pero su número ya existe en nuestra base de datos.

Denominamos a este método dentro de nuestra API como `compass.registerPhone`.

Sincronización contactos

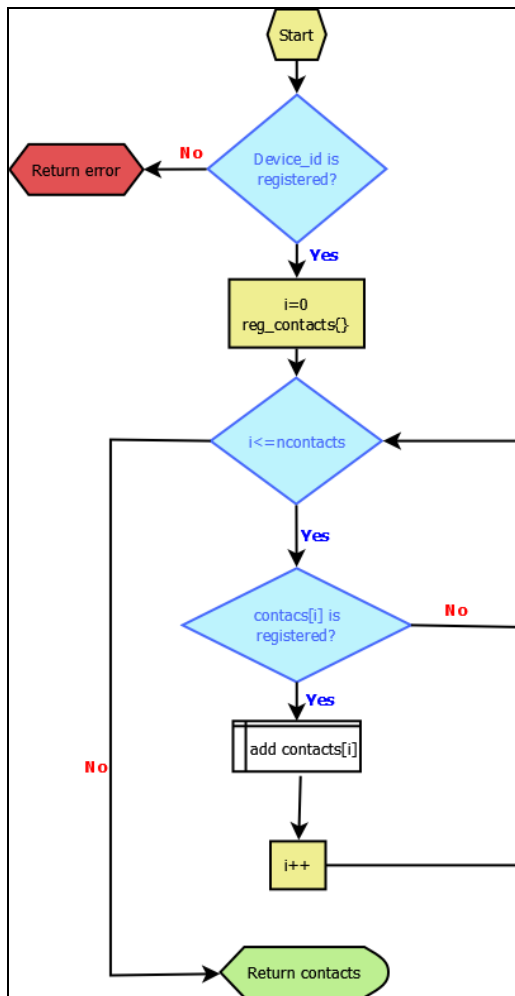


Ilustración 8 Obtención de contactos válidos en el servidor web

Cuando Drupal recibe una solicitud de sincronización de contactos, recibe el identificador del dispositivo así como una lista con los teléfonos de todos sus contactos. El servidor filtrará por aquellos que existen en su base de datos (y por lo tanto usan la aplicación) y estos serán devueltos al usuario.

Denominamos a este método dentro de nuestra API como `compass.requestContactList`.

Localizar usuario

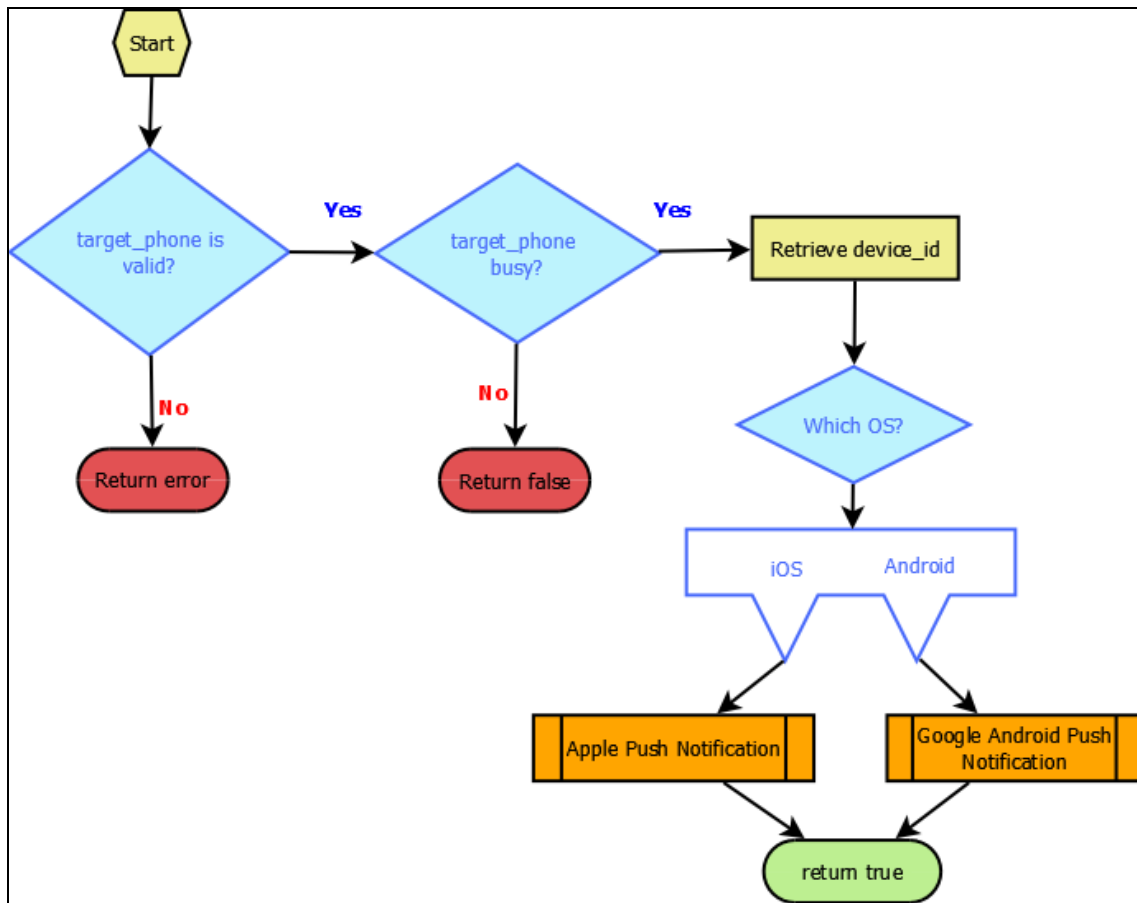


Ilustración 9 Cada vez que hay una solicitud, comprobamos la validez del usuario destino

En cuanto a la localización de usuarios, Drupal comprueba que el teléfono destino no esté ocupado o no sea válido, y en caso contrario comprueba su SO asociado y según éste, enviará una notificación push a través del servicio referenciado. En nuestra API es denominada como `compass.notifyTargetUser`.

3.2.3. Módulo Services

Tal y como hemos detallado previamente, nuestro módulo trabaja directamente sobre *Services*, un módulo de la comunidad de Drupal que se encarga de establecer una solución para integrar nuestros métodos de Drupal en servicios externos.

Dentro del apartado de dicho módulo podremos acceder a servicios como comprobar la funcionalidad de los métodos a través de una consola, u otros como configurar la seguridad.

En el primer caso nos permite realizar llamadas directamente a nuestro módulo desde dentro de la propia web y comprobar que su funcionamiento es correcto. **Las llamadas son reales** y accederán y escribirán en la base de datos si fuera necesario. No se trata de un servicio de *testing*. Para ellos existen otras alternativas como *SimpleTest*⁹.

En la siguiente imagen podremos ver un ejemplo de uso de la consola, donde accedemos al método encargado de obtener los contactos del usuario que pueden ser localizados y que, como hemos visto en el diseño, hemos denominado **compass.requestContactList**

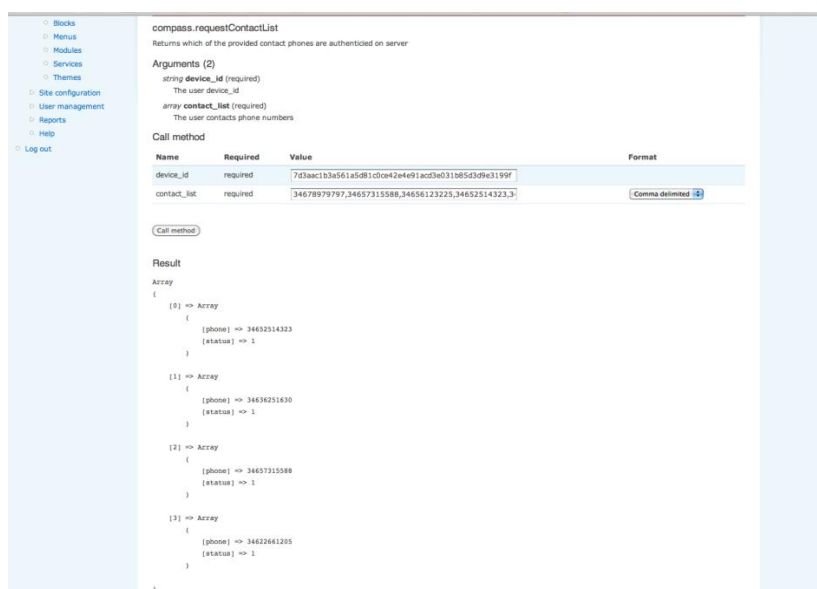


Ilustración 10 Un ejemplo de uso del método para obtener contactos desde la consola de Drupal

Otra parte de la cual se encarga el módulo *Services* es la relacionada con la seguridad. Si bien nuestra web en Drupal es accesible desde el exterior, configurando correctamente el módulo *Services* también conseguiremos esa accesibilidad. Sin embargo, al igual que no permitimos el acceso a nuestra base de datos directamente desde la web salvo que se tengan dichos permisos, tampoco podemos ofrecer esa falta de seguridad para las aplicaciones externas que soliciten nuestros servicios.

La seguridad por defecto y que usaremos en nuestro módulo es **API key authentication**¹⁰. Para ello, son necesarios los siguientes parámetros en cada llamada:

⁹ <http://drupal.org/node/1128366>

¹⁰ <http://drupal.org/node/762092>

- *Timestamp* – La hora actual en formato Unix.
- *Domain* – El valor introducido en el campo dominio.
- *Nonce* – Un valor aleatorio.
- *Hash* – Un sha256 hash formado por el timestamp, el nonce y el nombre del método remoto usando la API key como clave compartido.

Como podemos comprobar, todos los valores son triviales excepto dos: el dominio y una clave.

Podremos especificar diferentes dominios con diferentes privilegios que solo tendrán acceso a aquellos módulos integrados con *Services*.

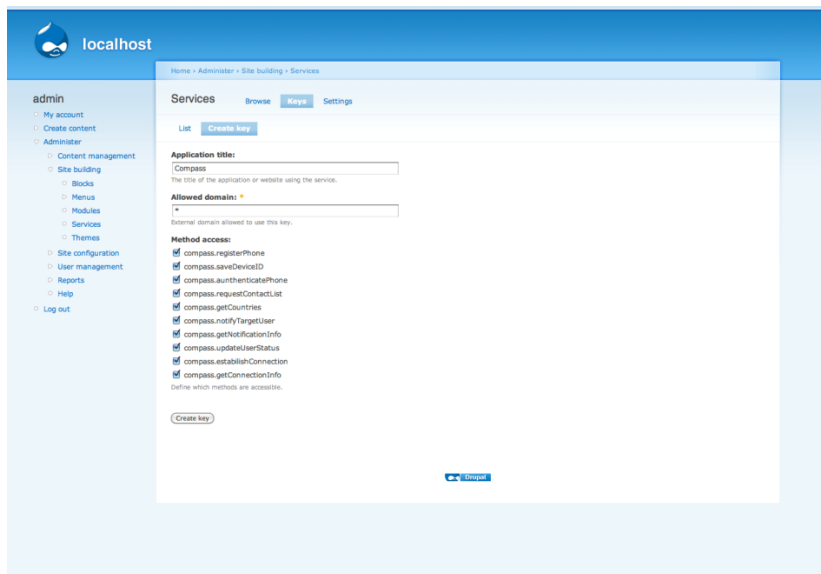


Ilustración 11 Podemos especificar un dominio y los métodos a los que podrá acceder

Una vez configurada correctamente, obtendremos una clave:

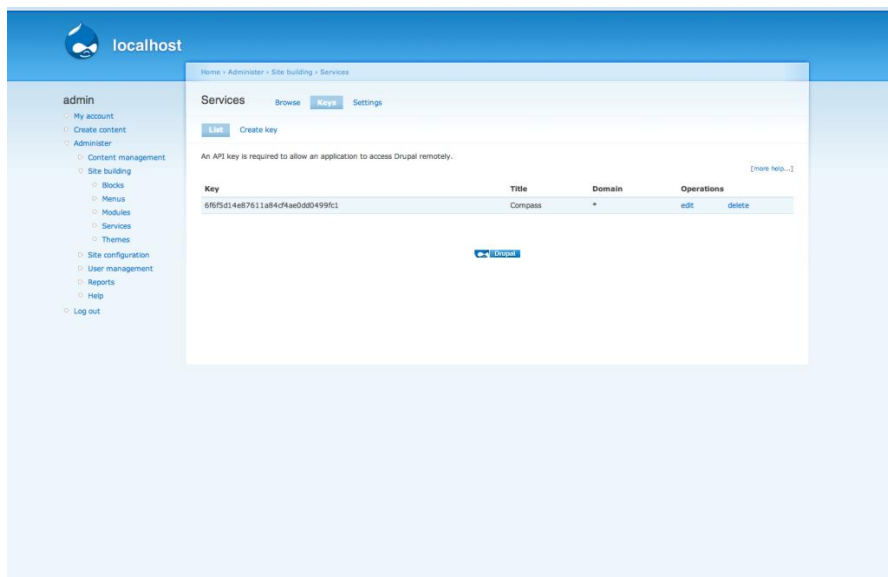


Ilustración 12 La clave generada será usada para acceder a la API

Con esta clave, que será compartida por aquellas aplicaciones que nosotros deseemos, nuestros métodos serán accesibles para dichas aplicaciones.

3.3. Diseño e implementación software móvil

En el presente capítulo vamos a definir y especificar algunas de las funciones presentes en la aplicación móvil, mencionando las diferencias y similitudes en las implementaciones de ambos lenguajes/SDK.

Primeramente vamos a comentar su diseño, el cual es común a ambas plataformas, para posteriormente hablar de su implementación. Dentro de la implementación vamos a comentar los siguientes apartados:

1. Interfaz
2. Localización
3. Conexiones de red

Como hemos mencionado en el apartado de Análisis, el desarrollo se ha realizado con el SDK 11 de *Google Android* y la SDK 5.0 de *Apple* para *iOS*. En futuras revisiones del SDK las implementaciones pueden cambiar o quedar obsoletas, pero a fecha del proyecto todas pertenecen a métodos soportados.

Tanto el desarrollo de software para *iOS* como para *Android* se basa en la arquitectura *MVC* (Modelo Vista Controlador), el cual es un patrón que separa los datos de la aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos.

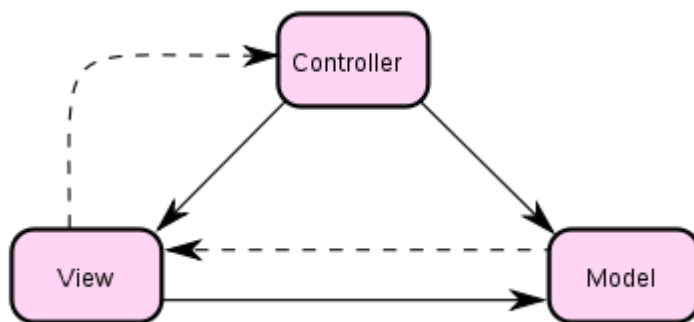


Ilustración 13 Esquema de modelo-vista-controlador

Es importante conocer este tipo de arquitectura para poder entender la implementación del proyecto, así que vamos a facilitar un ejemplo para entender mejor el concepto:

Supongamos que el usuario está en la pantalla de registro, la cual muestra varios campos a rellenar como el número de teléfono y el nombre, además de un botón de “Aceptar” para formalizar el registro. El usuario dispone de interacción con estos tres elementos.

El controlador por su parte, maneja todos los eventos de entrada de la interfaz de usuario y los convierte en una acción de usuario.

Cuando el controlador recibe un evento (por ejemplo el usuario ha dado al botón Aceptar), el controlador notifica al modelo de este cambio.

Desarrollo e implementación de software de localización en tiempo real para dispositivos móviles

La vista por su parte, obtiene los datos de dicho modelo (por ejemplo para comprobar si el registro es válido) y espera nuevas acciones del usuario, lo cual reinicia el ciclo.

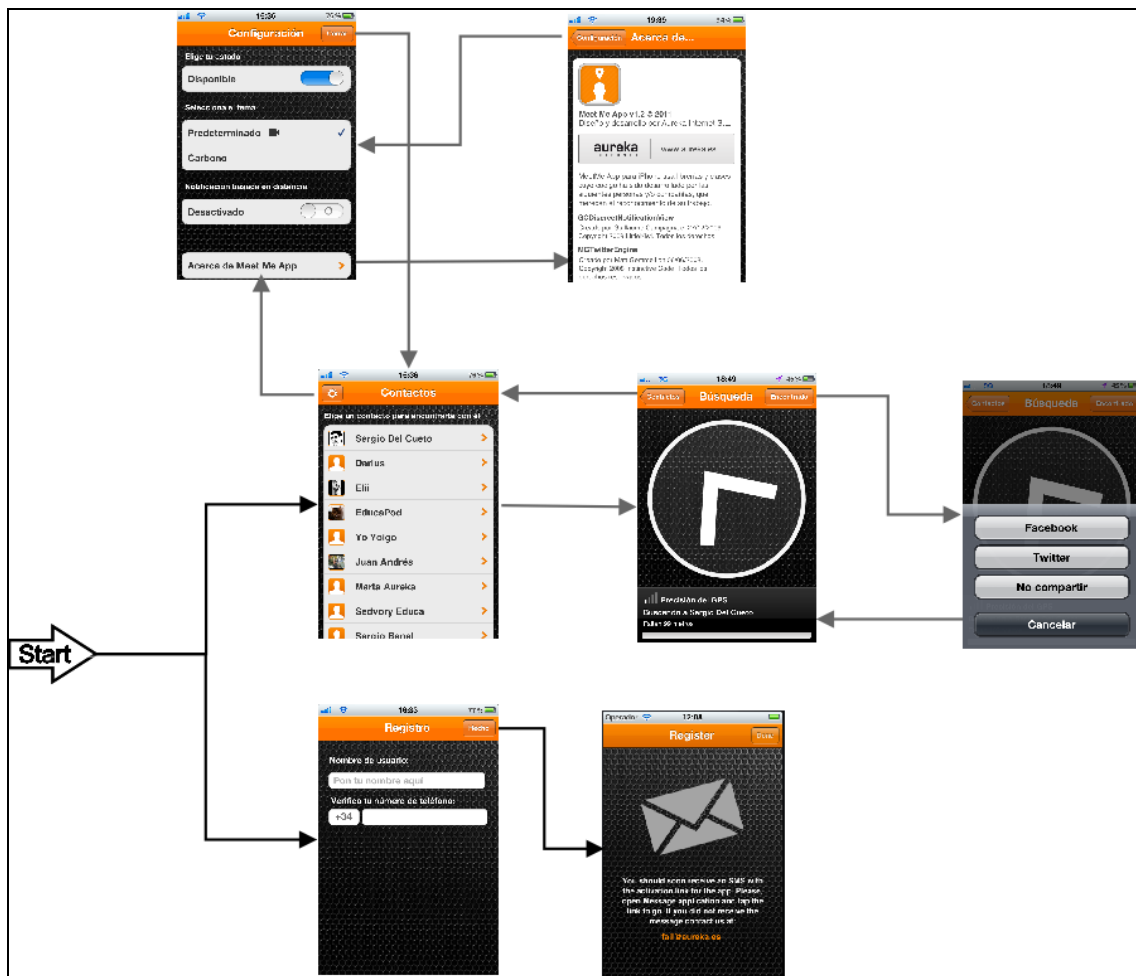


Ilustración 14 Storyboard de la aplicación

Visto la navegación, vamos a detallar las tres pantallas principales o que requieren mayor dificultad.

3.3.1. Registro del usuario

Tal y como hemos comentado en el ejemplo, la vista del registro usuario es una vista sencilla con una lógica detrás. Para determinar si es necesario el registro comprobamos si existen datos guardados de la aplicación dentro de los datos de usuario (*NSUserDefaults* en iOS, *SharedPreferences* en Android). Si no existen datos de registro es porque, o bien es la primera vez que se ejecuta la aplicación y por lo tanto no se ha registrado, o bien porque ya se había ejecutado con anterioridad pero sin embargo no se completó el registro.

El modelo encargado de ésta, deberá comprobar que los datos introducidos por el usuario son válidos, realizar la petición de registro en caso afirmativo y a su vez, notificar al usuario si existe algún problema durante dicho registro (por ejemplo, no hay conexión a Internet).

Podemos definir el diagrama de clases correspondiente como sigue (usando declaraciones en *Objective-C*).

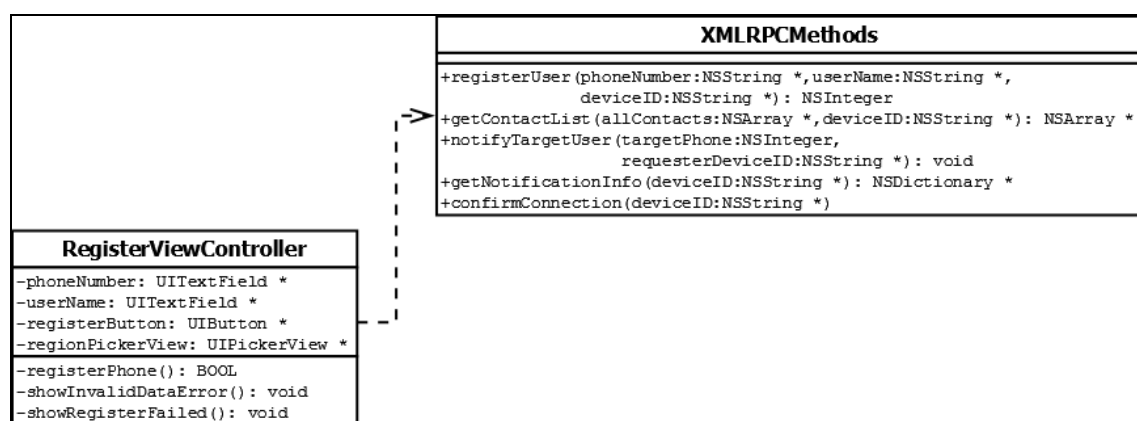


Ilustración 15 Diagrama UML del registro de usuario

El diagrama es bastante autoexplicativo. Disponemos de dos campos de texto, un botón de envío y un *UIPickerView*¹¹ con información relacionada respecto a los prefijos de los teléfonos. Al pulsar el botón de registro, comprobamos si los datos son válidos. En caso de ser válidos, realizamos una petición a *XMLRPCMethods*.

Por otra parte la vista se actualizará mostrando un nuevo estado solicitando un código de confirmación para completar el registro.

Una vez introducido dicho código, el registro estará completo y procederemos a la pantalla de selección de contacto.

NOTA: Como podemos comprobar en el diagrama, nuestra clase realizará llamadas (en este caso de registro) a una clase denominada *XMLRPCMethods*, la cual se encarga de todas las llamadas al servidor web y que explicaremos detalladamente dentro del apartado de conexiones de red. De momento, nos basta con saber que necesitaremos notificar a nuestro servidor web cuando registremos al usuario, para sincronizar contactos, solicitar una localización, obtener información extra de la notificación y para confirmar o denegar una solicitud de localización.

11

http://developer.apple.com/library/ios/#documentation/uikit/reference/UIPickerView_Class/Reference/UIPickerView.html





No existen diferencias entre las implementaciones Android y iOS. Sin embargo sí es conveniente mencionar la razón por la cual se pide el número de teléfono al usuario en vez de obtenerlo automáticamente. Y es que la razón es sencilla: ni Apple ni Google permiten en sus SDK acceder a este tipo de información de manera legal. No hay ninguna manera disponible en el SDK de obtener el número de teléfono del terminal. Sí es posible, en cambio, obtener todos los datos de los contactos de la agenda, pero no de uno propio.

3.3.2. Pantalla contactos

Esta pantalla o actividad se encarga de gran parte de la tarea de la aplicación. A través de diferentes implementaciones y clases subyacentes gestionará:

- Obtención de los contactos del usuario
- Sincronización de los contactos con el servidor
- Gestionar las notificaciones push entrantes

Obtención de los contactos de usuario

A continuación incluimos el diagrama encargado de obtener los contactos existentes del usuario que disponen de número de teléfono y que por lo tanto podrían haber instalado la aplicación.

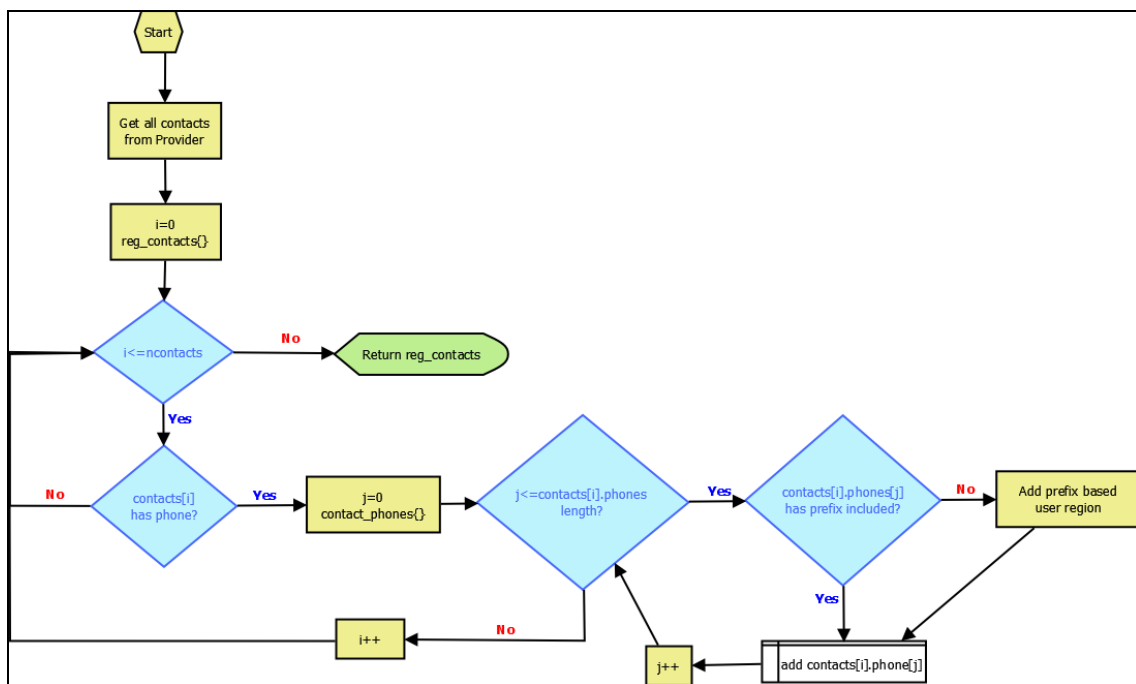


Ilustración 16 Diagrama de obtención de posibles contactos localizables

El diagrama es bastante autoexplicativo. Primero obtenemos la lista de contactos completa, luego obtenemos cada uno de los números de teléfono del contacto (un mismo contacto puede tener asociado varios números en la agenda) y consideramos la necesidad de añadir o no el prefijo del país (recordemos que cada usuario se registra con su número de teléfono incluyendo el prefijo).



La mayor diferencia entre ambas implementaciones es la manera de obtener los mismos datos. Mientras que en Android se obtienen los contactos mediante consultas a un proveedor estándar del sistema, en iOS tienes una librería disponible (AddressBook) para gestionar toda la obtención de contactos. También hay que mencionar que dicha librería trabaja a bajo nivel, por lo que todo código involucrado con este debe ser adaptador a CoreFramework.

Por últimos mencionar que en ambos casos hemos de lidiar con los prefijos manualmente. Ni Android ni iOS permiten obtener los contactos con/sin prefijos, y es la propia aplicación la que interpretar si el número obtenido incluye el prefijo del país, y en caso de no ser así, añadirlo manualmente.

Sincronización de los contactos de usuario

En este apartado detallamos como contactamos con el servidor web. El diseño es muy simple: una vez tengamos todos los números de teléfono de nuestros contactos locales, enviaremos dichos teléfonos a nuestro servidor web, el cual nos responderá con una lista filtrada la cual incluye solo los teléfonos que tienen instalada la aplicación y por lo tanto podemos contactar con ellos.

Este flujo es necesario repetirlo con asiduidad, ya que gracias a la multitarea de ambas plataformas no podemos solicitar una sincronización solo al inicio de la aplicación. Es por ello que debemos sincronizar cada cierto periodo, y en nuestro caso está definido a 30 segundos.

Dado que la mayor parte de la implementación corresponde a la interacción con el servidor web, definiremos ésta en su apartado correspondiente.

3.3.3. Interfaz

El diseño de la interfaz de la aplicación móvil es idéntico en ambas plataformas. Queríamos transmitir la misma experiencia de usuario indistintamente del terminal usado. Es por ello que ha sido necesario ajustar ciertos componentes gráficos que no son comunes en una plataforma en cuestión, pero sí en otra.

Nada más comenzar el desarrollo de la interfaz en Android y iOS las primeras diferencias saltan a la vista. Mientras que en iOS todo el desarrollo de la interfaz se puede hacer visualmente (esto es, puedes “diseñar” la interfaz mediante la colocación de las componentes visuales directamente en un editor), en Android esta posibilidad no existe. Como bien es sabido, existen centenares de diferentes dispositivos Android, por lo que no debemos especificar la posición o tamaño de los componentes en aspectos relativos, puesto que esos valores pueden no ajustarse al diseño que realmente queremos.



Ilustración 17 Pantalla de búsqueda en su versión iOS

Como podemos ver en la “Ilustración 17 Pantalla de búsqueda en su versión iOS”, hay diversas componentes visuales distintas, como el fondo, el círculo con la flecha o la vista inferior con información sobre la búsqueda. En iOS podríamos haber definido los valores de posición y tamaño mediante valores (por ejemplo, la vista inferior está en las coordenadas XY con un ancho y alto específico), o bien mediante valores relativos a la vista (el círculo con la flecha está centrado sobre la vista).

En Android la primera posibilidad es inadmisibles. Es posible, pero totalmente inadecuada. ¿Qué ocurriría si indicamos que el círculo es de un radio de 160px en diferentes pantallas con diferentes resoluciones? Si el terminal tiene una resolución baja, es posible que parte de la imagen no sea visible. Si el terminal tiene una resolución muy alta, podemos estar desaprovechando mucho espacio visual.

Es por ello que en Android las componentes visuales deben ser utilizadas respecto a **valores relativos**. Es decir, determinamos sus posiciones y tamaños en relación a otras componentes: una componente está situada encima de otra, una componente tiene un ancho igual al padre, ocupa toda la pantalla...

Y siendo aspectos relativos, ¿qué necesidad hay de usar un editor visual? La única necesidad es para comprobar que el diseño es correcto, pero toda la implementación la realizamos a través de ficheros *xml* denominados **layouts**.

NOTA: Los únicos valores totales admisibles en Android son aquellos relacionados con el tamaño de las fuentes y la densidad de la pantalla. La densidad de pantalla es independiente del tamaño de esta, por lo que no tendremos problemas, aunque no es recomendable el abuso de ésta, siendo usado marginalmente para delimitar márgenes o similares.

NOTA: En iOS también existen diferentes resoluciones según los dispositivos. A fecha de hoy existen cuatro soportadas, la del *iPhone* 3G y 3GS (480x320), la del *iPhone* 4 y 4S (960x640), la del *iPad* 1 y 2 (1024x768) y la del nuevo *iPad* (2048x1536). Sin embargo, la solución por parte de Apple ha sido genialmente abordada. Como se puede comprobar la resolución de los modelos nuevos es el

doble que la de los anteriores. Es por ello que siempre se mantiene el ratio de aspecto. Y a ojos del desarrollador no se diferencian ambos dispositivos. Todas las vistas tienen un tamaño predefinido de 480x320 o 1024x768 según hardware, y es el sistema operativo quien se encarga del reescalado. Dicho de otra manera, puedes situar un botón en la posición 240x160 (centra de la pantalla en iPhone 3G), que también se situará en el centro de la pantalla en las versiones posteriores. El único trabajo que ha de realizar el desarrollador es proporcionar imágenes en alta resolución (si fuera necesario), añadiendo el sufijo **@2x**. De esta manera, y nuevamente especificando el nombre de la imagen, el sistema determinará si debe usar la imagen en alta resolución o en baja según el dispositivo.



La diferencia es clara. No solo a ojos del desarrollador es más sencillo un editor visual que la combinación editor de texto más validador visual para comprobar lo mismo, sino que además las diferentes pantallas, resoluciones y densidades pueden suponer un inconveniente. Aun así, Google ha mejorado mucho en este aspecto, conscientes del inconveniente.

Además a ojos de un desarrollador iniciándose en el software para terminales móviles, un editor visual basado en modo texto puede resultar confuso e incluso carente de sentido.

En nuestra aplicación no dispusimos de ningún inconveniente a destacar a la hora de implementar la interfaz para Android, si bien sí fue claramente menos directo de implementar que en la plataforma contraria.

Por otra parte, también cabe mencionar que, si bien no era estrictamente necesario, hemos incorporado una componente visual de Android que es novedosa, en el sentido de que solo está presente en la SDK 11 y superior (*Android 3.0 Honeycomb*). Son las llamadas “*Action Bars*”, que ofrecen una barra superior similar a la barra de navegación de iOS que permite al usuario realizar gran parte de las acciones sin necesidad de acceder a los botones físicos de su terminal (en concreto los botones de Atrás y Menú).



Ilustración 18 Las nuevas Action Bar de Android

Tal y como hemos mencionado, solo está disponible a partir de terminales con Android 3.0 o superior, pero sin embargo nuestro proyecto está dirigido a partir de Froyo o superior (2.2). Es por ello que necesitamos una implementación externa para estos terminales.

Por suerte, disponemos de diversas librerías ofrecidas por la comunidad Android. Entre las opciones descubiertas, decidimos usar **Action Bar for Android de Johan Nilson**.

Así pues usaremos estas librerías externas para los terminales con un SO anterior al 3.0, y la implementación oficial por Google para el resto.

OpenGL

Junto a la aplicación se proporcionan dos temas distintos. Uno realizado con los *frameworks* de gráficos 2D (*CoreGraphics* o *Canvas*) y otra utilizando componentes visuales en 3D. Dado que resultan ser modelos sencillos, hemos optado por implementarlo en *OpenGL*. Como es sabido, *OpenGL* define una API multiplataforma por lo que el código usado en ambas versiones es idéntico y la única diferencia radica en cómo integrar OpenGL dentro del código nativo.

Hemos usado la variante especializada para dispositivos embebidos tales como smartphones, PDAs o videoconsolas: *OpenGL ES*. Si bien tanto Android como iOS son compatibles con *OpenGL ES 2.0*, hemos preferido utilizar la versión anterior, la 1.1 por simplicidad.

A continuación incluimos parte de la función de dibujado perteneciente a su versión Android e iOS

```
@Override
public void onDrawFrame(GL10 gl) {

    // TODO Auto-generated method stub

    /*
    * Usually, the first thing one might want to do is to clear
    * the screen. The most efficient way of doing this is to use
    * glClear().
    */
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    gl.glEnable(GL10.GL_BLEND);
    gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);

    /*
    * Now we're ready to draw some 3D objects
    */
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();

    gl.glScalef((2*scale)/5, (2*scale)/5, (2*scale)/5);
    gl.glRotatef(xRotation, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(zRotation, 0.0f, 0.0f, 1.0f);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

    // Draw the triangle
    gl.glColor4f(1.0f, 0.65f, 0.0f, 1.0f);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleVB);

    gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 3);

    gl.glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleVB);
    gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 1, 3);

    // Draw the circle
    gl.glTranslatef(0f, 0.01f, 0f);
    gl.glColor4f(0.0f, 0.0f, 0.0f, 0.5f);
    gl.glVertexPointer(2, GL10.GL_FLOAT, 0, circleVB);
    gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 361);

    gl.glTranslatef(0f, 0f, -0.01f);
    gl.glColor4f(1.0f, 0.65f, 0.0f, 1.0f);
    gl.glVertexPointer(2, GL10.GL_FLOAT, 0, top_circleVB);
    gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 361);

    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}
}
```

```

/**
 * Draws the compass.
 */
- (void) GLDrawCompass {

    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_POINTS);
    glLoadIdentity();

    glTranslatef(-0.01, -0.1, 0);
    glRotatef(_currentRotationX, 1, 0, 0);
    glRotatef(_currentRotationZ, 0, 0, 1);
    glScalef(0.36, 0.36, 0.36);

    // Draw the triangle
    glColor4f(1.0f, 0.65f, 0.0f, 1.0f);
    glVertexPointer(3, GL_FLOAT, 0, &compassArrow);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 3);

    glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
    glVertexPointer(3, GL_FLOAT, 0, &compassArrow);
    glDrawArrays(GL_TRIANGLE_FAN, 3, 6);

    glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
    glVertexPointer(3, GL_FLOAT, 0, &compassArrow);
    glDrawArrays(GL_TRIANGLE_FAN, 9, 12);

    glColor4f(1.0f, 0.65f, 0.0f, 1.0f);
    glVertexPointer(3, GL_FLOAT, 0, &compassArrow);
    glDrawArrays(GL_TRIANGLE_FAN, 6, 9);

    // Draw the circle
    glColor4f(0.0f, 0.0f, 0.0f, 0.3);
    glVertexPointer(2, GL_FLOAT, 0, &vertices);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 361);
    glTranslatef(0, 0, -0.01);

    glColor4f(1.0f, 0.65f, 0.0f, 1.0f);
    glVertexPointer(2, GL_FLOAT, 0, &top_vertices);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 361);

    glPopMatrix();
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_POINTS);
}

```

```
/* Main drawing view
 * Implements MSAA
 */
- (void)drawView {

    [EAGLContext setCurrentContext:context];

    if (!ready)
        glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
    else
        glBindFramebufferOES(GL_FRAMEBUFFER_OES, msaaFrameBuffer);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glOrthof(-1.0f, 1.0f, -1.0f/(backingWidth / backingHeight),
1.0f/(backingWidth/backingHeight), -1.0f, 1.0f);
    glViewport(0, 0, backingWidth, backingHeight);

    glMatrixMode(GL_MODELVIEW);

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    [self GLDrawCompass];
    if (msaaEnabled) {
        GLenum attachments[] = {GL_DEPTH_ATTACHMENT_OES};
        glDiscardFramebufferEXT(GL_READ_FRAMEBUFFER_APPLE, 1, attachments);
        // Bind both MSAA and View framebuffers
        glBindFramebufferOES(GL_READ_FRAMEBUFFER_APPLE, msaaFrameBuffer);
        glBindFramebufferOES(GL_DRAW_FRAMEBUFFER_APPLE, viewFramebuffer);
        // Call a resolve to combine both buffers
        glResolveMultisampleFramebufferAPPLE();
    }
}
```

Hemos incluido la función de inicialización de la versión iOS por una razón, y es para mostrar una implementación que Apple ofrece en su SDK: el *multisampling*¹². Sin embargo, el *multisampling* no es compatible con terminales anteriores a iPhone3GS, así que debemos descartar la implementación del *antialiasing* para dichos terminales.

Dado que nuestro modelo OpenGL es muy simple, no existen bajadas de rendimiento. Sin embargo, en modelados o escenas más complejas sería conveniente realizar *benchmarks* para justificar su uso.

¹² Técnica antialiasing para mejorar la calidad de imagen relacionada con los bordes.

3.3.4. Localización y orientación

En este apartado vamos a tratar todo lo relacionado con los servicios de geolocalización y sensores ofrecidos por Google y Apple en sus respectivos SDK.

Mediante la localización, podemos obtener diversos parámetros relacionados con la posición del terminal, tales como su latitud y longitud, precisión de los datos, tipo de localización (GPS o A-GPS), velocidad de movimiento, altitud... Mediante un servicio conocido como *Reverse Geocoding* que “traduce” la latitud y longitud podemos incluso saber una dirección física (como por ejemplo una calle).

A pesar de la cantidad de servicios disponibles, nosotros solo necesitaremos utilizar los siguientes:

- Longitud y latitud: Mediante estos dos valores podremos precisar la distancia existente entre la posición del terminal y cualquier otra posición (en nuestro caso sería el otro usuario).
- Precisión de los datos: Expresada en metros, nos indica cuan preciso son los valores obtenidos por el sensor de localización. A menor valor, mayor precisión. Mostraremos un indicativo visual al usuario respecto a este valor.

En cuanto a la orientación, nos referimos a orientación a conocer la dirección respecto a los polos magnéticos las cuales es apuntado el terminal, y además, conocer la inclinación del terminal sobre el suelo. Estos dos valores nos permitirán ajustar la dirección y rotación de la componente visual que indica la dirección que el usuario debe tomar.

- La orientación se obtiene usando principalmente el magnetómetro (brújula). Mediante dicho sensor podemos conocer la posición respecto a los ejes del terminal, de manera que si, el usuario conserva su posición geográfica (latitud y longitud) pero rota sobre sí mismo, la aplicación sea capaz de adecuarse al cambio y seguir manteniendo una dirección destino correcto.
- La rotación la obtenemos mediante el giroscopio y nos permite saber el eje de inclinación sobre el suelo. De esta manera, si el usuario inclina su terminal, el modelo gráfico encargado de mostrar la dirección sea capaz de mostrar esa misma inclinación.

Así pues, en nuestra aplicación móvil usamos tres sensores: la localización, el magnetómetro y la rotación. Vamos a detallar cada uno de los sensores, con especial hincapié al primero de ellos, dado que es la base de todo. Una mala implementación conllevará a una mala experiencia de uso en el principal aspecto de la aplicación: localizar.



Localización

La localización tanto en Android como en iOS se resuelve de manera similar. Notificas al sistema que deseas ser informado de cambios de posición, y cuando el evento ocurre, eres notificado. Filosofía *Model-View-Controller*.

En nuestro caso implementamos la localización como un *singleton*¹³, de manera que en cualquier momento y cualquier clase de nuestra aplicación, podamos acceder a los valores de una única instancia.

En el caso de iOS es necesario utilizar el *framework* **CoreLocation** que permite obtener la posición y dirección del dispositivo. En Android usaremos la librería **Location**.

Como se demostrará más adelante y formará parte habitual de la implementación, en Android obtenemos muchas más posibilidades de cara al desarrollador, siendo en iOS todo más directo y a su vez, simple. Mientras que en iOS todos los parámetros de configuración respecto a la localización se basan en la precisión deseada y la mínima distancia para ser notificada, en Android podemos configurar también los siguientes aspectos:

1. Obtener una lista de proveedores de localización válidos, así como sus nombres.
2. Especificar si queremos obtener localizaciones mediante GPS, A-GPS o ambos.
3. Obtener la última localización conocida (antes de ser notificado).
4. Crear localizaciones y proveedores de prueba o *mock*.

Además Google incorpora integración en su SDK con utilidades de localización, tales como añadir localizaciones manualmente o incorporar compatibilidad con ficheros *kml* (Google Earth), muy útil de cara al desarrollador.

Sin embargo, realizadas las pruebas obtenemos que, ya sea por el hardware o por el código en sí, en iOS se obtiene una señal de GPS (o A-GPS ya que no es posible distinguir en el actual SDK) más precisa y con más actualizaciones que su vertiente en Android.

Para cerciorarnos de estos indicios, realizamos una prueba sencilla. Montados en un coche con un terminal iPhone 4 y otro Android (un HTC *ChaChaCha*), ambos con conexión de datos 3G activada, intentamos buscar uno al otro. Obtuvimos una serie de localizaciones, posteriormente unidas en rutas.

¹³ Es un patrón de diseño cuya finalidad es tener una única instancia y proporcionar un punto de acceso global a ella.



Ilustración 19 Una de nuestras pruebas para demostrar la dispersión de datos. La ruta azul corresponde a un terminal iPhone. La ruta roja a uno Android

Tal y como podemos comprobar en ., hay una gran dispersión entre los datos, siendo en el caso de Android incluso poco razonables. Es por ello que se hizo necesario un sistema de filtrado de localizaciones donde establecimos las siguientes condiciones:

Se descartarán las siguientes localizaciones

1. Si la precisión es mayor que la de un valor decidido, en este caso 200 metros.
2. Si la precisión es claramente inferior a la localización previa.
3. Si ya hemos obtenido una localización igual a la actual.
4. Si el *timestamp* de la obtención de la localización es considerado “viejo”. En nuestro caso, consideramos antiguo cualquier localización con un intervalo entre la obtención de la posición y su posterior evento de más de 5 segundos.

Excepciones

1. Aceptaremos cualquier localización si es la primera que obtenemos.
2. Aceptaremos cualquier localización si hemos descartado cuatro localizaciones previas por falta de precisión (podría ser que el usuario estuviera pasando por una zona con poca cobertura).

Ayudas a la localización

Si bien la localización es la base de la aplicación, también es necesario transformar o interpretar esos valores de cara al usuario. Dicho de otra manera, al usuario de la aplicación no le interesa saber su latitud y longitud, sino que distancia existe entre su posición y la de su amigo. De la misma manera, tampoco le interesa saber si está apuntando hacia el norte o hacia el sur, sino hacia dónde debe dirigirse.

En el primer caso, obtener la distancia entre dos coordenadas, tanto Apple como Android ofrecen métodos que te permiten obtener dicho valor.

En el caso de Android este método es el siguiente:

```
public float distanceTo (Location dest)
Parametros
```

dest La localización destino.

Return Value La distancia en metros entre ambas localizaciones.

Mientras que para iOS es la siguiente:

```
- (CLLocationDistance)distanceFromLocation:(const CLLocation *)location
```

Parametros

Location La localización destino.

Return Value La distancia en metros entre ambas localizaciones.

Sin embargo, en un afán de mejorar la precisión, decidimos implementar otros modelos de localización. Implementamos el modelo de *Haversine*, La formula de *Haversine* se define de la siguiente manera:

$$d = 2r \arcsin \left(\sqrt{\text{haversion}(\phi_2 - \phi_1) + \cos(\phi_1) \cos(\phi_2) \text{haversion}(\psi_2 - \psi_1)} \right)$$
$$= 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\psi_2 - \psi_1}{2} \right)} \right)$$

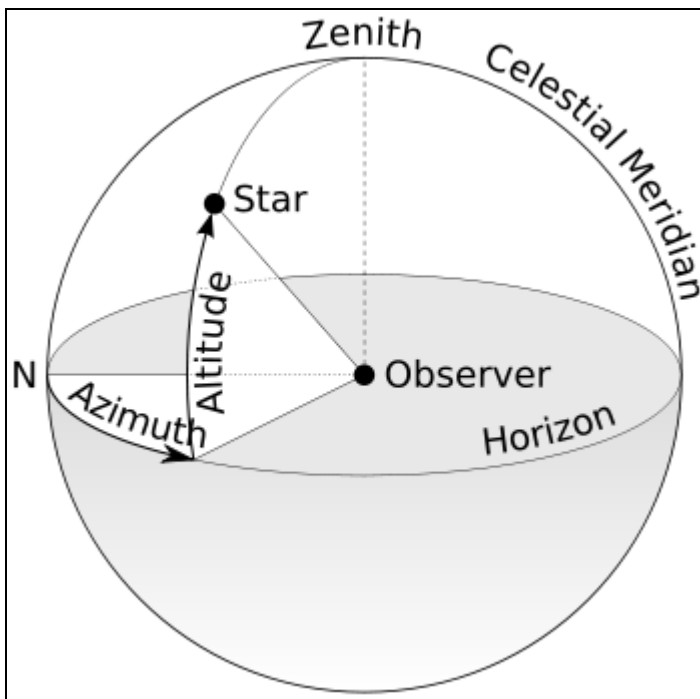
Siendo:

- d la distancia entre dos puntos
- r el radio de la esfera (en nuestro caso el planeta tierra)
- ϕ_1, ϕ_2 : las latitudes del punto inicial y final
- ψ_1, ψ_2 : las longitudes del punto inicial y final

En las pruebas realizadas obtuvimos que, manteniendo el modelo proporcionado por Apple y Android, obteníamos una precisión en torno al 2% mayor que con la fórmula de *Haversine*, por lo que fue descartada.

Orientación

Ya disponemos de la distancia entre dos puntos, pero ¿cómo calculamos la posición relativa entre ambos puntos? ¿Cómo sabemos qué punto está a la derecha y cual a la izquierda? ¿Cómo sabemos cómo llegar de un punto a otro? Sabemos la distancia pero no sabemos nada más. Es aquí donde entra el **bearing o azimuth**:



El *azimuth* es el ángulo formado entre una referencia fija (el Norte) y la línea de referencia entre el observador y el punto de interés y podemos calcularlo de la siguiente manera:

$$\theta = \text{atan2}(\sin(\Delta\text{long}) \cdot \cos(\text{lat}_2), \cos(\text{lat}_1) \cdot \sin(\text{lat}_2) - \sin(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \cos(\Delta\text{long}));$$

Como viene siendo habitual, Google ya implementa su propio método de cálculo del *bearing*. Sin embargo, Apple no incluye nada similar en su SDK, por lo que tuvimos que implementar manualmente la fórmula ya mostrada.

Pero tal y como hemos mencionado en la definición, tenemos siempre una referencia fija, el norte. No podemos presuponer que el usuario siempre va a estar mirando al norte.

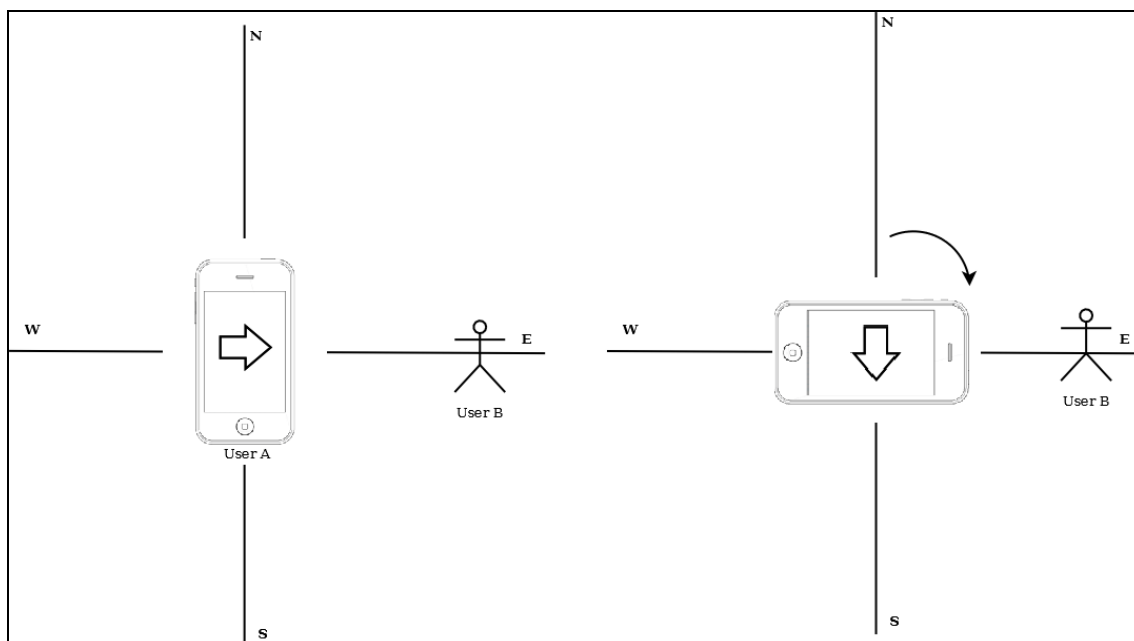


Ilustración 20 ¿Qué ocurre si el usuario A cambia su orientación respecto a su eje vertical pero mantiene su posición respecto a su eje horizontal?

Es aquí donde entra en juego el magnetómetro. Si sabemos la posición relativa del usuario respecto a los ejes cardinales, podemos actualizar el valor de referencia del *azimuth* y así mostrar la dirección correcta independientemente de la orientación del usuario.

Obtener los datos relativos al magnetómetro es muy similar a obtener los datos de localización. Al fin y al cabo, ambos son sensores y no existe diferencias entre la implementación, salvo el delegar sobre las clases adecuadas y sobrescribir los métodos necesarios. Y tal es la poca diferencia que nuevamente las semblanzas entre Apple y Google se mantienen. Mientras Apple te ofrece lo básico pero muy trabajado, Google te ofrece mucho pero menos elaborado. Ya depende la habilidad del desarrollador en elegir sus preferencias, aunque en el caso del magnetómetro se agravia bastante en el caso de Android ya que usando la misma implementación los resultados obtenidos pueden variar bastante según los terminales: tal y como hemos visto con el GPS algunos fabricantes usan mejores sensores que otros.

3.3.5. Conexiones de red

Una de las preguntas clave de todo el desarrollo del proyecto era la siguiente: ¿cómo nos comunicamos con nuestro servidor web? ¿Y con nuestro servidor de repetición? ¿Y nosotros con la aplicación? Es evidente que cada uno de ellos debe ser accesible desde el exterior, pero ¿cómo?

Gestionar las notificaciones push entrantes

Mediante las notificaciones push seremos capaces de enviar información de manera directa a nuestra aplicación, tales como solicitudes de localización recibidas, sin necesidad de que el usuario esté usando la aplicación en ese momento.

Toda la implementación del sistema de envío es proporcionado por Apple/Google, y está disponible desde la bibliografía.

En cambio, es deber de la aplicación como gestionar la notificación una vez recibida.

Es en nuestra pantalla de contactos donde vamos a gestionar las notificaciones push que recibamos, así como el estado de ellas.

- Notificaciones push entrants

Cuando la aplicación reciba una notificación push deberá acceder a la información contenida en ella, tal como teléfono o nombre del destinatario y adecuarla al usuario. Esto es, comprobar si el destinatario existe en sus contactos locales y en caso afirmativo, adecuar el mensaje (por ejemplo mostrar el nombre del contacto en la agenda en vez del recibido) a mostrar con dichos datos. Una vez realizada dicha comprobación, mostrará al usuario dicha solicitud.

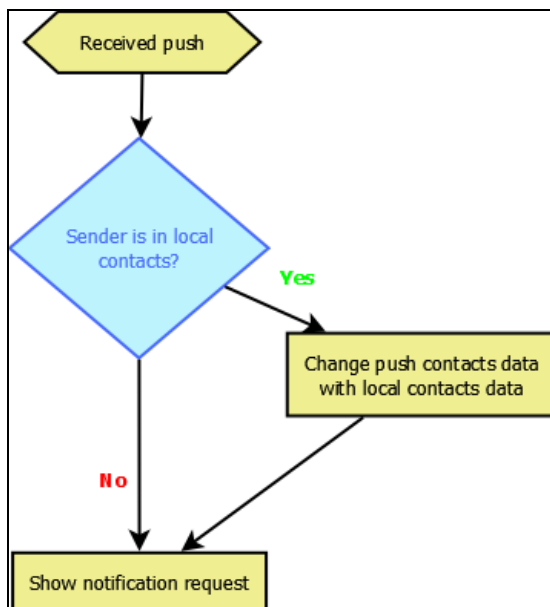


Ilustración 21 En cada notificación recibida localizamos los datos para el usuario

- Estado de las notificaciones push

Es indispensable para el funcionamiento de la aplicación que las notificaciones push funcionen y estén activadas. Dado que todas nuestras solicitudes y peticiones de acceso a los usuarios se realizan mediante dichos sistemas de comunicación, en el momento que estas no funcionen, la funcionalidad de nuestra aplicación queda gravemente disminuida al mínimo. Si bien no podemos acceder al estado general de la red de servicios push y hemos de confiar que siempre funcionen, sí podemos acceder al estado de las notificaciones push en el terminal del cliente. Si el cliente ha desactivado las notificaciones, ya sea de manera global o solo para nuestra aplicación, se le notificará de dicho inconveniente.

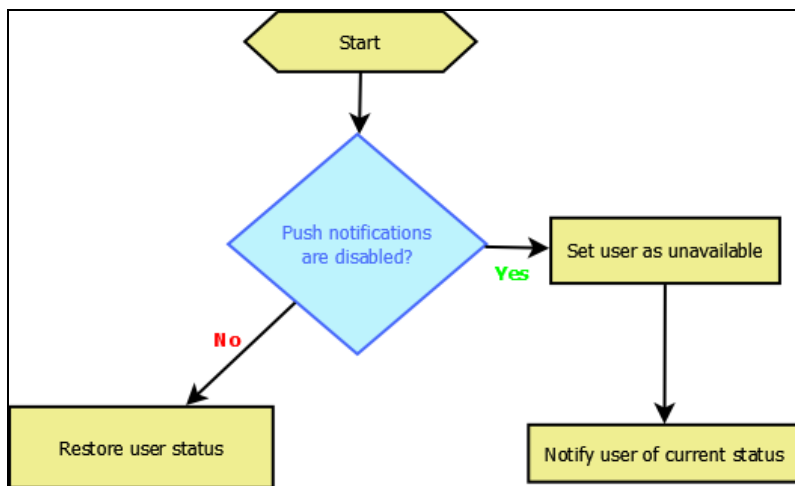



Ilustración 22 Las notificaciones push son estrictamente necesarias. Es por ello que debemos controlar cuando estén desactivadas



Aquí una diferencia enorme que cabe reseñar, y es la siguiente: en iOS no tenemos ningún tipo de control cuando nuestra aplicación está en segundo plano, salvo ligeras excepciones. Si bien en Android podemos crear servicios que se encarguen de determinadas tareas como las que acabamos de detallar, en iOS solo podemos realizar dichas tareas mientras nuestra aplicación esté siendo mostrada al usuario. En caso contrario y en este concreto, al usuario se le mostrará una notificación con los datos del remitente proveniente del registro. Sin embargo en Android podremos gestionar esta notificación entrante antes de que sea mostrada al usuario.

Conexión con nuestro servidor web

Es conocido que *Drupal*, siendo un gestor de contenidos, ya ofrece su propia integración con conexiones de red. Pero nosotros no buscamos una conexión directa con la web de *Drupal*. Nosotros queremos una conexión con el corazón de *Drupal*, con su implementación directa, independiente de interfaces o páginas web. Básicamente, acceder a una API de nuestro servidor.

Tal y como hemos visto en el apartado referente al diseño e implementación del servidor web, *Drupal* ofrece un módulo comunitario denominado *Services* que nos permite ofrecer este tipo de API al exterior. Nuestro módulo creado, debe cerciorarse de integrarse correctamente con dicho módulo.

Ahora ya tenemos una API, pero seguimos sin tener un protocolo de comunicación. El módulo *Services* ofrece varios como *XMLRPC* o *JSONRPC*.

Decidimos usar *XMLRPC* porque viene implementado de serie en el módulo. *JSONRPC* es un módulo aparte que se complementa con *Services*. Que sea la opción por defecto añade cierto factor de seguridad a la implementación.

Ahora ya tenemos el protocolo de comunicación y los métodos disponibles, pero sigue sin estar claro como conectamos nuestros *smartphones* a dicha API. Es evidente que si usamos *XMLRPC* como protocolo de comunicación, necesitaremos implementar una solución basada en éste.

Por suerte no fue necesario. Ni Apple ni Google ofrecen en su SDK implementaciones *XMLRPC*, pero sí la comunidad. En el caso de iOS usamos *Cocoa XML-RPC Client Framework* de *Divisible By Zero*, mientras que en Android usamos *android-xmlrpc*, también de código abierto.

El funcionamiento es bastante simple: nosotros hemos creado varios métodos en nuestro módulo para *Drupal*. Estos métodos se identifican mediante la tupla “<módulo>.<método>”, de manera que una simple llamada, a por ejemplo, el registro de usuario (ver diseño e implementación) se realizaría de la siguiente manera (versión Android)

```
XMLRPCClient client = new XMLRPCClient("http://<nuestroservidorweb>.com");  
  
bool registerOk = (Integer) client.call("compass.registerPhone", <telf_usuario>,  
<device_id>, <nombre_usuario>);
```

Tal y como podemos observar, más simple no puede ser. Pero también podemos apreciar que no hemos delimitado ningún tipo de seguridad. Cualquier usuario o cualquier aplicación externa podrían acceder a nuestros datos si conociera nuestro servidor web y la nomenclatura de nuestros métodos. Tal y como hemos definido en



Módulo Services hemos añadido una seguridad de tipo *API key authentication* a nuestro módulo y por lo tanto, nuestras llamadas a Drupal deben contener la siguiente información extra:

- *Timestamp* - El tiempo actual en formato Unix.
- *Domain* - El valor establecido en la configuración de Drupal.
- *Nonce* - Un valor aleatorio.
- *Hash* - Un hash con cifrado sha256 formado a partir del *timestamp*, dominio, *nonce* y nombre del método, y usando la API key establecida como llave compartida.

Por lo tanto, nuestras llamadas a Drupal añadirán un nuevo parámetro que incluirá dicha información extra.

Si denominamos el método encargado de generar dicha información como *generateDefaultParams(string methodname)*, nuestra anterior llamada a Drupal ahora será de la siguiente manera:

```
XMLRPCClient client = new XMLRPCClient("http://<nuestroservidorweb>.com");  
  
bool registerOk = (Integer) client.call(generateDefaultParams("compass.register"),  
<telf_usuario>, <nombre_usuario>, <device_id>);
```


3.4. Diseño e implementación servidor Echo

Hemos definido el servidor Echo¹⁴ como el servidor que va a repetir los mensajes entre los usuarios. Dado que los terminales no pueden conectarse entre sí (salvo que estén en la misma red privada), resultó necesario el diseño e implementación de dicho servidor. Su funcionalidad es simple: abrirá una conexión para cada cliente donde leerá sus mensajes y los reenviará al cliente destino.

En el actual apartado vamos a explicar los siguientes conceptos

- Diseño del servidor echo
- Implementación del servidor echo
- Implementación móvil de conexión al servidor.

¹⁴ Echo en inglés significa repetir. Nuestro servidor se encarga de repetir los mensajes que le llegan, de ahí la denominación.



3.4.1. Diseño del servidor echo

La clave del diseño del servidor *echo* radica en cómo estructurar el sistema de repetición de mensajes. Existen dos elementos clave, que son los siguientes:

- **Control de usuarios:** El servidor debe mantener una organización de los usuarios, crear/cerrar sockets según la conexión y garantizar la privacidad de los usuarios. Esto es, comprobar que un usuario A no obtenga datos de localización de un usuario B sin el consentimiento de éste. Es por ello que el servidor *echo* deberá comprobar los privilegios en cada conexión de usuario.
- **Control de mensajes:** El servidor debe reconocer los mensajes recibidos, interpretarlos adecuadamente y reenviar el mensaje al destinatario correspondiente. Así como en el servicio postal de correos la dirección destino está especificada en el mensaje (carta), nuestros mensajes al servidor *echo* también deben indicar el destinatario del mensaje.

Visto estos elementos, podemos separar nuestro servidor en dos clases: una clase “hija”, que gestionará individualmente cada conexión (enviar y recibir mensajes), y luego una clase “padre” que se encargará de la gestión de cada hijo así de como redireccionar cada mensaje entrante al hijo correspondiente. Dado esta estructura, decidimos nombrar a nuestros componentes **swarm** (enjambre) y **spawn** (huevo/semilla). El *swarm* se encargará de, a cada conexión entrante, comprobar si tiene permisos para conectarse y en caso afirmativo, asignarle un spawn (creado dinámicamente).

El *spawn* por su parte, leerá del socket cada mensaje que el cliente envíe. Si el mensaje recibido necesita ser redirigido, notificará al *swarm*, el cual buscará el *spawn* destinatario y éste escribirá en el socket para que el usuario destino obtenga el mensaje. Por lo tanto, designamos los diferentes tipos de mensajes que *spawn* puede interpretar:

- **iam : <device_id> - <req_phone_1> - <req_phone_2> ...**
Mensaje de bienvenida. Cuando el *spawn* reconozca este mensaje pedirá al *swarm* que compruebe si el *device_id* puede contactar con los teléfonos indicados.
- **msg: <message>**
Mensaje de repetición. Notificará al *swarm* que debe reenviar este mensaje a los amigos asociados al *spawn* actual.
- **bye: <message>**
Mensaje de cierre de conexión. Cierra el socket del remitente y notifica al *swarm* para que comunique a sus amigos que el cliente ha cerrado la conexión.

Podemos interpretar el flujo de mensajes reconocidos mediante el siguiente diagrama:

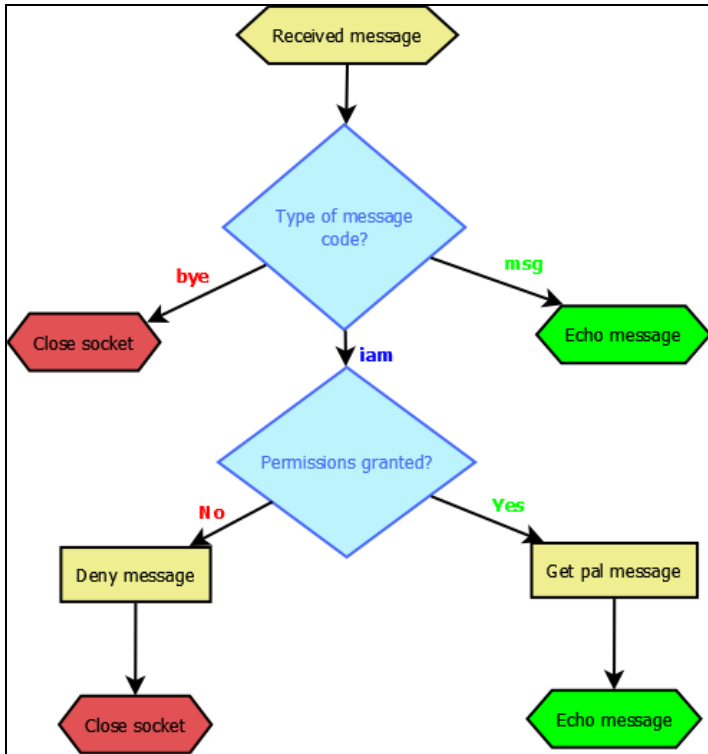


Ilustración 23 El servidor echo reconoce tres tipos de mensajes

3.4.2. Implementación del servidor echo

Para poder implementar el diseño establecido decidimos utilizar C++ como lenguaje de programación. Las razones fueron las siguientes:

- Experiencia previa en C/C++
- Documentación abundante
- Gestión de sockets
- Multiplataforma
- Integración con SSL (*OpenSSL*)
- Control de hilos de ejecución

La integración con SSL la consideramos necesaria para añadir encriptación a la información confidencial que se transmite. Si bien la falta de seguridad en la encriptación de los *device_id* no es prioritaria (ya que no aporta nada saberlo, al crearse individualmente por terminal y aplicación), sí lo eran por los mensajes relacionados con la posición y sobretodo con aquellos que incluyen el número de teléfono. Para ello usamos **OpenSSL**. OpenSSL se define como “*un esfuerzo de colaboración para desarrollar unas librerías Open Source robustas, completas y calidad comercial, que implementen los protocolos Secure Sockets Layer (SSL v2/v3) y Transport Layer Security (TLS v1).*”

Podemos definir el diagrama de clases tal y como prosigue:

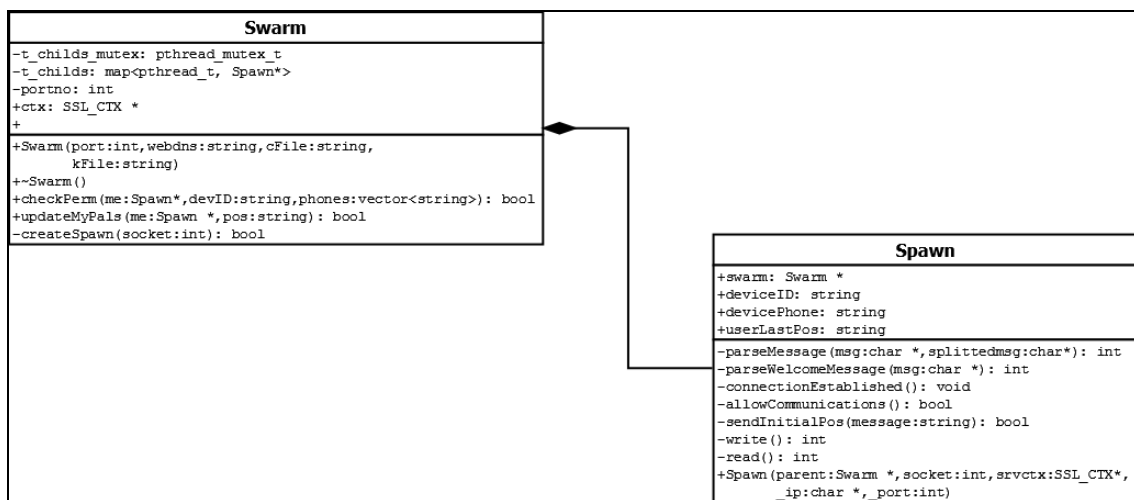


Ilustración 24 Diagrama de clases del servidor echo

Donde:

- `Swarm(int port, string webdns, string cFile, string kFile)`
: Constructor de la clase enjambre. Se le especifica parámetros tales como puerto o ficheros SSL a cargar.

- `bool checkPerm(Spawn *me, string devID, vector<string> phones)` : comprueba los permisos de localización almacenados en el **WebService**.
- `bool updateMyPals(Spawn *me, string pos)` : notifica a los **compañeros del spawn especificado sobre su nueva posición**.
- `bool createSpawn(int socket)` : **Crea un spawn en un socket determinado**.
- `int parseMessage(char *msg, char *splittedmsg)` : **parsea el mensaje recibido y devuelve el tipo de mensaje reconocido**
- `int parseWelcomeMessage(char *msg)` : **parsea el mensaje de bienvenida y almacena los datos necesarios**



3.4.3. Implementación móvil de conexión al servidor.

En cuanto a la conexión con nuestro servidor de repetición, implica un manejo de lenguaje a más bajo nivel principalmente por la inclusión del SSL en la conexión con el servidor.

La implementación no es complicada, salvo la inclusión del SSL, principalmente porque durante el periodo de pruebas no dispusimos de un certificado SSL autorizado por una empresa autorizadora.

Implementación iOS

En la versión para iPhone, la creación de sockets está gestionada por la clase `NSStream`. Sin embargo la implementación con SSL es muy exigente en relación a las condiciones de conexión. Por ejemplo, no podemos usar un certificado auto validado o uno ya expirado. La creación del socket es correcta, pero sin embargo no hay lectura/escritura alguna.

Es por ello que a la hora de hacer pruebas sin certificados válidos, tuvimos que rebajar las pretensiones de los sockets. Para ello, tuvimos que acceder a un nivel de implementación todavía más bajo y acceder directamente a la librería `CoreServices` y crear los sockets con `CFStream`.

`CFStream` sí que nos permite cancelar la comprobación de ciertos parámetros de la validación los niveles como certificados expirados o certificados raíz.

Las posibilidades son las siguientes (Apple, `CFStream Socket Additions`, 2008):

```
const CFStringRef kCFStreamSSLLevel;
const CFStringRef kCFStreamSSLAllowsExpiredCertificates;
const CFStringRef kCFStreamSSLAllowsExpiredRoots;
const CFStringRef kCFStreamSSLAllowsAnyRoot;
const CFStringRef kCFStreamSSLValidatesCertificateChain;
const CFStringRef kCFStreamSSLPeerName;
const CFStringRef kCFStreamSSLCertificates;
const CFStringRef kCFStreamSSLIsServer;
```

Por lo tanto podríamos definir unos sockets de prueba sin comprobación SSL de la siguiente manera

```
NSMutableDictionary *settings = [[NSMutableDictionary alloc] initWithObjectsAndKeys:
    [NSNumber numberWithInt:YES], kCFStreamSSLAllowsExpiredCertificates,
    [NSNumber numberWithInt:YES], kCFStreamSSLAllowsAnyRoot,
    [NSNumber numberWithInt:NO], kCFStreamSSLValidatesCertificateChain,
    kCFNull, kCFStreamSSLPeerName,
    nil];

CFReadStreamSetProperty((CFReadStreamRef)inStream, kCFStreamPropertySSLSettings,
    CFTypeRef) settings);
```

```
CFWriteStreamSetProperty((CFWriteStreamRef)outStream, kCFStreamPropertySSLSettings,  
(CFTypeRef) settings);
```

Implementación para Android

En Android como viene siendo costumbre, las posibilidades se multiplican y la libertad de implementación es prácticamente total. En Android disponemos de la clase `TrustManager` además de soporte a ficheros BKS (BouncyCastle, 2011), con los que podemos definir de manera segura qué certificados son válidos y comenzar con la negociación (*handshake*) con el servidor.

Para ello,

1. Obtenemos una copia del certificado del servidor
2. Importar la *KeyStore* como fichero BKS y añadirlo como recurso de la aplicación.
3. Creamos nuestro propio `TrustManager` con las especificaciones que queramos que se encargará del certificado y lo cargará dentro de una clase de tipo `SSLContext`
4. Usaremos esta clase `SSLContext` para todas las conexiones SSL.

De esta manera, el código equivalente a aceptar todos los certificados sería el siguiente:

```
/**  
 * Class which holds all the connecting implementation.  
 * @author jorge  
 *  
 */  
public class ClientThread implements Runnable {  
  
    /**  
     * Create a trust manager that does not validate certification chains  
     * Please take in mind that using this will make your connection vulnerable  
     * Should use only in development stages  
     */  
    TrustManager[] trustAllCerts = new TrustManager[] {  
        new X509TrustManager() {  
            public java.security.cert.X509Certificate[] getAcceptedIssuers() {  
                return null;  
            }  
            public void checkClientTrusted(  
                java.security.cert.X509Certificate[] certs, String authType) {  
            }  
            public void checkServerTrusted(  
                java.security.cert.X509Certificate[] certs, String authType) {  
            }  
        }  
    };  
  
    /**  
     * thread execution process (non-Javadoc)  
     * Tries to connect using a SSL connection with a BKS file as certificate  
     starting point. */
```



```
* The BKS file should be available as raw resource, with the specified name
* @see java.lang.Runnable#run()
*/
public void run() {

    InputStream store = context.getResources().openRawResource(STORE_RES);
    InetAddress serverAddr = null;
    try {
        serverAddr = InetAddress.getByName(ECHO_URL);
        KeyStore trusted = KeyStore.getInstance("BKS");
        try {
            trusted.load(store, BKS_PASS.toCharArray());
        } catch (Exception e) {
            e.printStackTrace();
            socketConnectionListener.connectionFailed(e.getMessage());
        } finally {
            try {
                store.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            socketConnectionListener.connectionFailed(e.getMessage());
        }
    }
    SSLSocketFactory socketFactory = new SSLSocketFactory(trusted);
    s = new Socket(serverAddr, ECHO_PORT);
    socket = (SSLSocket) socketFactory.createSocket(s, ECHO_URL,
ECHO_PORT, false);

    /* If you want to allow all certificates change it for this
    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(null, trustAllCerts, null);
    SSLSocketFactory socketFactory = sslContext.getSocketFactory();

    socket = (SSLSocket) socketFactory.createSocket("192.168.1.31",
1421);
    socket.setUseClientMode(true);*/

    socket.startHandshake();

    if (socket.isConnected() && socketConnectionListener != null) {
        socketConnectionListener.connectionEstablished();
    }

    connected = true;
    out = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
    in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
    manual_disconnect = false;

    sendWelcomeMessage(mDeviceID, mTargetPhone);
    listen();

    } catch (Exception e) {
        try {
            if (s != null)
                s.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        connected = false;
        if (socketConnectionListener != null) {
            socketConnectionListener.connectionFailed(e.getMessage());
        }
        e.printStackTrace();
    }
}
}
```


Una vez la conexión establecida, simplemente mantenemos abiertos los sockets de escritura y lectura y notificamos a nuestro controlador de cualquier cambio en la conexión para que actúe en consonancia. Las notificaciones que se han implementado son las siguientes:

1. Conexión establecida: Cuando la negociación entre el host y el cliente se ha establecido y ambos sockets están disponibles.
2. Conexión fallada: Cuando la conexión no se ha podido establecer.
3. Mensaje recibido: Cuando el socket de lectura ha recibido un mensaje.
4. Conexión cerrada por el host: Cuando el servidor ha cerrado de manera manual la conexión.

4. Manual de usuario

En el siguiente apartado vamos a explicar al usuario cómo acceder a cada una de las funcionalidades de la aplicación.

Registro del producto

Es necesario que registre sus datos para poder ser localizado o localizar contactos. Para ello, y si es la primera vez que inicie la aplicación verá una pantalla como la de la Ilustración 25 Pantalla de registro de usuario



Ilustración 25 Pantalla de registro de usuario

1. Indique un identificador de usuario y su número de teléfono. El teléfono solo se pide como parte de registro y le permite que otros contactos con su número de teléfono en su agenda puedan contactar con usted.
2. Además del número de teléfono debe especificar su país de origen. Esto es necesario debido a los prefijos telefónicos existentes.
3. Una vez haya completado el proceso, presione el botón de Enviar. Si ha rellenado correctamente los datos y además dispone de una conexión a Internet, en breves segundos debería recibir un SMS con un código de activación. Introdúzcalo en el apartado correspondiente de la aplicación.

Si no recibe ningún mensaje, siga las instrucciones que le muestra la aplicación.

Enviar solicitud de localización de contactos

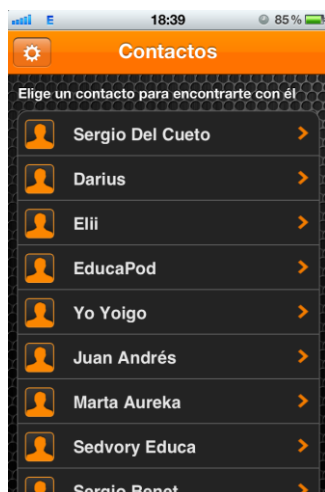


Ilustración 26 Pantalla de contactos

1. Para localizar un contacto, presione el contacto brevemente. Si dispone de una conexión a Internet accederá al modo de espera Ilustración 26 Pantalla de contactos.
2. En esta pantalla se le muestra un breve indicador del estado de su solicitud. Los indicadores cambiarán según el transcurso de ésta.
3. Si el contacto destino no ha aceptado su solicitud o bien no está disponible, se le notificará y volverá a la pantalla de contactos (ver Ilustración 27 No se puede encontrar al usuario).



Ilustración 27 No se puede encontrar al usuario

4. En caso de que el usuario haya aceptado la solicitud, comenzará el intercambio de posiciones. Intente situarse en posiciones con buena cobertura para una localización más fiable. Puede saber la calidad de su señal mediante el icono de cobertura.

Recibir solicitud de localización de contactos

1. Si algún usuario ha deseado contactar con usted, recibirá una notificación en su terminal con información del nombre y el teléfono remitente.



2. Si entra en la aplicación con una solicitud recibida tendrá la posibilidad de responder a dicha solicitud. Responda "Cancelar" si no desea contactar con dicho usuario. En caso contrario responda "Aceptar" y accederá a la pantalla de búsqueda.
3. A partir de este momento, comenzará el intercambio de posiciones. Intente situarse en posiciones con buena cobertura para una localización más fiable.

Deshabilitar localizaciones



Si desea no ser contactado, puede deshabilitar las solicitudes que reciba. Para ello:

1. Abra la aplicación y acceda al menú de Ajustes.
2. En el apartado de Estado, marque la opción "No Disponible".

A partir de este momento ningún usuario podrá enviarle solicitudes. Puede restablecer su estado en cualquier momento.

Cambio de apariencia

Dispone de varios aspectos para la aplicación que le permiten cambiar la interfaz del sistema. Para ello:

1. Abra la aplicación y acceda al menú de Ajustes.
2. En el apartado “Tema” marque la opción deseada. Verá como el aspecto de la aplicación cambia.

Compruebe los diferentes temas disponibles para elegir el más adecuado a sus gustos.

Compartir en redes sociales

Dispone de la posibilidad de compartir el trayecto que ha realizado en formato *Google Maps* a la hora de localizar a cualquier contacto suyo, con la gran mayoría de redes sociales. Para ello:



1. Localice un contacto e intercambie posiciones.
2. A partir del primer intercambio, el botón de Compartir se habilitará.
3. Si pulsa el botón Compartir, se le mostrará un diálogo para compartir el trayecto realizado en distintas redes sociales.
4. Seleccione la correspondiente y siga los pasos en pantalla.

5. Conclusiones

Finalizado el proyecto podemos concluir ciertos aspectos obtenidos en relación a la realización de éste.

- *Drupal* es un excelente gestor de contenidos, pero en las últimas versiones ha abandonado la simplicidad en pos de mayores posibilidades *out-of-the-box*. Podría darse el caso de que futuras versiones del *Core* de *Drupal* provocaran que no fueran una alternativa válida para proyectos pequeños y livianos como el del presente proyecto.
- En cuanto a Android y Apple, se ha venido demostrando durante toda la memoria y ha sido una constante: Android te ofrece libertad, mientras que Apple te ofrece simpleza y funcionalidad.

La SDK proporcionada por Apple además de completa, funciona perfectamente, con un emulador estable y con APIs que funcionan correctamente y tal y como se espera en cualquiera de sus terminales. Sin embargo, el afán de los de Cupertino en controlar todo, puede provocar muchos quebraderos de cabeza y complicaciones que no deberían existir, tal y como hemos visto en el apartado Implementación móvil de conexión al servidor.

Por su parte Google ofrece una SDK completa pero con multitud de fallos, con un emulador algo deficiente (en la última versión, la SDK 17 han sacado un emulador en x86 lo que debería mejorar su estabilidad). Eso unido a los centenares de terminales móviles compatibles puede complicar la implementación. Sin embargo, apoyados por el gran soporte que ofrece Java por la comunidad, supone una gran facilidad de obtención de código así como de su implementación.

La conclusión final al respecto es que Apple facilita mucho la depuración debido a su fiabilidad y la poca variedad de dispositivos, pero puede flaquear mucho en la implementación, al carecer de ciertas funcionalidades que otras SDK en otros lenguajes sí ofrecen. Google, con Android, es todo lo contrario. Un soporte enorme y una libertad igual, pero falto de experiencia en entornos de desarrollo, además de la complicación causada por tantos terminales compatibles con hardware diferente.

Lo mejor de todo es que las deficiencias de ambas compañías en este aspecto son fácilmente solucionables, por lo que el futuro debería deparar un camino para los desarrolladores todavía mejor.

6. Mejoras para el futuro

Debido a la falta de tiempo y a ser un proyecto muy voluminoso, se dejaron ciertas posibilidades que se podrían implementar en el futuro.

- **Más skins:** La introducción de más skins vendría acompañada de una ampliación de conocimiento relacionada a tecnologías como OpenGL. Además, se podría tratar temáticas como la descarga de contenido para su inclusión en la aplicación.
- **Soporte para Windows Phone:** Si bien el porcentaje de mercado actual de Microsoft en el mercado móvil es todavía menor, va en claro aumento y podría ser un buen contendiente en la lucha actual que mantienen Apple y Google. Eso, unido al buen hacer de Microsoft en términos relacionados a SDK e IDE, podría ser una buena iniciativa para adentrarse en .net.
- **Eliminación de contactos:** El usuario puede borrar la aplicación y es el propio SO el que se encarga de ignorar cualquier notificación entrante de aplicaciones no existentes. Sin embargo, ese usuario sigue existiendo en la base de datos y por lo tanto, será mostrado como válido para sus contactos. Para la implementación de esta mejora se proponen dos soluciones no excluyentes:
 - Opción directa al usuario de eliminar cuenta. Al pulsar el botón se eliminaría cualquier dato del usuario en los servidores.
 - Feedback de las notificaciones push: Apple y Google proporcionan un mecanismo para determinar si el envío de cualquier notificación push es válido. Podemos basarnos en los errores devueltos para determinar si el usuario ya no tiene instalada la aplicación.
- **Mejor gestión de la batería:** Uno de los mayores gastos producidos por la aplicación viene por las conexiones de red. Para solucionarlo se proponen dos alternativas, de nuevo no excluyentes
Mejorar las pruebas relacionadas con el GPS del móvil para así suavizar o maximizar la potencia de éste (y por tanto su consumo) en relación a la calidad de señal recibida.
Evitar las conexiones constantes con el servidor web para solicitar contactos. Podríamos determinar si el usuario necesita actualizar sus contactos en el momento que alguno de ellos se haya registrado.
- **Soporte multiusuario:** Si bien la finalidad del proyecto era de hacer una aplicación simple, toda la estructura detrás permite unas mayores

posibilidades. Una de ellas, y de la cual el servidor de repetición ya está preparado, es implementar búsquedas multiusuario, de manera que un usuario pueda obtener información de localización de varios de sus contactos simultáneamente. Obviamente habría que modificar la interfaz para estos nuevos usos.

- **Mensajes de texto:** Hemos realizado un protocolo de envío de mensajes como hemos visto en Diseño e implementación servidor Echo. Podríamos incluir una nueva funcionalidad a dicho protocolo para que interpretara mensajes de texto predefinidos o escritos por el usuario, que serían mostrados al usuario destino. Esto añadiría una funcionalidad de chat a la aplicación, pero con la premisa de conservar la identidad del producto. Que el usuario pudiera enviar mensajes del estilo “*estoy llegando*”, “*no te encuentro*”...podría enriquecer la experiencia del usuario final.
- **Conexión https al servidor Drupal:** La conexión al servidor de repetición es segura y encriptada. Sin embargo el módulo Services proporcionado por la comunidad de Drupal ofrece una seguridad basada en autenticaciones, pero no en seguridad https (SSL/TLS).



7. Bibliografía

Android-xmlrpc. (2010). Retrieved from Android-xmlrpc:

<http://code.google.com/p/android-xmlrpc/>

Apple. (2008). *CFStream Socket Additions*. Retrieved from CFStream Socket Additions:

<http://developer.apple.com/library/mac/#documentation/CoreFoundation/Reference/CFReadStreamRef/Reference/reference.html>

Apple. (2011, 08 09). *Local and Push Notification Programming Guide*. Retrieved from Local and Push Notification Programming Guide:

<http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html>

Apple. (2011, 02 24). *OpenGL ES Programming Guide for iOS*. Retrieved from OpenGL ES Programming Guide for iOS:

http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGL_ES_ProgrammingGuide/WorkingwithEAGLContexts/WorkingwithEAGLContexts.html

BouncyCastle. (2011). *The Legion of the Bouncy Castle*. Retrieved from The Legion of the Bouncy Castle: <http://www.bouncycastle.org/specifications.html>

Chisnall, D. (2011). *Objective-C Phrasebook*. Addison-Wesley.

Drupal. (n.d.). *Drupal: Open Source CMS*. Retrieved from <http://drupal.org/>

Drupal. (n.d.). *Installation guide: drupal.org*. Retrieved from <http://drupal.org/documentation/install>

Drupal. (2011, 11 05). *Writing .info files (Drupal 6.x)*. Retrieved from Writing .info files (Drupal 6.x): <http://drupal.org/node/231036>

Drupal. (2008, 07 21). *Writing .install files*. Retrieved from Writing .install files: <http://drupal.org/node/51220>

GandoGames. (2010, 07). *Tutorial: Using Anti-Aliasing (MSAA) in the iPhone*. Retrieved from Tutorial: Using Anti-Aliasing (MSAA) in the iPhone:

<http://www.gandogames.com/2010/07/tutorial-using-anti-aliasing-msaa-in-the-iphone/>

Google. (n.d.). *Android Cloud to Device Messaging Framework*. Retrieved from Android Cloud to Device Messaging Framework: <http://code.google.com/intl/es-ES/android/c2dm/>

LaMarche, J. (2009). *iPhone Development : OpenGL ES From the Ground Up*. Retrieved from <http://iphonedevdevelopment.blogspot.com.es/2009/04/opengl-es-from-ground-up-part-1-basic.html>

Nilson, J. (2010). *Action Bar for Android*. Retrieved from Action Bar for Android: <https://github.com/johannilsson/android-actionbar>

Rocchi, C. (2011). *Ray Wenderlich : How To Create A Socket Based iPhone App and Server*. Retrieved from Ray Wenderlich : How To Create A Socket Based iPhone App and Server: <http://www.raywenderlich.com/3932/how-to-create-a-socket-based-iphone-app-and-server>

Veness, C. (2010). *Movable Type Scripts : Calculate distance, bearing and more between Latitude/Longitude points*. Retrieved from <http://www.movable-type.co.uk/scripts/latlong.html>

Wenderlich, R. (2011). *iOS5 by Tutorials*.

