Escuela Técnica Superior de Ingeniería de Telecomunicación

TRABAJO FIN DE GRADO

Autor:
Miguel Sánchez del Águila
Director:
Fernando Losilla López

Septiembre 2014

Ficha general del proyecto

Autor: Miguel Sánchez del Águila	**E-mail: miguelmsa1@gmail.com**
Director: Fernando Losilla López	**E-mail: fernando.losilla@upct.es**

Título del proyecto:
Desarrollo de un sistema de seguimiento de vehículos para dispositivos Android basado en el control de redes de sensores

Resumen:
Sistema de seguimiento de vehículos, compuesto por:

- Equipos de bajo consumo equipados con una cámara web, que gestionan una serie de dispositivos con un sensor magnético, programados para almacenar la información de los objetos metálicos que circulen sobre ellos.

- Un servidor web encargado de acceder a la información centralizada en una base de datos MySQL.

- Una aplicación para dispositivos móviles (Android e iOS) encargada de clasificar toda esa información de manera útil para el usuario, representando la información a monitorizar, y representando en caso que el usuario lo solicite, un mapa con la ubicación de los vehículos seleccionados.

Titulación:
GRADO EN INGENIERÍA TELEMÁTICA

Departamento:
Depto. TIC

Fecha de presentación:
Septiembre 2014

Índice

1. Descripción general del proyecto

1.1. Motivación y Objetivo

Como Ingeniero Técnico de Telecomunicaciones especializado en Telemática, siento especial interés por la tecnología, tanto la actual, como la predecesora. A esto se añade que, dentro del mundo tecnológico, ha quedado demostrada la importancia de las comunicaciones, el intercambio de información, en un mundo cada vez más conectado y actualizado.

Un claro ejemplo de esta situación se observa en el crecimiento de los denominados como "Servicios Cloud".

Gracias a la potencia de las comunicaciones actuales, se ha permitido a los usuarios reducir sus infraestructuras de red y telefonía (servidores, centralitas telefónicas,…) optando por obtener esos mismos servicios, proporcionados por máquinas virtuales alojadas en localizaciones externas, reduciendo así consumo eléctrico, espacio, mantenimiento…

Esto provoca que el mundo de las telecomunicaciones tenga un impacto muy fuerte en la sociedad, marcando casi en gran parte la tecnólogía que se usa en la vida cotidiana.

Antiguamente, estamos hablando de no hace ni diez años, teníamos muchos dispositivos, y cada uno de ellos realizaba su función, hoy en día, es muy común que el teléfono inteligente que llevamos en el bolsillo sea más potente que todos esos dispositivos juntos. Por esta razón, se dice que estamos en el momento de las "apps", es decir, pequeños programas para los teléfonos inteligentes, tablets y electrónica de consumo que cumplen las necesidades del usuario.

Si a esta evolución, añadimos avances necesarios en determinados sectores como los cuerpos de seguridad, se entiende que la policía y el ejército debe contar con herramientas avanzadas que le permitan cumplir con su labor de proteger al ciudadano. Para esto, no tenemos por qué hablar de hardware más avanzado, a veces es suficiente con enfocar los recursos actuales en la dirección adecuada para suplir otras necesidades.

Éste proyecto es una muestra de lo mencionado hasta ahora, juntar la potencia de la nueva electrónica de consumo, que permite enfocarla a aplicaciones específicas, junto con hardware actual, y darle un enfoque para que cumpla con una función, que hasta ahora no se había implementado.

El objetivo de éste proyecto es la monitorización mediante terminales móviles, de los vehículos que pasan por diferentes calles en una cuidad objetivo, y una vez seleccionado uno de esos vehículos, obtener información sobre él, como por ejemplo, recorrido que sigue y distintas imágenes del vehículo.

La caracterización de cada vehículo se consigue a través de una red de sensores magnéticos que miden el campo magnético generado por un cuerpo metálico de

gran volumen (en este caso, un vehículo). La conversión de éste campo magnético a un valor cuantificable será lo que denominaremos “firma magnética”.

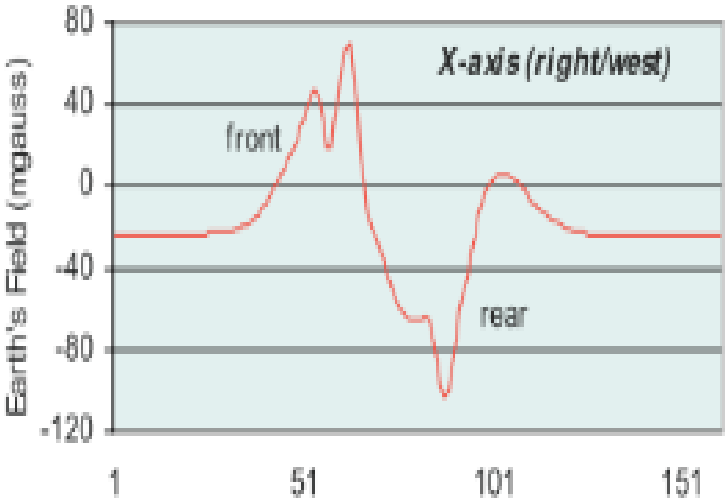


Figura 0. Posible firma magnética de un vehículo

Éste proyecto consiste en el desarrollo del sistema que intercambiará mensajes con éstas firmas magnéticas y los hará llegar hasta los sensores conectados a la red, permitiendo así comparar estas firmas con las que sean detectadas por ésta red de sensores externa, recopilar todos éstos datos, y representarlos en una aplicación móvil.

Para ello, aparte de la propia aplicación móvil, se ha desarrollado:

- Una base de datos
- Un servidor web
- Un equipo con una que gestione la información de los vehículos
- Dispositivos que recojan información de los vehículos

Los procesos de cada uno de los elementos componentes del sistema se explican de manera más extendida a lo largo de ésta memoria.

1.2. Cuestiones metodológicas

1. ¿De dónde surge la idea del proyecto?
2. ¿Cómo de eficiente es el sistema?
3. ¿Por qué he escogido estos lenguajes de programación?
4. ¿Es un proyecto viable?
5. ¿Llegará a implantarse para su uso cotidiano?

1. ¿De dónde surge la idea del proyecto?

La idea original del proyecto, surgió hace dos años, con el proyecto titulado "Desarrollo de un sistema de seguimiento de vehículos para dispositivos iOS basado en el control de redes de sensores" realizado por mí en la Universidad Politécnica de Cartagena. Éste sería la ampliación y continuación del proyecto mencionado

Ha sido desarrollado paralelamente con otro proyecto realizado por el alumno José Manuel López Egea, estudiante de la ETSIT, el cuál se titula "Estudio E Implementación De Un Sistema De Seguimiento De Vehículos Con Una Red De Sensores Inalámbrica", ambos dirigidos por el profesor Fernando Losilla.

Como bien se detalla en el otro proyecto, los nodos sensores "Motes MicaZ" utilizados en el sistema son capaces de catalogar diferentes modelos de objetos metálicos, lo cuál es muy interesante en ciertos campos. Si a esto le unimos la fuerza que está cogiendo el mundo de las aplicaciones móviles, surge la idea de hacer una aplicación para controlar la información que se puede extraer del sistema.

2. ¿Cómo de eficiente es el sistema?

Si está completamente implementado, es bastante preciso y rápido, teniendo siempre en cuenta las limitaciones que tienen los sensores magnéticos, pero para ello se añade también el apoyo visual de las imágenes captadas por una cámara, lo cuál facilita la identificación del vehículo.

3. ¿Por qué he escogido éstos lenguajes de programación?

Remítase al punto 2.2 titulado "Descripción de los lenguajes utilizados".

4. *¿Es un proyecto viable?*

La viabilidad del proyecto está directamente relacionada con el precio de los nodos sensores "Motes MicaZ", los cuales todavía no tienen un uso cotidiano, y no se fabrican en una escala tan grande, pero se espera que en los próximos años, la cantidad de estos productos en el mercado será más grande, y por tanto, su coste de producción bajará drásticamente, en ese momento si considero que será un proyecto viable.

5. *¿Llegará a implantarse para su uso cotidiano?*

Ese aspecto lo dejo para el punto 6 de esta memoria, titulado "Conclusión y ampliaciones".

1.3. Entorno de la aplicación

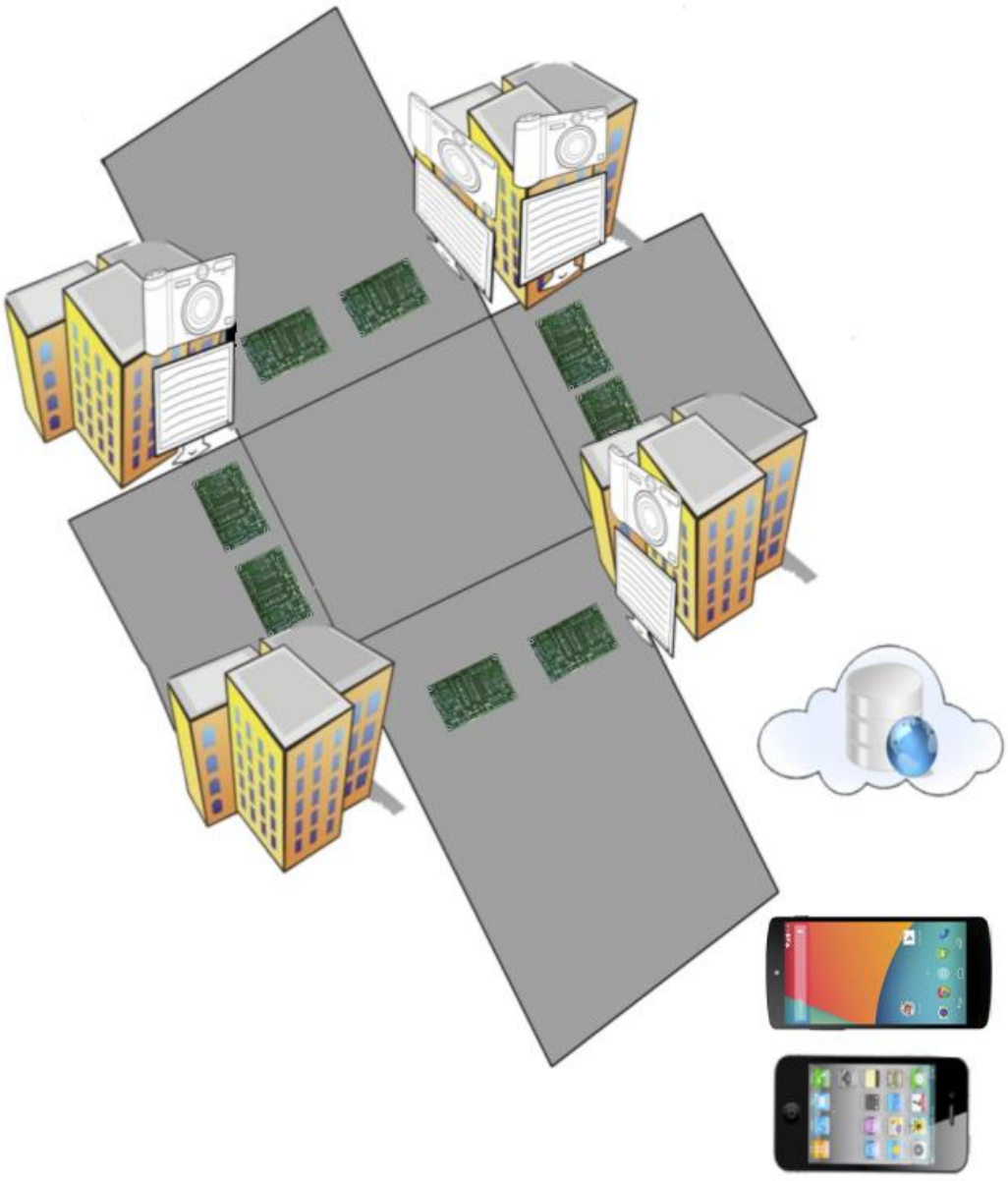


Figura 1. Boceto del entorno general del sistema

El entorno propicio para este sistema, sería el que se puede apreciar en la imagen, un conjunto de nodos sensores en cada calle, con el correspondiente equipo que los controle.

Como se ve en el esquema, el sistema está compuesto por:

- Un conjunto de nodos sensores en cada calle que se encarga de identificar los vehículos.
- Un equipo equipado con cámara
- Un servidor web con una base de datos
- Una aplicación para Android
- Una aplicación para iOS

2. Descripción general del producto

2.1. Visión general del sistema

2.1.1. Elementos

Los **nodos sensores** con los que están equipados las calles, tienen como función identificar el elemento que pasa por sus sensores magnéticos.

El equipo que gestiona cada conjunto de nodos sensores (**equipos gestores**), ejecuta un script, el cuál le permite enviar y recibir determinados mensajes tanto de los nodos sensores como del servidor web. Posee también una cámara, que utilizará para obtener más información de los coches que pasen por los nodos sensores.

El **servidor web** posee unos web services PHP que le permiten actuar de pasarela entre la base de datos MySQL, la aplicación móvil y los equipos gestores.

La **base de datos** MySQL, contiene toda información sobre las redes de nodos sensores registradas en el sistema, los coches que pasan por cada calle, y el recorrido concreto del coche seleccionado.

Las **aplicaciones** para dispositivos móviles, son el visor de esa información de cara al usuario, facilitando la observación de ésta, y realizando el proceso de forma transparente para el usuario.

2.1.2. Funcionalidad básica

La aplicación tiene dos modos de funcionalidad básica:

- Observar los coches que circulan por una calle determinada.
- Obtener el recorrido en tiempo real de un coche determinado.

Para ambos, el comienzo del proceso es similar:

El primer paso es instalar los sensores en la localización correspondiente de cada calle a monitorizar.

Una vez colocados, habría que registrar ese conjunto de sensores en la red, de ésta tarea se encarga el equipo gestor con un software destinado a tal efecto.

El servidor web y la base de datos estarían en continuo funcionamiento desde el primer momento.

La siguiente interacción depende de cuando el usuario final (policía patrulla) inicie la aplicación móvil. En ese momento obtiene la información de las calles registradas por los equipos gestores.

El usuario selecciona una de las calles registradas.

Aparece una lista con los coches que circulan por esa calle.

En este punto, el usuario puede seleccionar un coche para obtener su recorrido, o puede volver atrás para observar los coches que circulan por otra calle.

En caso de que decida obtener el recorrido de un determinado coche, la aplicación muestra una ventana con un mapa y una lista, que muestra las diferentes calles por las que el coche ha pasado desde que inició la petición.

2.1.3. Usuarios

No sería conveniente ceder la información que proporciona este sistema a un usuario común, pues podría hacer uso de ella para cometer delitos como acoso, o beneficiarse de saber dónde está en ese momento un coche de policía.

Por tanto, está enfocada para un uso en el ámbito policial, por tanto, el usuario de la aplicación, podría ser un policía que estuviese de patrulla por la cuidad.

Para no permitir que salga de ese entorno, se podría recurrir a técnicas comentadas en la sección 6 de la memoria titulada “Conclusión y ampliaciones”.

2.1.4. Interacción con sistemas externos

Otras sistemas con los que la aplicación podría interactuar para aportar datos interesantes y útiles, podrían ser, los centros de estadística, aportando datos propicios para los estudios que se realizan desde este organismo en materia de tráfico y congestión automovilística.

Otro posible interesado en la recopilación de éstos datos podría ser la DGT, recopilando la cantidad de vehículos que circulan por cada calle por día u hora. El estado de ciertas calles y la monitorización de los conocidos como “puntos negros”.

También podría interactuar con una posible aplicación móvil para otros sistemas operativos, la cuál no queda recogida en este proyecto.

2.2. Descripción de los lenguajes utilizados

Cada uno de los componentes del sistema está programado en un lenguaje diferente:

- Para los equipos que coordinan los nodos sensores con el servidor (equipos gestores), se ha usado el lenguaje de programación **Java**, el cuál es un lenguaje orientado a objetos. Java elimina herramientas de bajo nivel como los punteros. Al ser un lenguaje interpretado, permite tener independencia del hardware. Por lo tanto, un programa escrito en Java, funciona en cualquier equipo que tenga la máquina virtual de Java instalada. Cuenta también con un recolector de basura, el cual almacena las referencias que se hacen a un objeto, y cuando esas referencias se pierden, al convertirse en un objeto inaccesible, lo elimina, recuperando así espacio en memoria.

  He seleccionado este lenguaje para ésta aplicación por la versatilidad que puede tener a la hora de funcionar en diferentes equipos, por su eficiente comportamiento con conexiones a bases de datos, por la facilidad para manejar sockets y porque resulta sencillo el acceso a la cámara para captar las imágenes de los vehículos.

- Para la centralización desde el servidor se han utilizado web services desarrollados en **PHP**, el cuál se ha convertido en el lenguaje más utilizado para la creación de páginas web dinámicas, y su potente relación con las bases de datos lo convierte en un lenguaje de programación muy interesante para proyectos de gran envergadura.

  Me he decantado por PHP debido a lo anteriormente comentado, junto con la característica de que no es un lenguaje muy complejo, una vez que se conocen los aspectos básicos de C, resulta muy parecido, y por tanto la curva de aprendizaje para pasar de uno a otro es muy pequeña.

  También es interesante saber, que un código escrito en PHP, es difícil de obtener de cara a un usuario malintencionado, pues el código se ejecuta en el servidor, y el navegador devuelve el código HTML resultante.

  Cabe destacar, que estos web services están basados en una técnica de arquitectura de software conocida como “REST” que ha sustituido a los web services que se venían utilizando hasta ahora basados principalmente en “SOAP”.

  Esta arquitectura simplifica mucho el funcionamiento del sistema, ya que se basa en peticiones HTTP y no depende de ningún estado.

  Una implementación REST pura puede dar lugar a discusión incluso entre los más experimentados que ésta metodología, es por ésta razón, que alguna de las API desarrolladas en este proyecto puede contener características de RPC.

  REST queda explicado de una manera más extensa en el anexo destinado a tal efecto.

- En los entornos de bases de datos, el lenguaje utilizado para realizar las consultas es **SQL**, desarrollado por IBM en 1986. SQL apareció por la necesidad del manejo de grandes bases de datos relacionales.

  En una consulta SQL, únicamente hay que especificar la información que se quiere consultar, no hay que hacer ningún tipo de mención al proceso.

  Las consultas SQL aparecen en las partes del proyecto en las cuáles hay que crear o alterar alguna tabla de la base de datos, o cuando se requiere algún tipo de información que esté almacenada en ella, tal como los nodos sensores registrados en el sistema, o los coches que registra cada uno de los nodos sensores. No obstante, la estructura inicial de la base de datos también se modela con SQL y están en continuo funcionamiento antes de que el usuario interactúe con la aplicación.

- La información que se envía desde el servidor hacia la aplicación móvil está codificada en **JSON**. Éste no es un lenguaje como tal, es un formato para el intercambio de datos, pero con la importancia del Cloud Computing y la cantidad de información que viaja por la red éstos años, comienza a ocupar un papel importante en el intercambio de información.

  JSON es la alternativa a XML, codifica objetos dando como resultado una cadena de texto de reducido tamaño.
  Aquí se puede ver el resultado de un posible objeto JSON:

  [{"latitud":"37.995914","longitud":"-1.139705","nombre":"Miguel de Unamuno","id":"1"},{"latitud":"37.995433","longitud":"-1.140143","nombre":"Calle Pina","id":"2"}]

  La información que contiene, se estructura de la siguiente manera según el estándar del formato:

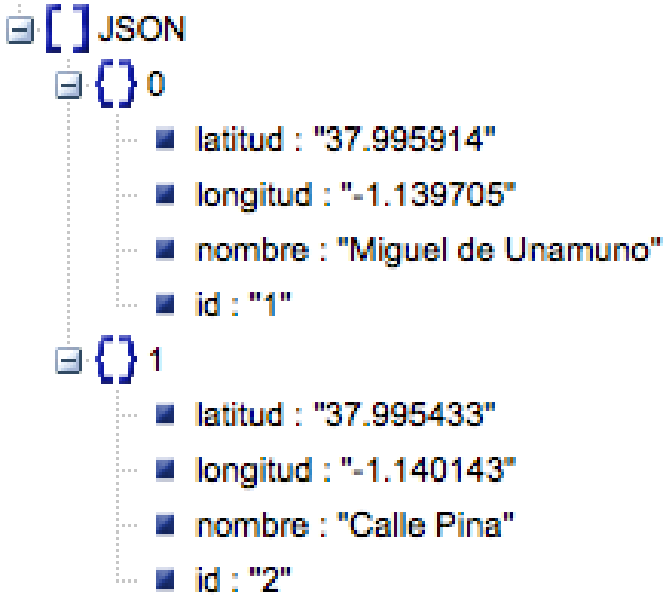


Figura 2. Ejemplo de estructura de objeto JSON

Con la estandarización de JSON han surgido visores (extensiones) que facilitan la visualización de éstos objetos en los navegadores web más utilizado. Es recomendable obtener alguno de ellos si se comienza un proyecto que integre este formato.

- Para la aplicación móvil de iOS se han utilizado las herramientas para desarrolladores que ofrece Apple, tal es el caso del entorno XCode, junto con los simuladores para dispositivos iOS. El lenguaje utilizado es **Objective-C** (Obj-C), un lenguaje orientado a objetos, diseñado por Brad Cox en 1980.

  Adopta gran parte de la filosofía de SmallTalk, es decir, las aplicaciones constan de un gran número de objetos que se comunican entre ellos mediante el paso de mensajes.

  Por tanto, en Obj-C el programador no llama a métodos, sino que envía mensajes.
  Al igual que en C++ en Obj-C los objetos de definen por un archivo interfaz ".h", donde quedan recogidas las variables de instancia y los métodos de la clase, y un archivo donde se realiza la implementación del objeto ".m".

  Apple se encargó de diseñar un framework llamado "Cocoa" que permite el desarrollo de aplicaciones nativas para su sistema operativo "Mac OS X". Más adelante, con la aparición de los dispositivos iOS, realizó ciertos cambios sobre ese framework, lo que dió lugar a un nuevo framework que recibe el nombre de "Cocoa Touch".

  Este framework proporciona al lenguaje Obj-C las herramientas necesarias para manejar los objetos fundamentales a la hora de crear aplicaciones para sus dispositivos, como las ventanas de la aplicación (llamadas también vistas), tablas donde insertar la información, botones de acción, librerías de gestos para el panel táctil, barras de navegación para la aplicación, un mapa con multitud de herramientas… etc.

Con la aparición de iOS 5, Apple diseñó una herramienta para el diseño del apartado gráfico de la aplicación "StoryBoard", más adelante se expondrá la potencia de ésta herramienta con un pequeño ejemplo.

- Por otro lado, para la otra versión de la aplicación móvil destinada al sistema operativo Android, he tenido que ceñirme al propio SDK proporcionado por el equipo de Google. Aunque la base de **Android** es Java, junto con el IDE Eclipe, son tantas las librerías y frameworks que este SDK añade al lenguaje, que deberíamos considerarlo un lenguaje aparte, porque no por ser un buen programador en Java quiere decir que te desenvuelvas perfectamente con Android.

  Esta experiencia puede servir de ayuda sí, pero requiere un ejercicio de documentación, conocer en profundidad las librerías principales del lenguaje así como hacerte al modo de desarrollar las interfaces.

  En este aspecto he de decir que Google tiene trabajo pendiente, que está intentando encaminar con su herramienta (todavía en fase beta) Android Studio.
  Bajo mi punto de vista, opino que deberían centrarse más en cuidar las herramientas para desarrolladores, ya que el uso tanto del IDE basado en eclipse, como los emuladores que proporcionan para testear las aplicaciones, te hace sentir que estás ante una herramienta inacabada, con fallos muchas veces sin sentido, y la experiencia de los emuladores, es simplemente nefasta.

  Hay que recurrir a implementaciones de terceros o a dispositivos físicos para poder realizar las pruebas pertinentes.

La sección "Anexos" contiene más información sobre cada uno de los lenguajes utilizados para la implementación del sistema.

3. Manual de usuario y Look & Feel

En este punto se va a hacer una aclaración del apartado visual de la aplicación móvil, ya que es la parte del sistema que más interacción necesita de cara al usuario.

Al iniciar, aparece una vista de inicio simple, que permite dar comienzo al programa, o nos permite salir de él:

Figura 3. Vista principal del programa

El cierre forzado de aplicaciones en iPhone no está permitido por Apple, por tanto, si el usuario decide salir pulsando el botón, recibe la indicación de cómo cerrarlo. En Android el botón sí que cumple correctamente su función.

Si se presiona el botón comenzar, se da paso a todo el proceso descrito anteriormente, y aparece ante el usuario una ventana que le permite seleccionar uno de los equipos gestores registrados en el sistema.

Dependiendo del sistema operativo que posea el terminal, la navegación por la aplicación cambiará levemente:

- En Android, seleccionar un elemento de la tabla nos llevará a la siguiente vista, mientras que para volver atrás lo realizaremos con el botón destinado a ésta función que forma parte del sistema operativo, situado abajo a la derecha.
- En iOS, la navegación se realiza mediante los botones que existen en cada uno de los elementos de la tabla, y para volver atrás tendremos que utilizar los botones que aparecen en la barra de navegación superior.

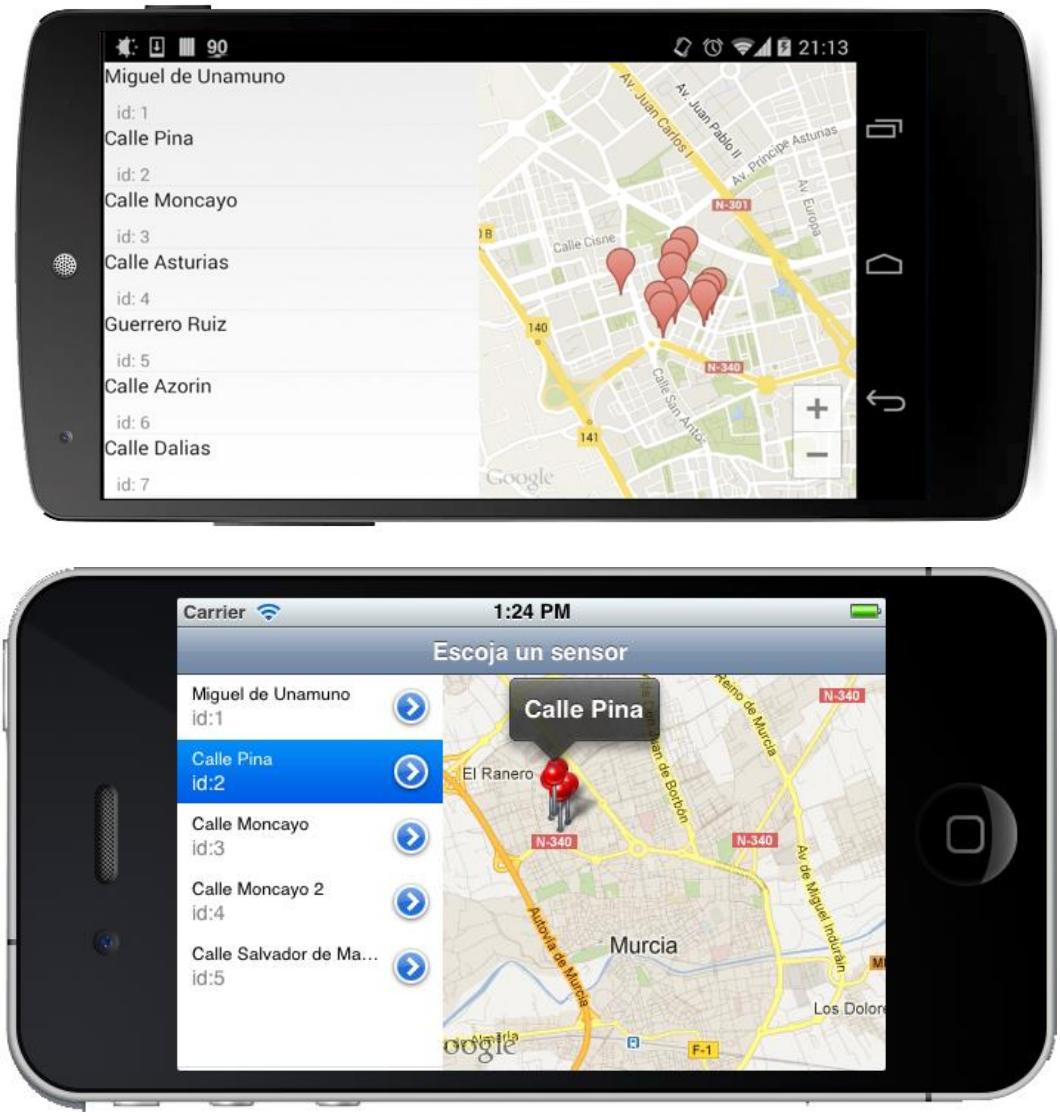


Figura 4. Antes de escoger un sensor

Cuando el usuario se decanta por uno de los nodos sensores, pasa a una vista que posee una tabla vacía con el título “Cargando”.

A partir de éste momento, cuando un coche pasa por encima del gestor seleccionado, al usuario le aparece la información de ese coche representada de la siguiente manera.

Figura 5. Lista de firmas de un sensor

Contenido de la tabla:

- Foto del vehículo
- Identificador único para cada firma

La interacción del usuario con la aplicación puede acabar en éste punto, ya que puede darse el caso que solo le interese monitorizar el tráfico que circula por una calle.

En caso de que desee inspeccionar el recorrido de un vehículo determinado, se pasa a la siguiente ventana, muy similar a la vista de selección del sensor inicial:

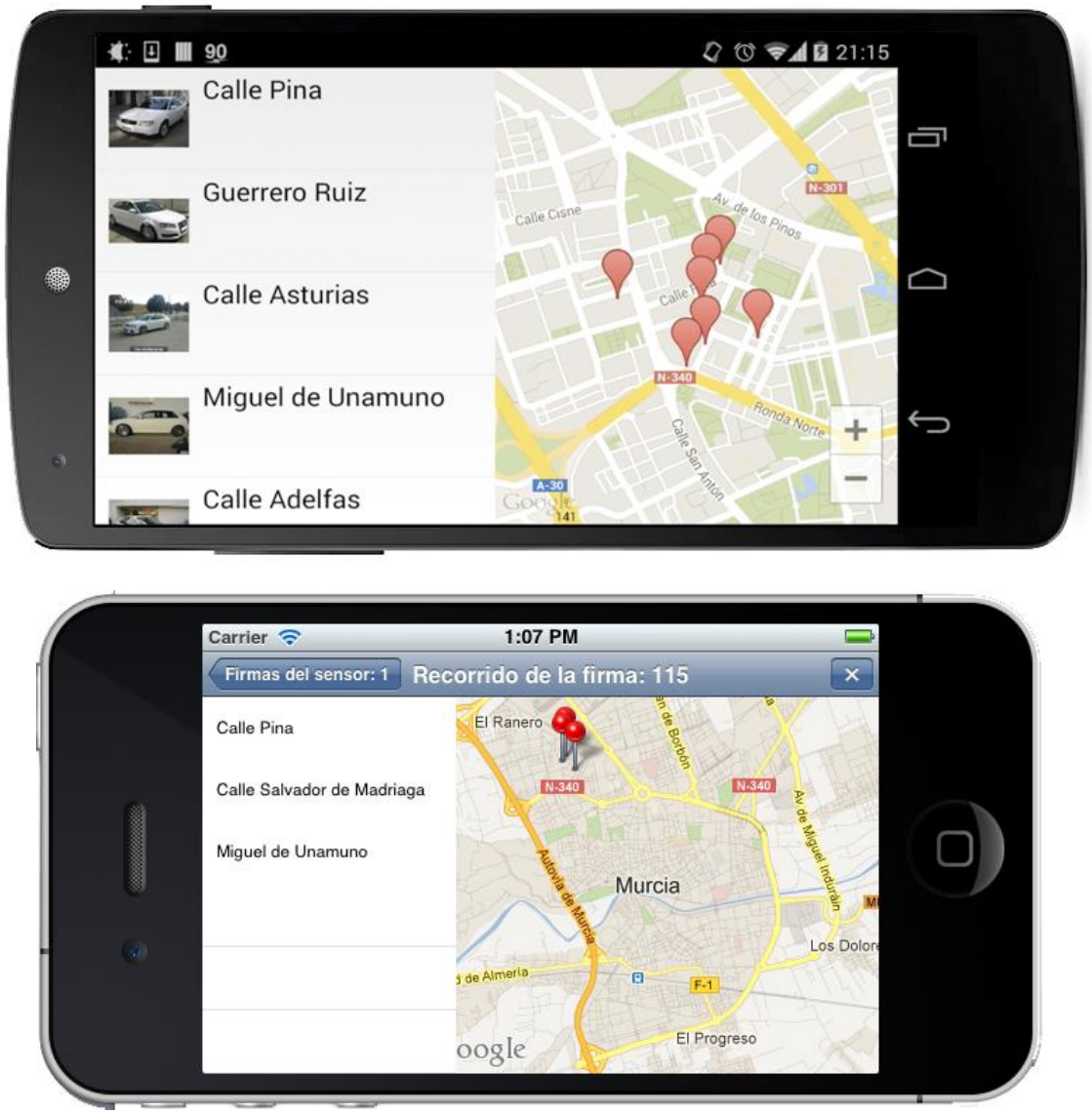


Figura 6. Vista del recorrido de una firma

Aquí el usuario verá actualizada en tiempo real la tabla del recorrido que va siguiendo el coche seleccionado.

Si presiona alguna de las calles de la tabla, tanto en la vista para seleccionar el sensor inicial, como en cualquiera de las vistas del recorrido de un vehículo, se seleccionará la chincheta asociada en el mapa, para facilitar la localización del vehículo.

A partir de éste momento, puede volver atrás para seleccionar otro vehículo, u otro punto de partida, o si el usuario ha terminado de realizar su tarea, también puede cerrar la aplicación.

Se aprecia que en ambos sistemas operativos se ha decidido reflejar un Look & Feel similar, facilitando así el uso de la aplicación y siendo intuitivo para el usuario el uso del mismo sistema, con independencia del sistema operativo móvil que esté utilizando en ese momento.

El resto de programas y piezas de software no poseen interacción por parte del usuario.

Son procesos automáticos que se intercambian información sin necesidad de agentes externos, se supone que es un sistema estable.

Ya que esa parte del sistema, está enfocada a ser modificada por un administrador que conozca la plataforma, no es necesario realizar un manual de uso, por la amplia labor técnica que ello supone.

4. Descripción de la implementación

4.1. Esquema general

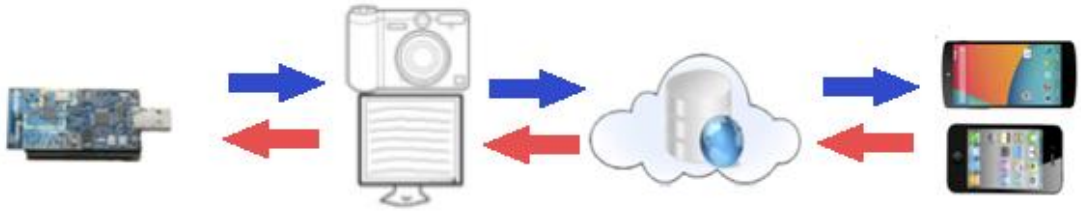


Figura 4. Jerarquía de intercambio de información entre los diferentes elementos del sistema

En la imagen se puede observar cual es el ámbito de interacción de cada uno de los elementos que conforman el sistema, definiendo y acotando los límites de visibilidad e intercambio de información entre ellos.

4.2. Gestor de los nodos sensores

Como bien se ha dicho anteriormente, el programa que ejecutan los equipos encargados de gestionar los nodos sensores, es un programa diseñado en Java.

Lo primero que hace este programa al arrancar es comprobar si el equipo que lo pone en funcionamiento ya estaba registrado en la base de datos, para eso le pide el nombre de la calle. En caso afirmativo pasa al estado de "esperar conexión":

```
Introduzca el nombre de su calle
Miguel de Unamuno
Miguel de Unamuno está seguro?
1:SI    2:NO
1
Esperando una conexión:
```

Figura 5. Mensaje inicial de la aplicación de los equipos gestores

En caso contrario, se pide la información necesaria y una vez introducida inserta sus datos en la tabla de la base de datos titulada "*coordenadas*":

```
Introduzca el nombre de su calle
Calle Cartagena
Calle Cartagena está seguro?
1:SI    2:NO
1
PC no registrado
Introduzca latitud
37.995
Introduzca longitud
-1.139392
Esperando una conexión:
```

Figura 6. Menú para registrar un nuevo equipo gestor en el sistema

En el estado de "Esperando una conexión" hay un socket a la espera de recibir una conexión, encargado de recibir a los clientes que se conectan.

Cuando un usuario se conecta, se envía a un nuevo hilo de procesamiento, permitiendo así que el sistema no se bloquee, y multitud de usuarios puedan utilizarlo al mismo tiempo.

En este punto puede empezar a recibir también mensajes desde los nodos sensores, correspondientes a las firmas de los vehículos, las cuáles son mediciones del campo magnético creado por el objeto metálico (en este caso el coche) a lo largo de dos ejes. La interpretación final de una firma magnética es un array de enteros.

Este nuevo hilo es el encargado de leer el mensaje que recibe del cliente conectado.

El gestor puede recibir cinco tipos de mensaje:

- *Sensor:X*
- *FirmaTx:Y*
- *FirmaSelec:X|Y*
- *FirmaRx:X*
- *CorrelacionOK:X*

- El primer tipo sirve para avisar a los equipos gestores del sensor escogido por el usuario (policía selecciona la calle a monitorizar), el argumento que lo acompaña es un ID único de tipo entero. Cada equipo tiene su ID que se le asigna cuando se registra en el sistema.

  El equipo gestor comprueba si el ID recibido es el suyo, en este caso se crea una tabla en la base de datos con el título "*firmasX*" donde X coincide con el ID del equipo.

  Después de eso, el equipo envía la orden a su conjunto de nodos sensores para que empiecen a analizar los coches que pasan sobre ellos.

- El mensaje *FirmaTx:Y* es enviado por el conjunto de nodos sensores hacia el equipo gestor cada vez que identifican un nuevo coche que circula sobre ellos. En ese momento, el equipo capta una foto con la imagen, y actualiza la tabla "*firmasX*" con la información de la firma "Y".

- *FirmaSelec:X|Y* es el tipo de mensaje que se envía a los equipos gestores cuando un usuario (policía que detecta a un vehículo infractor) selecciona una firma de la tabla correspondiente. Los argumentos que acompañan a la cabecera son:

    - "X" es la ID del sensor que posee la tabla de firmas correspondiente.
    - "Y" es el identificador de la firma seleccionada.

  Cuando el equipo que posee el mismo ID la recibe, crea una nueva tabla titulada "*recorridoY"* que contendrá la información de los equipos gestores que identifiquen esa firma en concreto. Después de crearla, inserta su ID como primer punto de recorrido, y añade a éste primer punto la imagen correspondiente del coche que posee la firma "Y".

  Aquí envía una señal a su conjunto de nodos sensores para que dejen de analizar los coches que circulan sobre ellos, pues el usuario (policía patrulla) ya está interesado en uno concreto.

- Cuando el usuario selecciona una firma, hay que informar al resto de nodos sensores de que deben comenzar a comparar las firmas que captan con la firma seleccionada, para ello se utiliza el mensaje "*FirmaRx:Y"* y la información de la firma "Y" se envía a nuestro conjunto de nodos sensores.

- Cuando nuestro conjunto de nodos sensores confirma que la firma seleccionada ha pasado por ellos, envía al gestor un mensaje con el formato: "*CorrelacionOK:X*".

  En el momento que el equipo gestor recibe ese mensaje, introduce su ID en la tabla "*recorridoX"* informando de que es el último sensor que lo ha localizado.
  La estructura de los archivos que componen este programa es la siguiente:

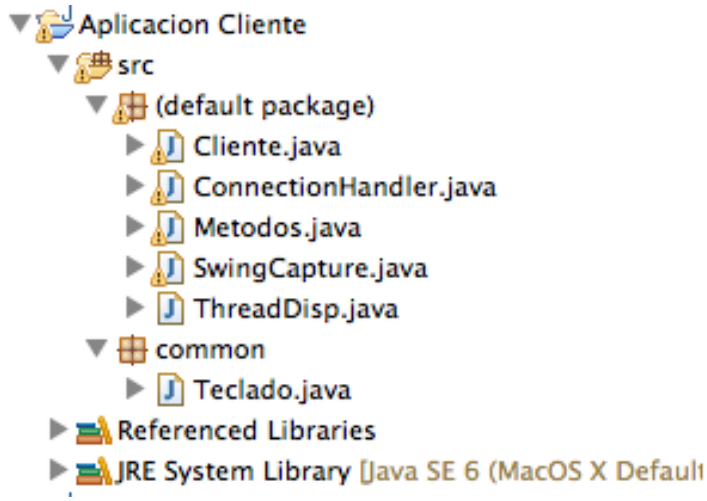


Figura 7. Estructura de los archivos del programa para los equipos gestores

Este programa está pensado para ejecutarse en equipos que tengan un bajo consumo, pues la batería debería durar bastante tiempo para no tener que estar reponiendo miles de calles en una ciudad cuando la energía de estas se agotase.

Otra opción sería conectarlos a la red eléctrica, para ello habría que conseguir los permisos correspondientes, ésta es la opción más favorable, ya que sería inviable reemplazar cada batería que se agotase en cada uno de los gestores registrados del panorama.

Un posible hardware capaz de realizar ésta función podría ser "Raspberry Pi" . Este dispositivo es un SBC que se puso a la venta hace dos años.

La hoja de especificaciones del dispositivo es la siguiente:

	Modelo A	Modelo B
Precio:[5]	$25	$35
SoC:[5]	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM)	
CPU:	ARM1176JZF-S a 700 MHz	
GPU:	Broadcom VideoCore IV,[26] OpenGL ES 2.0, decodificador 1080p30 H.264	
Memoria (SDRAM):	256 MiB (compartidos con la GPU)	
Puertos USB 2.0:	1	2 (vía hub USB integrado)
Salidas de vídeo:[5]	Conector RCA, HDMI	
Salidas de audio:[5]	3.5 mm jack, HDMI	
Almacenamiento integrado:	SD / MMC / ranura para SDIO	
Conectividad de red:[5]	Ninguna	10/100 Ethernet (RJ45)
Periféricos de bajo nivel:	Pines GPIO, SPI, I²C, UART[26]	
Reloj en tiempo real:[5]	Ninguno	
Consumo energético:	500 mA, (2.5 W)[5]	700 mA, (3.5 W)
Fuente de alimentación:[5]	5 V vía Micro USB o GPIO header	
Dimensiones:	85.60mm × 53.98mm[27] (3.370 × 2.125 inch)	
Sistemas operativos soportados:	Debian GNU/Linux, Fedora, Arch Linux[2]	
Sistemas operativos no soportados:	RISC OS[3] (shared source)	

Figura 8. Hoja de especificaciones del dispositivo Raspberry Pi

Como se puede apreciar, es un hardware más que suficiente para hacer funcionar esta aplicación y manejar una cámara.

Se podría pensar que éste dispositivo es mucho más potente que lo que requiere este programa, y en éste sentido se estarían desaprovechando recursos disponibles, no obstante, al ser un dispositivo que se ha hecho bastante popular, supone un bajo coste, debido a su fabricación en masa.

Diseñar un dispositivo específico para realizar estas labores, podría suponer un recorte de costes a la hora de utilizar componentes menos potentes, pero el coste en labores de producción no sería tan asumible.

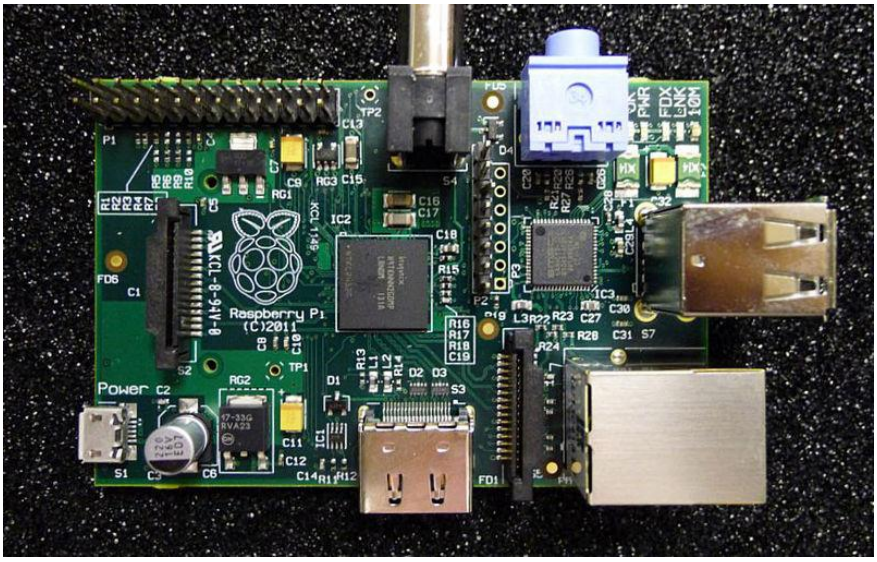


Figura 9. Imagen del dispositivo Raspberry Pi

4.3. Servidor web y base de datos

4.3.1. Servidor web

Las interacciones de la aplicación móvil con los nodos sensores tienen lugar pasando primero por una serie de scripts PHP alojados en un servidor web.

De hecho, las aplicaciones móviles no tienen visibilidad directa con los equipos gestores, únicamente se comunican con éste servidor web, como se aprecia en el apartado "3.1. Esquema general"

Las funciones principales de los scripts del servidor web son:

- Consultar la información de equipos gestores y firmas coches alojada en la base de datos.
- Enviar los mensajes dependientes del usuario a los equipos gestores de los conjuntos de nodos sensores.
- Codificar la información almacenada en la base de datos, y enviarla a la aplicación móvil correspondiente.

En total hay seis scripts y un fichero con funciones comunes:

- *"index.php"*
- *"chooseSensor.php"*
- *"txFirmas.php"*
- *"chooseFirma.php"*
- *"compararFirma.php"*
- *"refreshRecorrido.php"*
- *"funciones.php"*

- El primer fichero (*"index.php"*) es consultado por la aplicación móvil cuando el usuario presiona sobre "Comenzar" de la aplicación, en ese momento, el script se encarga de consultar a la base de datos la información referente a los equipos gestores registrados en el sistema, se muestra la información codificada en JSON en el contenido BODY de la petición HTTP response.

- El segundo fichero, "*chooseSensor.php"* espera una petición del tipo POST con el contenido "sensor=X" , es el encargado de generar el mensaje tipo "*Sensor:X"* y enviarlo a los equipos gestores para informar de que el usuario ha seleccionado un equipo gestor.

  Para enviar paquetes a los equipos gestores, el script se conecta al socket abierto en el programa de los equipos con la función "fsockopen".

```
$fp = fsockopen($row["ip"], 5000, $errno, $errstr, 30);
```

Figura 10. Ejemplo de la función fsockopen

- El tercero, "*txFirmas.php"* se ejecuta cuando el usuario ha seleccionado un gestor, y al igual que el anterior espera un paquete POST con el sensor correspondiente.

  Tiene como objetivo consultar la información contenida en la base de datos sobre los vehículos registrados por los nodos sensores pertenecientes a ese gestor. Codifica esa información en JSON en elementos del tipo "*Firma*" y la envía al usuario en otro HTTP response.

```
class Firma{
   var $id;
   var $imagen;
   var $numero;

   function __construct($id,$imagen,$numero){
        $this->id = $id;
        $this->imagen = $imagen;
        $this->numero = $numero;
   }
}
```

Figura 11. Estructura de los elementos tipo Firma

```
while($row = mysql_fetch_array($result)) {
        $aux[$i]=new Firma($row["id"],$row["imagen"],$row["numero"]);
        $i++;
   }
        echo $aux=json_encode($aux);
```

Figura 12. Parte del código donde se crean las firmas y se devuelve el resultado.

- El cuarto, "*chooseFirma.php"* es el encargado de crear los mensajes tipo "*FirmaSelec:X|Y"* que se envían a los equipos gestores. Es ejecutado en el momento que el usuario selecciona una firma de las que aparecen en la tabla del sensor "X".

- "*compararFirma.php"* se ejecuta después de "*chooseFirma.php"*, su función es la de enviar los mensajes con el formato: "*FirmaRx:X"* a los equipos gestores para informar que deben comenzar a examinar sus firmas recogidas para ver si el coche infractor figura entre ellas.

- El último, "*refreshRecorrido.php*" tiene como función principal obtener la información de los equipos gestores que han introducido su ID en la tabla "recorridoX", pues esto indica que el coche infractor ha pasado por ellos en el orden que aparece en la tabla.

Codifica esa información en elementos del tipo "*PRecorrido*" los introduce en una cadena JSON y los envía al usuario.

Hay una tabla recorrido para cada una de las firmas seleccionadas, así que este script espera como argumento un número de firma, que será el correspondiente a la tabla que muestre.

```php
class PRecorrido{
   var $latitud;
   var $longitud;
   var $nombre;
   var $imagen;

   function __construct($latitud,$longitud,$nombre,$imagen){
        $this->latitud = $latitud;
        $this->longitud = $longitud;
        $this->nombre = $nombre;
        $this->imagen = $imagen;
   }
}
```

Figura 13. Elemento del tipo PRecorrido

```php
$firma = $_POST["firma"];
$query ="select * from coordenadas,recorrido".$firma." where";
$query .="coordenadas.id=recorrido".$firma.".punto ORDER BY recorrido".$firma.".id ASC";
$result=mysql_query($query,$link);
```

Figura 14. Parte del código donde se realiza la consulta SQL

- El fichero "*funciones.php*" lleva principalmente la función para conectar los scripts a la base de datos y para poder realizar la petición SQL.

4.3.2. Base de datos

Las tablas que contienen la base de datos tienen la siguiente estructura:

- Tabla "coordenadas":

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	☑	☑	☐	☐	☐	☐	☑	
Nombre	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	
Latitud	DOUBLE	☐	☑	☐	☐	☐	☐	☐	
Longitud	DOUBLE	☐	☑	☐	☐	☐	☐	☐	
ip	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	

Figura 15. Estructura de la tabla coordenadas

Los campos tienen la siguiente función:

- "id" es un identificador único (número entero) que se autoincrementa en uno por cada nuevo elemento que se inserta. Tiene como finalidad, ordenar los objetos en la tabla.

- "Nombre" es una cadena de texto donde se almacena el nombre de la calle a registrar introducida por el equipo gestor.

- "Latitud" y "Longitud" son número dobles (con decimales) que representan las coordenadas geográficas de la nueva calle que se ha registrado.

- "ip" es el campo donde se almacena la dirección IP del equipo gestor que ha registrado la calle, y sirve para permitir al resto del sistema poder comunicarse con ese equipo en concreto.

Un posible ejemplo de ésta tabla con sus datos contenidos es el siguiente:

id	Nombre	Latitud	Longitud	ip
1	Miguel de Una...	37.995914	-1.139705	localhost
2	Calle Pina	37.995433	-1.140143	localhost
3	Calle Moncayo	37.99499	-1.140361	localhost
4	Calle Moncayo 2	37.993918	-1.139392	localhost
5	Calle Salvador...	37.994588	-1.138543	localhost

Figura 16. Ejemplo de la tabla coordenada rellena con datos

Ésta es la tabla donde viene a parar la información de los equipos gestores que se registran en el sistema antes de aceptar conexiones.

El valor de la ip se asigna automáticamente cuando el equipo se registra.

- Tabla "firmasX":

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	☑	☑	☐	☐	☐	☐	☑	
imagen	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
numero	INT(11)	☐	☑	☐	☐	☐	☐	☐	

Figura 17. Estructura de la tabla firmas

Ésta tabla se encarga de guardar elementos del tipo firma, los cuales están compuestos por:

- "id" identificador único que sirve para ordenar el elemento en la tabla.

- “imagen” la ruta hacia una imagen capturada por la cámara del equipo gestor correspondiente, que sirve para identificar gráficamente al vehículo que posee la firma.

- “numero” corresponde a la firma del vehículo, que se consigue modelando el campo magnético que genera, y dándole un formato numérico.

Un ejemplo de la tabla con datos:

id	imagen	numero
1	http://localhost/imagenes/a.jpg	115
2	http://localhost/imagenes/b.jpg	113
3	http://localhost/imagenes/c.jpg	200
4	http://localhost/imagenes/d.jpc	300

Figura 18. Ejemplo de la tabla firmas rellena con datos

Esta tabla contiene la información de los coches que pasan por un sensor determinado.

Las direcciones que aparecen en la captura comienzan por “localhost” porque al momento de escribir la memoria, todas las pruebas se simularon sobre un entorno local.

- Tabla “recorridoX”:

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(11)	☑	☑	☐	☐	☐	☐	☑	
punto	INT(11)	☐	☑	☐	☐	☐	☐	☐	
imagen	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	

Figura 19. Estructura de la tabla recorrido

Pasamos a explicar los campos que la componen:

- “id” al igual que en las tablas anteriores, un identificador único que sirve para ordenar los elementos que se insertan en ésta tabla.

- “punto” es un campo, donde el equipo gestor que inserta un nuevo elemento del tipo “objetoRecorrido” va a insertar el valor de su campo “id” haciendo así referencia a cual es el equipo gestor que ha introducido sus coordenadas en el recorrido por donde ha pasado el vehículo infractor.

- “imagen” es el campo en donde el equipo gestor insertará la ruta a la caputra que realiza cuando detecta al vehículo infractor pasando por su conjunto de nodos sensores.

La tabla con datos de relleno:

id	punto	imagen
1	2	http://localhost/imagenes/a.jpg
2	5	http://localhost/imagenes/b.jpg
3	1	http://localhost/imagenes/c.jpg

Figura 20. Ejemplo de la tabla recorrido rellena con datos

A esta tabla acceden los equipos gestores que identifican el coche objetivo con una de las firmas que acaban de analizar, por tanto, inserta su ID en el campo punto, y la imagen que su cámara ha captado del vehículo.

4.4. Aplicación Android

El diseño de la aplicación móvil para Android es quizá la parte más elaborada del proyecto.

Cuando en este apartado hago referencia a una "vista" de la aplicación me estoy refiriendo a lo que en Android se denomina "actividad", el cambio se debe a que Android/iOS lo llaman de manera diferente, y estoy más acostumbrado a la definición de iOS.

Cuenta con la siguiente estructura de archivos:

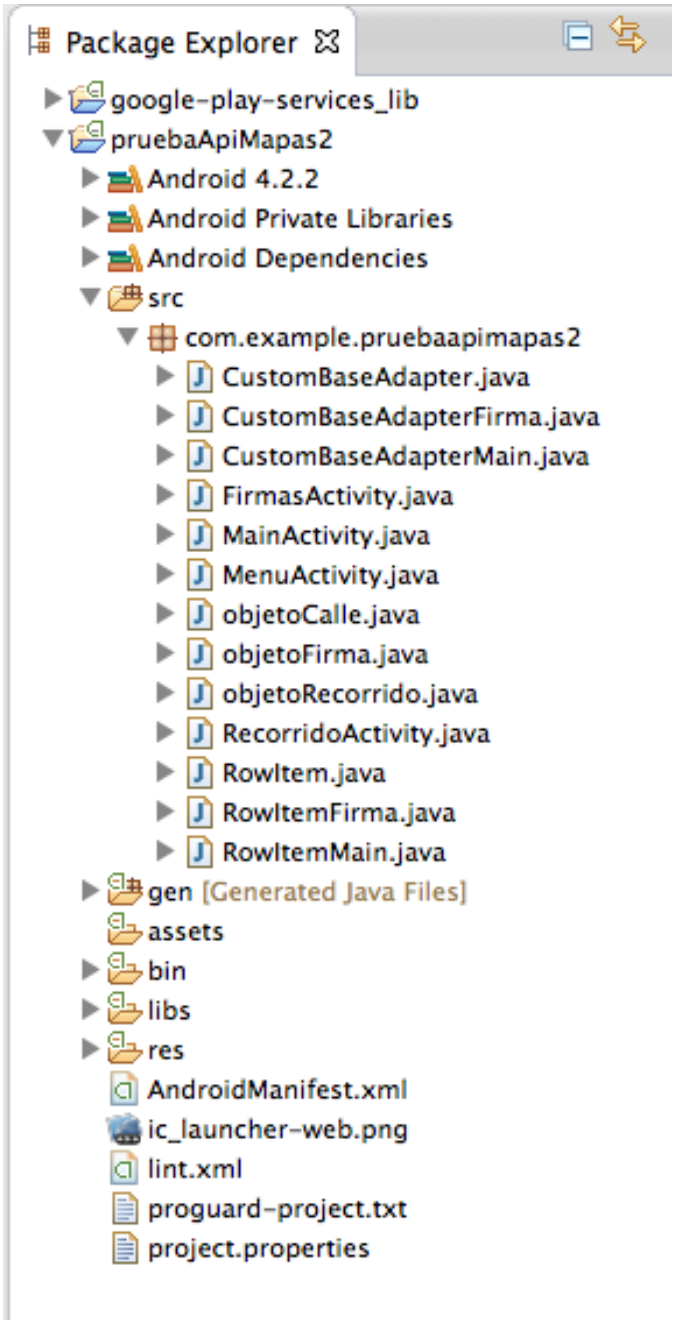


Figura 21. Estructura de la aplicación para Android

Podemos apreciar claramente que la base de Android es Java examinando únicamente la extensión de los archivos utilizados en el proyecto.

Al igual que en un proyecto Java, el código fuente del programa va en la carpeta src del proyecto, dentro de uno o varios paquetes que pueden servir para ordenar nuestros ficheros de clase.

El SDK de Android inserta archivos adicionales que son requeridos para que la aplicación funcione correctamente, como el conjunto de librerías que podemos apreciar entre los tres primeros elementos de la lista, las clases "gen" que se generan de manera automática, la carpeta "res" donde definimos la interfaz gráfica de la aplicación, o el archivo "AndroidManifest.xml" donde definimos características tan importantes de la aplicación como los permisos que va a requerir en el terminal, se establece con que dispositivos va a ser compatible, etc.

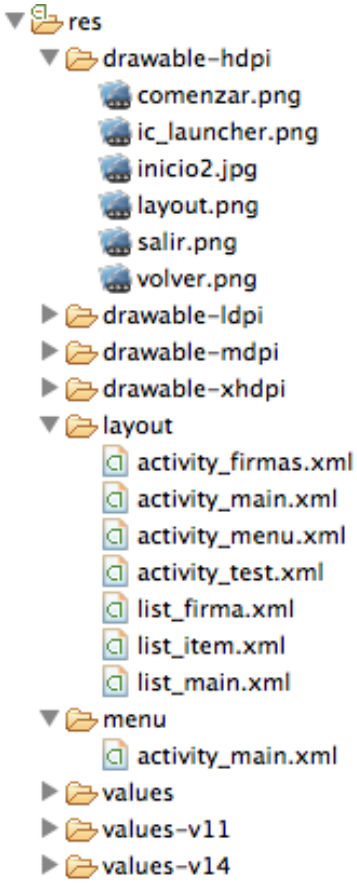


Figura 22. Estructura de la carpeta "res" del proyecto

Si no se tiene clara la estructura de archivos, recomiendo al lector buscar información de los componentes de un proyecto Android.

La clase "MenuActivity" es una clase muy sencilla que nos hace de introducción a la aplicación, será la pantalla de bienvenida que nos permite poner el sistema en funcionamiento, o salir de ésta en caso de que no sea necesario comenzar el proceso.

```java
package com.example.pruebaapimapas2;

import android.app.Activity;

public class MenuActivity extends Activity implements OnClickListener{

    private Button comenzar;
    private Button salir;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);

        comenzar = (Button) findViewById(R.id.button1);
        salir = (Button) findViewById(R.id.button2);

          comenzar.setOnClickListener(new OnClickListener() {
              public void onClick(View v) {
                 Intent intent = new Intent(MenuActivity.this, MainActivity.class);
                 startActivity(intent);
              }
           });

          salir.setOnClickListener(new OnClickListener() {
              public void onClick(View v) {
                 finish();
              }
           });
    }

    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub

    }

}
```

Figura 23. Código de la vista "MenuActivity"

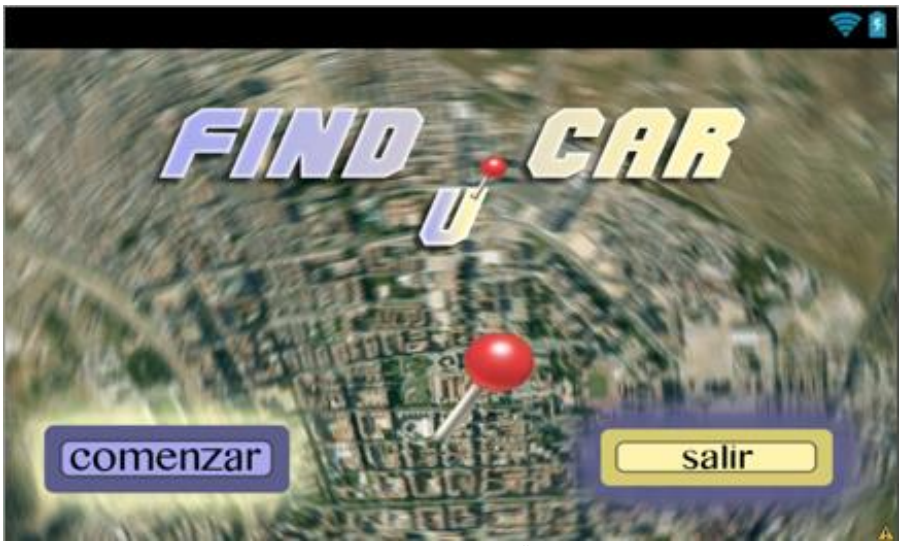


Figura 24. Layout de la vista "MenuActivity"

Cuando se comienza el proceso, se arranca una activity con nombre "MainActivity" que se encarga de consultar al servidor las calles registradas en el sistema.

```
100    private class MainAsyncTask extends AsyncTask<Void, Integer, String>{
101
102        @Override
103        protected String doInBackground(Void... params) {
104            HttpClient httpClient = new DefaultHttpClient();
105            HttpGet del = new HttpGet(url2);
106            del.setHeader("content-type", "application/json");
107
108            try
109            {
110                HttpResponse resp = httpClient.execute(del);
111                cadenaJSON = EntityUtils.toString(resp.getEntity());
112            }
113            catch(Exception ex)
114            {
115                Log.e("ServicioRest","Error!", ex);
116                cadenaJSON = "[{\"latitud\":\"0\",\"longitud\":\"0\",\"nombre\":\"Err
117            }
118            Log.d("getResponse",cadenaJSON);
119            return cadenaJSON;
120        }
121
122        protected void onProgressUpdate(Integer... values) {
124
126        protected void onPreExecute() {
128
129        protected void onPostExecute(String cadenaJSON) {
134
136        protected void onCancelled() {
138
139    }
140
141    private void decodeCallesJSON(){
176
177    private void rellenarLista(){
193
194    private void colocarCoordenadas(){
```

Figura 25. Código principal de "MainActiivty"

Se aprecia la tarea asíncrona que se ejecuta en segundo plano encargada de realizar la consulta al servidor, después de esto, un método encargado de dar formato a los datos obtenidos del servidor "decodeCallesJSON()".

Una vez que tenemos los datos con la estructura adecuada, se llama a la función que se encargará de volcar éstos datos en la lista de la parte izquierda de la aplicación, y también se llamará al método que se encarga de dibujar las chinchetas correspondientes en el mapa "colocarCoordenadas()"

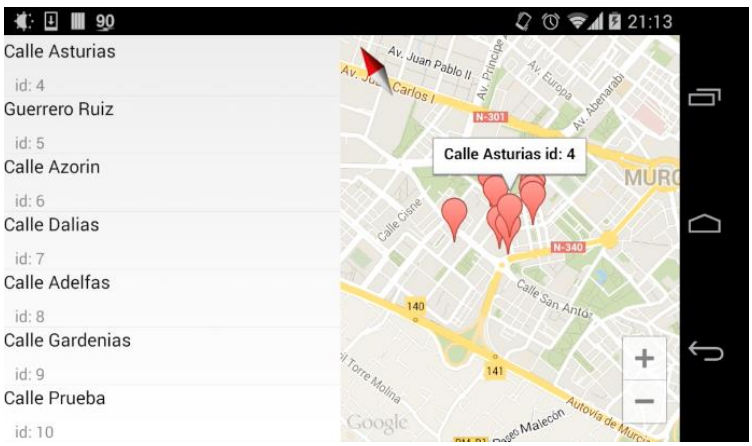


Figura 26. Layout de la vista "MainActiivty"

Cuando el usuario selecciona uno de los elementos de la vista, se ejecuta el método "onItemClickListener()" definido de la siguiente manera:

```
@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
    // TODO Auto-generated method stub
    Intent i = new Intent(contexto, FirmasActivity.class );
    i.putExtra("nombre", calles[arg2].getTitle());
    i.putExtra("id", calles[arg2].getId());
    i.putExtra("url",url);
    startActivity(i);
}
```

Figura 27. Método "onItemClickListener" de la clase "MainActivity"

La función se encarga de crear un elemento del tipo "FirmasAcitivty" y pasarle los argumentos de: "nombre" de la calle seleccionada, "id" identificador único de ésta y "url" para hacer saber a la aplicación cual es la IP del servidor para poder realizar consultas.

Paso ahora a describir la siguiente vista de nuestra aplicación "FirmasActivity". La tarea principal de ésta se puede apreciar en la siguiente captura:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_firmas);

    listView = (ListView) findViewById(R.id.list);
    textView = (TextView) findViewById(R.id.textView);

    handler = new Handler();

    firmasAnteriores = new objetoFirma[0];

     new ChooseSensorTask().execute();

    tarea = new TimerTask(){
        @Override
        public void run() {

            handler.post(new Runnable(){
                public void run() {
                    new TxFirmasTask().execute();
                };
            });
        }
    };
```

Figura 28. Tarea principal de FirmasActivity

Figura 29. Layout de la vista "FirmasActivity"

Se aprecia que ésta vista es la encargada de comunicar a la red de sensores el sensor escogido por el usuario con el método "ChooseSensorTask()"

```
private class ChooseSensorTask extends AsyncTask<Void, Integer, String>{

    @Override
    protected String doInBackground(Void... params) {
        ArrayList<NameValuePair> argumentos = new ArrayList<NameValuePair>(2);
        argumentos.add(new BasicNameValuePair("sensor",""+sensor));

                //envio de datos al servicio web
                try{
                    HttpClient httpclient = new DefaultHttpClient();
                    HttpPost httppost = new HttpPost(url+"/promos/chooseSensor.php");
                    httppost.setEntity(new UrlEncodedFormEntity(argumentos));
                    HttpResponse response = httpclient.execute(httppost);
                    HttpEntity entity = response.getEntity();
                    entity.getContent();
                    Log.i("Connection made", response.getStatusLine().toString());
                }catch(Exception e){
                    Log.e("log_tag", "Error in http connection "+e.toString());
                }
                new TxFirmasTask().execute();

                return "OK";
    }

    protected void onProgressUpdate(Integer... values) {

    @Override
    protected void onPreExecute() {
        textView.setText("Cargando...");
        Bundle bundle=getIntent().getExtras();
        nombreSensor = bundle.getString("nombre");
        sensor = bundle.getInt("id");
        url = bundle.getString("url");
        Log.e("TOSTRING",nombreSensor+" "+sensor+" "+url);

    }

    protected void onPostExecute(String estado) {
        contador = 0;
        contador2 = 0;

    }
```
Figura 30. Extracto de código encargado de comunicar a la red de sensores el sensor escogido

Una vez comunicada la elección del usuario, comienza a ejecutarse de manera periódica la transmisión de las firmas que reconoce el sensor escogido, el encargado de ésta tarea es "TxFirmasTask()":

```
private class TxFirmasTask extends AsyncTask<Void, StringBuilder, String>{

    @Override
    protected String doInBackground(Void... params) {
        ArrayList<NameValuePair> argumentos = new ArrayList<NameValuePair>(2);
        argumentos.add(new BasicNameValuePair("sensor",""+sensor));

        StringBuilder sb;
        InputStream is = null;

                //envio de datos al servicio web
                try{
                    HttpClient httpclient = new DefaultHttpClient();
                    HttpPost httppost = new HttpPost(url+"/promos/txFirmas.php");
                    httppost.setEntity(new UrlEncodedFormEntity(argumentos));
                    HttpResponse response = httpclient.execute(httppost);
                    HttpEntity entity = response.getEntity();
                    is = entity.getContent();
                    Log.i("Connection made", response.getStatusLine().toString());

                }catch(Exception e){
                    Log.e("log_tag2", "Error in http connection "+e.toString());
                }

                //recepcion de datos del servicio web
                 try {
                    BufferedReader reader = new BufferedReader(new InputStreamReader(is, "iso-8859-1"), 8);
                    sb = new StringBuilder();
                    sb.append(reader.readLine() + "\n");
                    String line = "0";
                    while ((line = reader.readLine()) != null) {
                        sb.append(line + "\n");
                        publishProgress();
                    }
                    is.close();
                    FirmasJSON = sb.toString();
                }
                 catch (Exception e) {
                    Log.e("log_tag3", "Error converting result " + e.toString());
                    FirmasJSON = "[{\"id\":\"0\",\"imagen\":\"ERROR\",\"numero\":\"0\"}]";
                }
                Log.e("handlerFirmas",FirmasJSON);
                return FirmasJSON;
```

Figura 31. Extracto de código encargado de transmitir las firmas detectadas.

En esta clase encontramos también los métodos encargados de decodificar los datos obtenidos del servidor, y de realizar de manera asíncrona la descarga de imágenes asociadas a las firmas detectadas.

```
private void decodeFirmasJSON(String cadenaJSON){..

private void obtenerImagenes(){..

public Bitmap getBitmapFromURL(String imageUrl) {..

private class ObtenerImagenesTask extends AsyncTask<String, Integer, String>{..
```

Figura 32. Métodos adicionales de FirmasActivity

Al igual que en la vista anterior, cuando el usuario selecciona uno de los elementos de la lista, se ejecuta el método de "onItemClickListener()":

```
@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,

        long arg3) {
    // TODO Auto-generated method stub
    tarea.cancel();

    Intent i = new Intent(this, RecorridoActivity.class );
    i.putExtra("sensor",sensor);
    i.putExtra("url",url);
    Log.d("PosicionTabla",""+arg2);
    Log.d("firmaAEnviar",""+firmas[arg2].getNumero());
    i.putExtra("firma",firmas[(firmas.length-1)-arg2].getNumero());
    startActivity(i);

}
```

Figura 33. Método onClickListener de FirmasActivity

Crea un elemento del tipo "RecorridoActivity" y le pasa los valores de "sensor" escogido por el usuario y la "url" que contiene la IP del servidor.
El valor del sensor escogido se utilizará para determinar el primer punto del recorrido realizado por el coche objetivo.

La última de las vistas de nuestra aplicación es la vista "RecorridoActivity" la función principal de ésta es la siguiente:

```
new chooseFirmaTask().execute();

TimerTask tarea = new TimerTask(){
   @Override
   public void run() {

       handler.post(new Runnable(){
           public void run() {
               new refreshRecorridoTask().execute();
           };
       });
   }
};
```

Figura 34. Extracto de código de RecorridoActivity

Al igual que la vista anterior, tiene asociada dos tareas, una encargada de comunicar al sistema la firma escogida por el usuario "chooseFirmaTask()":

```
private class chooseFirmaTask extends AsyncTask<Void, Integer, String>{

    @Override
    protected String doInBackground(Void... params) {

        ArrayList<NameValuePair> argumentos = new ArrayList<NameValuePair>(2);
        argumentos.add(new BasicNameValuePair("firma",""+firma));
        argumentos.add(new BasicNameValuePair("sensor",""+sensor));

                //envio de datos al servicio web
                try{
                    HttpClient httpclient = new DefaultHttpClient();
                    HttpPost httppost = new HttpPost(url+"/promos/chooseFirma.php");
                    Log.e("TOSTRING",url+"/promos/chooseFirma.php"+"sensor: "+sensor+" firma: "+firma);
                    httppost.setEntity(new UrlEncodedFormEntity(argumentos));
                    HttpResponse response = httpclient.execute(httppost);
                    HttpEntity entity = response.getEntity();
                    entity.getContent();
                    Log.i("Connection made", response.getStatusLine().toString());

                }catch(Exception e){
                    Log.e("log_tag", "Error in http connection "+e.toString());
                    return "error";
                }
                return "ok";
    }

    protected void onProgressUpdate(Integer... values) {

    protected void onPreExecute() {

    protected void onPostExecute(String cadenaJSON) {
        contador = 0;
        contador2 = 0;
        //compararFirma();
        new refreshRecorridoTask().execute();
    }
```

Figura 35. Extracto de código de chooseFirmaTask()

Una vez transmitida esa información, se realiza de manera periódica la tarea de descargar desde el servidor y su base de datos, la información asociada al recorrido realizado por el vehículo. Para ello se utiliza la tarea "refreshRecorridoTask()":

```
private class refreshRecorridoTask extends AsyncTask<Void, StringBuilder, String>{

    @Override
    protected String doInBackground(Void... params) {

        ArrayList<NameValuePair> argumentos = new ArrayList<NameValuePair>(2);
        argumentos.add(new BasicNameValuePair("firma",""+firma));

        StringBuilder sb;
        InputStream is = null;

                //envio de datos al servicio web
                try{
                    HttpClient httpclient = new DefaultHttpClient();
                    HttpPost httppost = new HttpPost(url+"/promos/refreshRecorrido.php");
                    httppost.setEntity(new UrlEncodedFormEntity(argumentos));
                    HttpResponse response = httpclient.execute(httppost);
                    HttpEntity entity = response.getEntity();
                    is = entity.getContent();
                    Log.i("Connection made", response.getStatusLine().toString());

                }catch(Exception e){
                    Log.e("log_tag", "Error in http connection "+e.toString());
                }

                //recepcion de datos del servicio web
                 try {
                    BufferedReader reader = new BufferedReader(new InputStreamReader(is, "iso-8859-1"), 8);
                    sb = new StringBuilder();
                    sb.append(reader.readLine() + "\n");
                    String line = "0";
                    while ((line = reader.readLine()) != null) {
                        sb.append(line + "\n");
                        publishProgress(sb);
                    }
                    is.close();
                    recorridoJSON = sb.toString();
```

Figura 36. Extracto de código de chooseFirmaTask()

En el código de la imagen superior, se puede observar cómo se realiza la llamada al servicio web para refrescar los datos del recorrido solicitado.

Además de las tareas descritas, en esta clase aparecen también funciones adicionales para realizar tareas necesarias para el correcto funcionamiento de la aplicación, como el formateo de los datos recibidos desde el servidor para convertirlos en datos útiles con el método "decodeRecorridoJSON()", colocar las chinchetas en el mapa con "rellenarMapaJSON()" o el método necesario para obtener las imágenes asociadas al coche seleccionado conforme va siendo detectado por el resto de sensores de la red.

Los equipos gestores mandan esa información a la base de datos, y nosotros la obtenemos con "obtenerImagenes()".

```
private void decodeRecorridoJSON(String cadenaJSON){
private void rellenarMapaJSON(){
private void obtenerImagenes(){
public Bitmap getBitmapFromURL(String imageUrl) {
private class ObtenerImagenesTask extends AsyncTask<String, Integer, String>{
```

Figura 37. Funciones adicionales de la vista "RecorridoActivity"

La aplicación requiere que, en diferentes vistas, aparezca una lista de elementos que no son únicamente una etiqueta de texto, si no que pueden tener varias etiquetas, e incluso, una imagen por cada uno de éstos elementos.

Para realizar esa implementación en Android, he utilizado los elementos llamados "adaptadores" que se encargan de obtener una lista de elementos del tipo "ArrayList" he insertarlos en la vista correspondiente.

Así nos aparecen las clases "CustomBaseAdapter" y "RowItem" que definirán las listas de nuestra aplicación.

```
public class CustomBaseAdapterFirma extends BaseAdapter {
    Context context;
    List<RowItemFirma> rowItems;

    public CustomBaseAdapterFirma(Context context, List<RowItemFirma> items) {
        this.context = context;
        this.rowItems = items;
    }
```

Figura 38. Instanciación de elemento "CustomBaseAdapter"

Los adaptadores definen la lógica de nuestra lista, por eso asociaremos éste código a un elemento gráfico del tipo "ListView" que esté presente en alguna de las vistas, ya que al existir varios tipos de listas, tendremos que definir cual es el adaptador que rige cada una.

```
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder = null;

    LayoutInflater mInflater = (LayoutInflater)
        context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.list_firma, null);
```

Figura 39. Asociación código con interfaz gráfica

Comprobamos cómo nuestro elemento "CustomBaseAdapter" se asocia con los elementos gráficos del tipo "list_firma".

El adaptador recogerá elementos de una determinada clase, y les implantará una lógica dentro de un elemento gráfico, en este caso, si nos remitimos a la figura 41 podemos comprobar que los elementos recogidos por nuestro adaptador son del tipo "RowItemFirma".

```
public class RowItemFirma {
    private Bitmap bitmap;
    private String title;

    public RowItemFirma(Bitmap bitmap, String title) {
        this.bitmap = bitmap;
        this.title = title;
    }
```

Figura 40. Definición de "RowItemFirma"

Las clases "objetoCalle", "objetoFirma" y "objetoRecorrido" son clases intermedias utilizadas para realizar la traducción del formato recibido desde el servidor (JSON) a tipos de objeto que nos resultan útiles a la hora de utilizar los datos en nuestra aplicación.

Con esto quedaría descrito el funcionamiento principal de la aplicación para Android, sin entrar en profundidad en aspectos como el diseño de la interfaz gráfica, o la solicitud de permisos para utilizar las API's de servicios de Google, como en este caso, que ha sido necesario pedir permiso para obtener acceso a la API de Google Maps, ya que describir estos procesos en profundidad se sale fuera del dominio de éste proyecto.

Para esta última tarea, es necesario solicitar acceso a la consola de desarrolladores de Google, y solicitar una clave gratuita para realizar pruebas.

Se puede encontrar una gran cantidad de información sobre estos procesos en la documentación oficial destinada a ello en la página para desarrolladores de Android.

4.5. Aplicación iOS

Aunque existe el desarrollo de la aplicación para el sistema operativo de los dispositivos móviles de Apple, la descripción de ésta implementación queda fuera de éste proyecto, ya que se realizó una explicación en profundidad en otro proyecto realizado por el mismo autor titulado “Desarrollo de un Sistema de Seguimiento de Vehículos para Dispositivos iOS basado en el Control por Redes de Sensores”.

4.6. Interacciones entre los elementos del sistema

Se procede a realizar un esquema visual del sistema, con el intercambio de mensajes que se produce en cada momento.

En primer lugar, los equipos gestores de los nodos sensores, ejecutan su software para comprobar si están registrados en la base de datos.

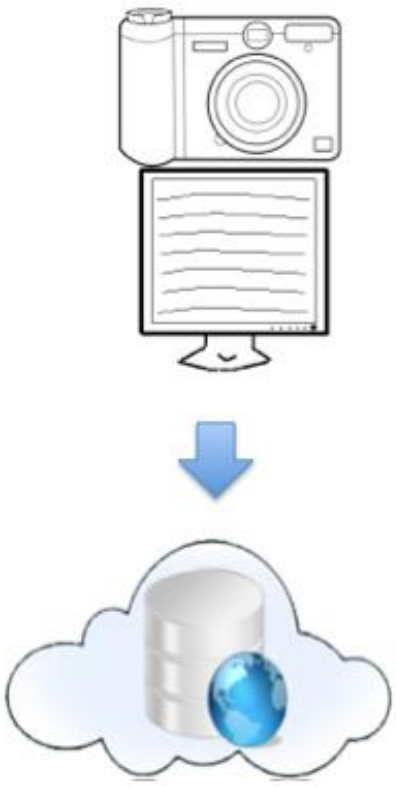


Figura 41. Interacción 1

El servidor comprueba el nombre introducido en la tabla "coordenadas" de la base de datos:

id	Nombre	Latitud	Longitud	ip
1	Miguel de Una...	37.995914	-1.139705	localhost
2	Calle Pina	37.995433	-1.140143	localhost
3	Calle Moncayo	37.99499	-1.140361	localhost
4	Calle Moncayo 2	37.993918	-1.139392	localhost
5	Calle Salvador...	37.994588	-1.138543	localhost

Figura 42. Tabla coordenadas

En caso que exista una entrada con el mismo nombre, se intuye que está ya está registrada, y pasa a estado "*Esperando una conexión*".

Si esa calle no ha sido dada de alta, solicita al interlocutor que introduzca la localización de ese gestor, y acto seguido pasa a estado "*Esperando una conexión*".

No vuelve a haber ninguna interacción hasta que un usuario (policía patrulla) inicie la aplicación, en ese momento, la aplicación solicita al servidor la información de la tabla "coordenadas".

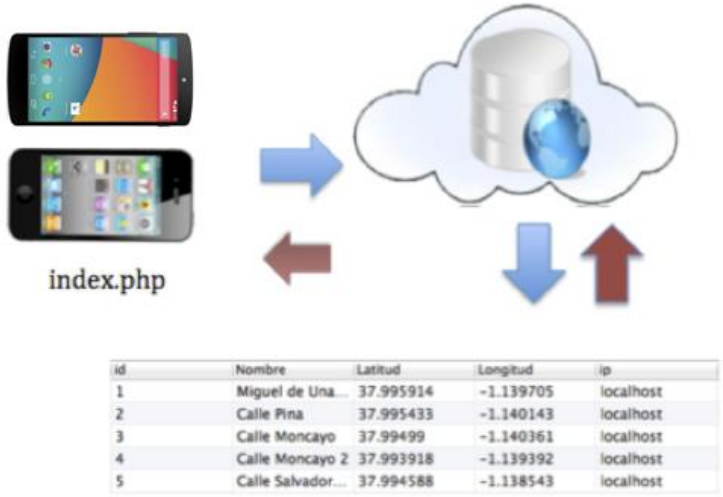


Figura 43. Interacción 2

En la aplicación se representa la información de la tabla "coordenadas".

Cuando el usuario decide desde que equipo comienza la búsqueda, se transmite al servidor ese dato, y éste lo retransmite a los equipos gestores.

El equipo gestor objetivo, indica a sus nodos sensores que comiencen a almacenar las firmas de los vehículos que circulen sobre ellos.

En este momento también se crea la tabla "firmasX" en la base de datos.

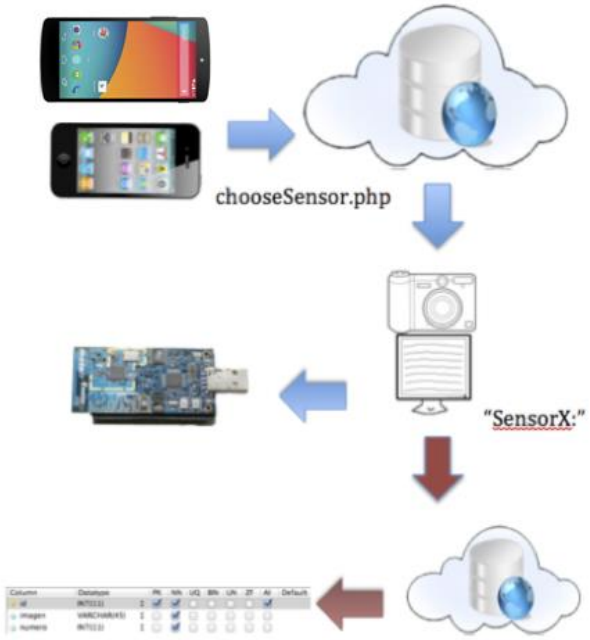


Figura 44. Interacción 3

Cuando los nodos sensores almacenan alguna firma, se la envían al equipo gestor, y cuando el equipo la recibe, captura una foto del vehículo con su cámara.

Toda esa información se almacena como una nueva entrada de la tabla “firmasX”.

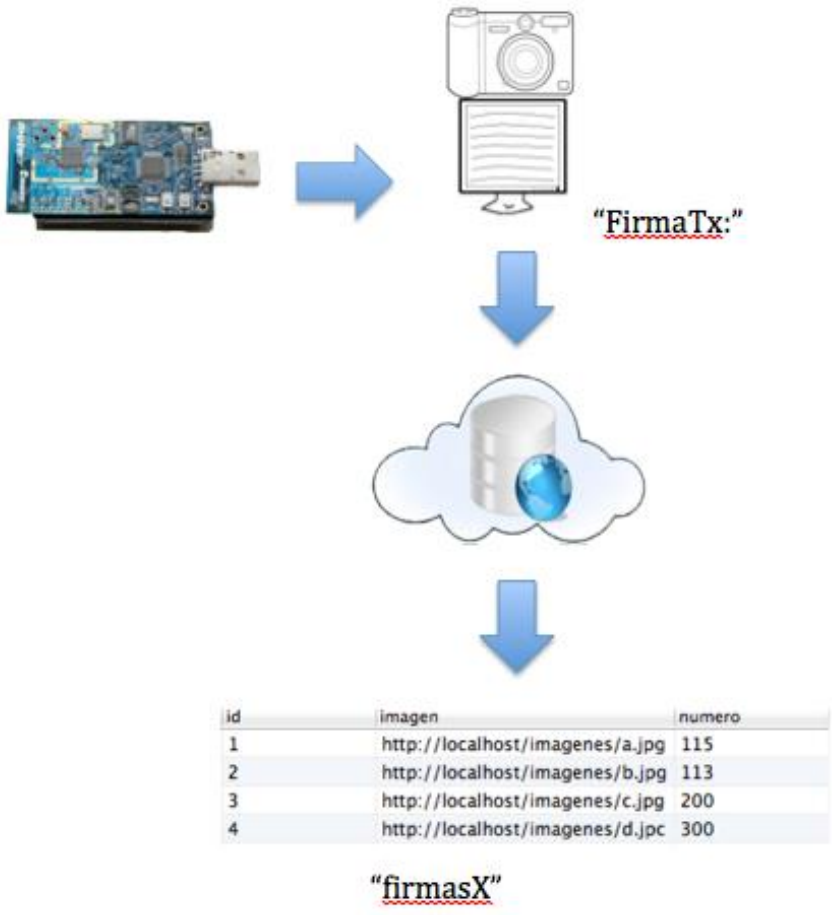


Figura 45. Interacción 4

Se realizan de manera periódica las consultas al script que envía las firmas almacenadas en la tabla “firmasX” (donde X es un entero que se envía tipo POST) hasta la aplicación.

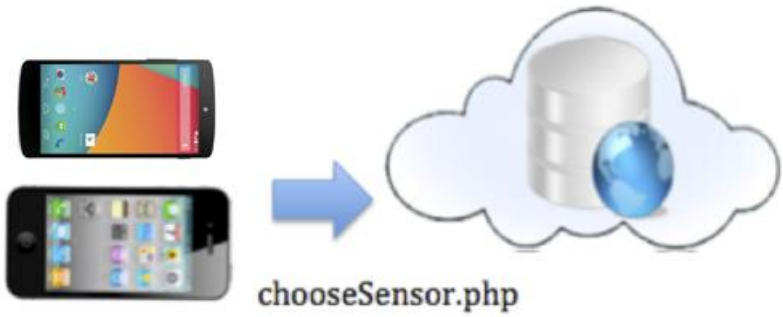


Figura 46. Interacción 5

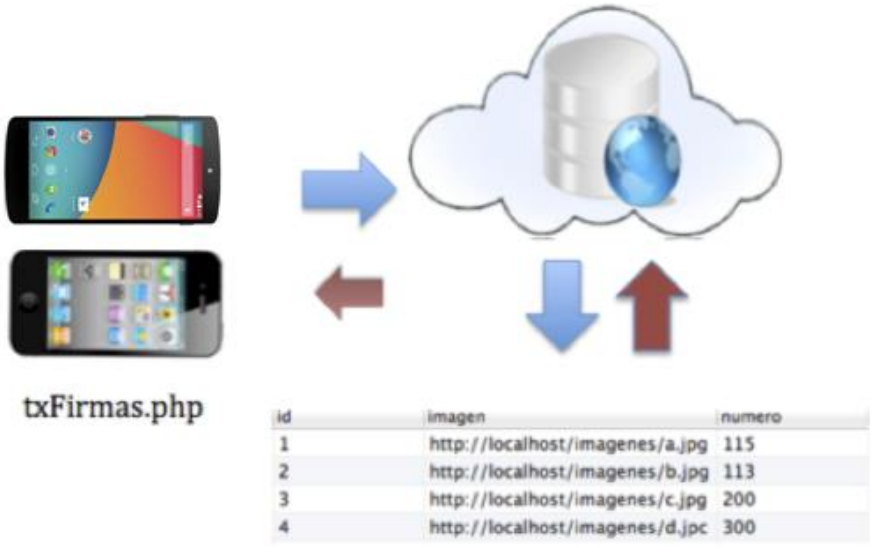


Figura 47. Interacción 6

Cuando el usuario selecciona una de las firmas de la tabla, primero se ejecuta un script que tiene como objetivo detener el almacenamiento de las firmas capturadas por los nodos sensores es éste equipo.

En este punto se crea la tabla "recorridoX" y se introducen nuestros datos como primer punto del circuito.

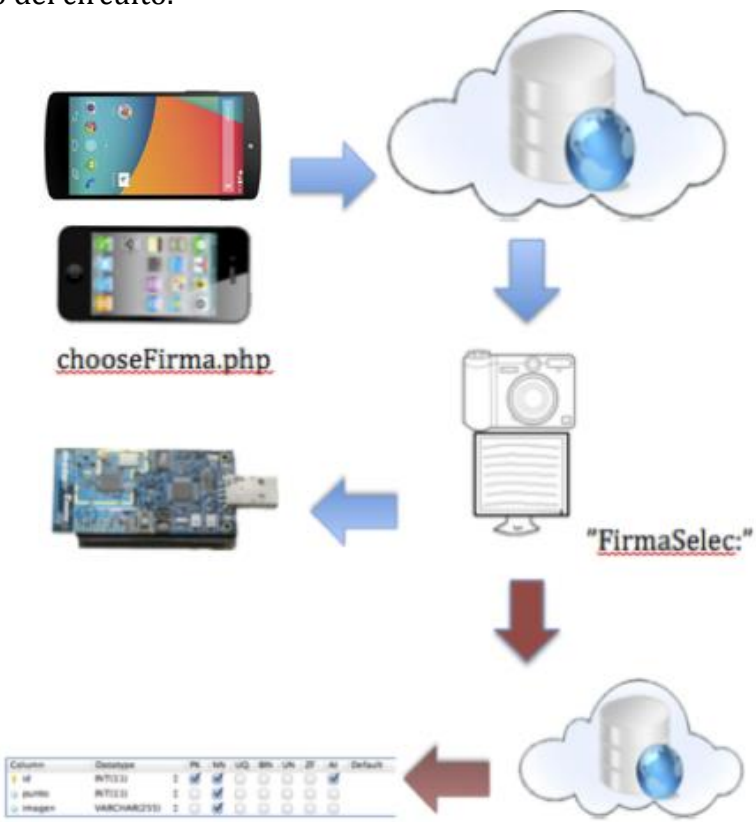


Figura 48. Interacción 7

Después de esto, la aplicación ejecuta otro script, que afecta al resto de equipos gestores, cuya función es enviar los datos de la firma seleccionada, para que la comparen con las firmas que están capturando en ese momento.

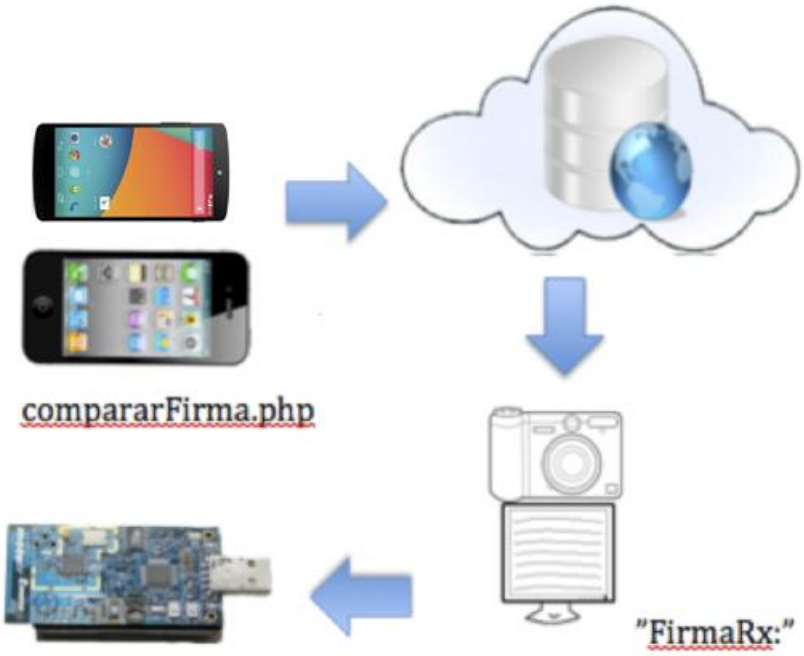


Figura 49. Interacción 8

Cuando los nodos sensores identifican un vehículo similar al que tienen que comparar, avisan a este gestor con el mensaje "CorrelaciónOK:X". En ese momento se captura una imagen del vehículo, y se inserta esa información en la tabla "recorridoX" localizada en la base de datos.

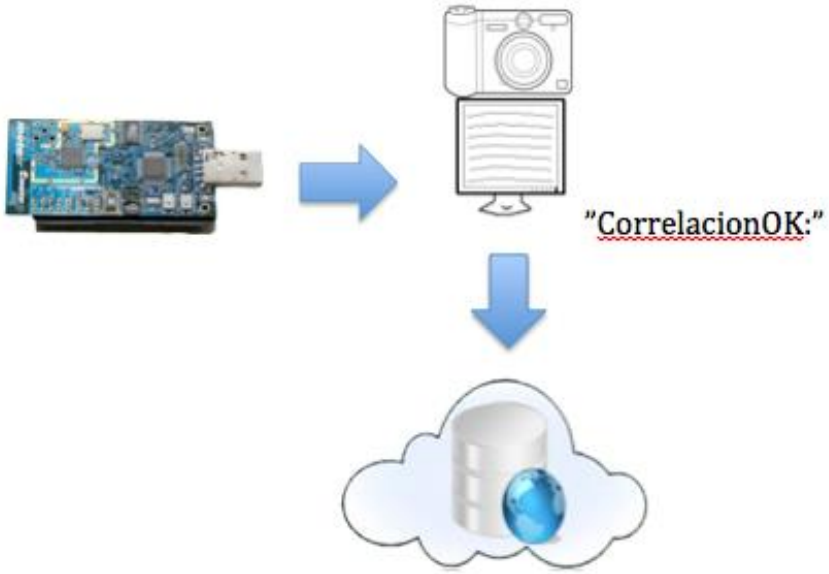
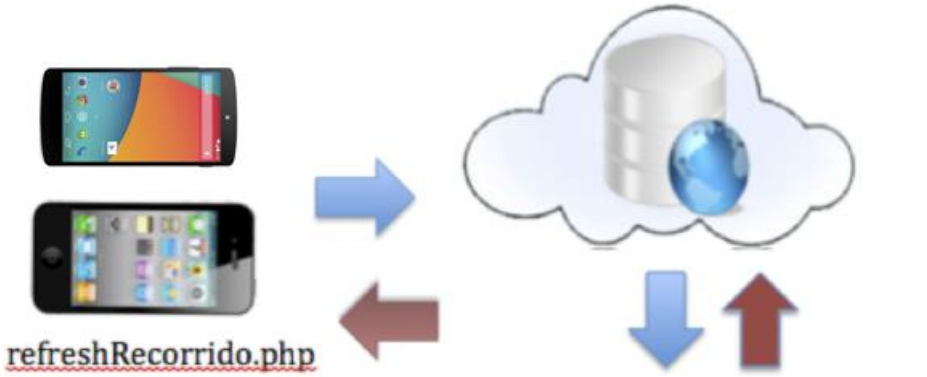


Figura 50. Interacción 9

Por último, la última vista de la aplicación ejecuta repetidamente el script que le devuelve la información de la tabla "recorridoX", hasta que se pulsa el botón finalizar, o se vuelve atrás.



id	punto	imagen
1	2	http://localhost/imagenes/a.jpg
2	5	http://localhost/imagenes/b.jpg
3	1	http://localhost/imagenes/c.jpg

Figura 51. Interacción 10

5. Conclusión y ampliaciones

El objetivo inicial del proyecto, era la completa implementación del sistema descrito, en un entorno reducido para poder demostrar su correcto comportamiento. Para ello, se debe unir el desarrollo, junto con el proyecto realizado en esta escuela por José Manuel López Egea, titulado "Estudio E Implementación De Un Sistema De Seguimiento De Vehículos Con Una Red De Nodos sensores Inalámbrica".

Para exponer el funcionamiento de mi parte, se ha diseñado un pequeño script, que simula a los nodos sensores conectados a uno de los equipos gestores[1]. Así se envían los mensajes que el equipo espera recibir, y se puede visualizar el funcionamiento de la aplicación.

Cabe destacar, el cambio tan grande que ha realizado Google con la API de su servicio Google Maps, ya que cuando se comenzó el proyecto, se comenzó utilizando la versión 1 de ésta, y con la llegada de la versión 2, toda la metodología y el código que hacía referencia a ésta ha quedado inservible, por tanto se ha debido redefinir todas las interacciones con los mapas, conllevando un esfuerzo que no se tenía contemplado al comenzar el desarrollo del proyecto.

También se ha corregido otra funcionalidad de las aplicaciones móviles, causante de que a la hora de recargar las tablas de datos, no comprobaba si los datos a insertar eran los que estaban ya introducidos, esto ahora queda corregido, necesitando así un menor esfuerzo por parte de la aplicación, lo que conlleva un ahorro de batería y memoria RAM.

Se ha depurado el código de los scripts para cumplir con los estándares de la nueva filosofía en el desarrollo de web services REST.
Para obtener más información sobre REST, diríjase al punto 7.2 de la memoria.

Finalizando con el análisis del proyecto, respondo a las cuestiones que quedan en el aire:

- **¿Es un proyecto viable?**

La completa instalación es una posibilidad de cara a un futuro algo lejano, cuando la producción en masa de los nodos sensores y el hardware necesario para los equipos gestores disminuyan su precio drásticamente por unidad, mientras tanto, se puede implementar en zonas acotadas como en la entrada y salida de una autovía, vías principales ó entrada y salida de un parking.

Desconozco si la parte del modelado del campo magnético, se podría realizar con un equipamiento hardware más barato, ya que hace tres años que se comenzó éste desarrollo e intuyo que habrán salido al mercado productos capaces de realizar ésta función. Una vía a estudiar, sería la posibilidad de implementarlo con una placa del proyecto "Arduino" y algún sensor del campo magnético, debido al bajo coste de éste, aunque ya digo, que desconozco la viabilidad de ésta modificación, y la repercusión que tendría en el consumo eléctrico el cambio de los sensores,

seguramente habría que optar por la solución de conectar los equipos a la red eléctrica.

Quizá, a raíz de esto si surja otro proyecto más viable de cara al instante actual. En todo caso, este proyecto sigue siendo un claro ejemplo de la importancia que va obteniendo el mundo de las aplicaciones móviles, y la utilidad de trabajar con información almacenada en la red junto con la demostración de una comunicación entre un servidor y un dispositivo móvil, así como la interoperabilidad entre los distintos sistemas operativos actuales de éstos dispositivos.

- **¿Se le ocurre alguna ampliación al sistema?**

Se me ocurren varias, en primer lugar, ésta aplicación al ser de carácter policial, debe asegurar cierta seguridad, y en esta implementación carece de ella, por tanto, ya que el usuario sólo interactúa con el servidor, se podría poner un sistema de identificación, que en caso de no poseer el ID y la contraseña correspondiente, no le permitiese acceder a ninguno de los archivos del servidor.

Mejorando ésta ampliación, también se podría usar HASH para cifrar el contenido de los mensajes que se intercambian entre los dispositivos, para así evitar que posibles atacantes obtuvieran o modificasen la información que circula por el sistema, potencialmente peligrosa para un usuario común.

Por último, no es una ampliación propiamente dicha, pero si una opción a estudiar, se podrían hacer las modificaciones pertinentes sobre el sistema y la aplicación, para aplicarlo en otros campos, por ejemplo:

Podría resultar útil implementar un sistema así en una amplia cadena de montaje, donde un supuesto supervisor, podría controlar las piezas metálicas por las distintas secciones del circuito, el estado de cada una de las cintas, etc. Sabiendo que la influencia de los smartphones es cada vez mas frecuente, y al estar toda la información del sistema en la red centralizada, los empleados encargados de éstas funciones pudieran tener el proceso controlado de un solo vistazo mientras realizan otras labores.

1. Las imágenes usadas para la simulación han sido obtenidas desde Google imágenes.

6. Bibliografía

[1] Cómo programar en Java-, 2a Edición y siguientes. Prentice Hall, 2004.

[2] WILLIAMS, H.; LANE, D. Web database applications with PHP & MySQL. 2ª ed. Sebastopol: O'Reilly, 2004.

[3] BecomeAnXcoder, Cocoalab, 2008

[4] iOS Developer Library

[5] Android Developers: http://developer.android.com/

[6] Curso de programación en Android, Salvador Gómez Oliver, 2014. http://www.sgoliver.net

[7] Wikipedia - http://es.wikipedia.org

[8] Losilla, F.; Garcia-Sanchez, A.-J.; Garcia-Sanchez, F.; Garcia-Haro, J.; Haas, Z.J. A Comprehensive Approach to WSN-Based ITS Applications: A Survey. Sensors 2011, 11, 10220-10265.

7. Anexos

7.1. Lenguajes utilizados en el proyecto

Como se puede ver en el ranking de los lenguajes de programación más utilizados del año pasado:

Position Oct 2013	Position Oct 2012	Delta in Position	Programming Language	Ratings Oct 2013	Delta Oct 2012	Status
1	1	=	C	17.246%	-2.58%	A
2	2	=	Java	16.107%	-1.09%	A
3	3	=	Objective-C	8.992%	-0.49%	A
4	4	=	C++	8.664%	-0.60%	A
5	6	↑	PHP	6.094%	+0.43%	A
6	5	↓	C#	5.718%	-0.81%	A
7	7	=	(Visual) Basic	4.819%	-0.30%	A
8	8	=	Python	3.107%	-0.79%	A
9	23	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.621%	+2.13%	A
10	11	↑	JavaScript	2.038%	+0.78%	A
11	18	↑↑↑↑↑↑↑	Visual Basic .NET	1.933%	+1.33%	A
12	9	↓↓↓	Perl	1.607%	-0.52%	A
13	10	↓↓↓	Ruby	1.246%	-0.56%	A
14	14	=	Pascal	0.753%	-0.09%	A
15	17	↑↑	PL/SQL	0.730%	+0.10%	A
16	13	↓↓↓	Lisp	0.725%	-0.22%	A
17	12	↓↓↓↓↓	Delphi/Object Pascal	0.701%	-0.40%	A
18	53	↑↑↑↑↑↑↑↑↑↑	Groovy	0.658%	+0.53%	B
19	19	=	MATLAB	0.614%	+0.02%	B
20	26	↑↑↑↑↑↑	COBOL	0.599%	+0.15%	B

Figura 52. Índice Tiboe 2013

Si la comparamos con la situación existente cuando se comenzó el desarrollo del proyecto (la cual se puede visualizar en la siguiente imagen) los lenguajes escogidos para el desarrollo han sido bastante acertados, ya que conforme han pasado los años, han ocupado un mejor hueco en el ranking, notándose así el incremento que han sufrido.

No cabe duda que gran parte de la culpa es de los dispositivos móviles, en especial Android e iOS, los cuáles han impulsado Java y Objective-C de una manera bastante considerable.

Posición Dic 2011	Posición Dic 2010	Variación en la posición	Lenguaje de programación	Porcentaje Dic 2011	Variación Dic 2010
1	1	=	Java	17.561%	-0.44%
2	2	=	C	17.057%	+0.98%
3	3	=	C++	8.252%	-0.76%
4	5	↑	C#	8.205%	+1.52%
5	8	↑↑↑	Objective-C	6.805%	+3.56%
6	4	↓↓	PHP	6.001%	-1.51%
7	7	=	(Visual) Basic	4.757%	-0.36%
8	6	↓↓	Python	3.492%	-2.99%
9	9	=	Perl	2.472%	+0.14%
10	12	↑↑	JavaScript	2.199%	+0.69%
11	11	=	Ruby	1.494%	-0.29%
12	10	↓↓	Delphi/Object Pascal	1.245%	-0.93%
13	13	=	Lisp	1.175%	+0.11%

Figura 53. Índice Tiboe 2011

7.1.1. Java

Java es un lenguaje de programación de alto nivel orientado a objetos, desarrollado por James Gosling en 1995. El lenguaje en sí mismo toma mucha de su sintaxis de C, Cobol y Visual Basic, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. La memoria es gestionada mediante un recolector de basura.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre aunque la biblioteca de clases de paginas web comprendidas en las librerias de objetacion de objetos para ser compilados como aplicaciones comprimidas no están totalmente acopladas de acuerdo con Sun que dice que se requiere un interprete para ejecutar los programas de Java.

Historia

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada *the Green Project* en Sun Microsystems en el año 1991. El equipo (*Green Team*), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente *Oak* (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse *Green* tras descubrir que *Oak* era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a *Java*.

Es frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: ***J**ames Gosling, **A**rthur **V**an Hoff, y **A**ndy Bechtolsheim*. Otros abogan por el siguiente acrónimo, ***J**ust **A**nother **V**ague **A**cronym* ("sólo otro acrónimo ambiguo más"). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana, de ahí que el icono de java sea una taza de café caliente. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el *número mágico*) de los archivos.class que genera el compilador, son en hexadecimal, 0xCAFEBABE. A pesar de todas estas teorías, el nombre fue sacado al parecer de una lista aleatoria de palabras.[1]

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gaga, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, de que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el

grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. [1] Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era *Write Once, Run Anywhere* (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.[2]

Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa *Java Specification Requests* (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la *Java Language Specification* (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

Además de los cambios en el lenguaje, con el paso de los años se han efectuado muchos más cambios dramáticos en la biblioteca de clases de Java (*Java class library*) que ha crecido de unos pocos cientos de clases en JDK 1.0 hasta más de tres mil en J2SE 5.0. APIs completamente nuevas, como Swing y Java2D, han sido introducidas y muchos de los métodos y clases originales de JDK 1.0 están obsoletas.

En el 2005 se calcula en 4,5 millones el número de desarrolladores y 2.500 millones de dispositivos habilitados con tecnología Java.

Filosofía

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar el paradigma de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA (Common

Object Request Broker Architecture), Internet Communications Engine o OSGi respectivamente.

Orientado a objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

Independencia de la plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "'write once, run anywhere"'.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ''dependientes'' de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o '''compilación al vuelo'''), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una '''recompilación dinámica''' en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria.

La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en cualquier parte” por “Escríbelo una vez, depúralo en todas partes”)

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

El recolector de basura

En Java el problema de las fugas de memoria se evita en gran medida gracias a la recolección de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos y mayor seguridad.

Entornos de funcionamiento

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

En dispositivos móviles y sistemas empotrados

Desde la creación de la especificación J2ME (Java 2 Platform, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de

consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos.

El modelo de desarrollo de estas aplicaciones es muy semejante a las *applets* de los navegadores salvo que en este caso se denominan MIDlets.

Véase Sun Mobile Device Tecnology

En el navegador web

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (*plug-in*) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

El éxito de este tipo de aplicaciones (la visión del equipo de Gosling) no fue realmente el esperado debido a diversos factores, siendo quizás el más importante la lentitud y el reducido ancho de banda de las comunicaciones en aquel entonces que limitaba el tamaño de las applets que se incrustaban en el navegador. La aparición posterior de otras alternativas (aplicaciones web dinámicas de servidor) dejó un reducido ámbito de uso para esta tecnología, quedando hoy relegada fundamentalmente a componentes específicos para la intermediación desde una aplicación web dinámica de servidor con dispositivos ubicados en la máquina cliente donde se ejecuta el navegador.

Las *applets* Java no son las únicas tecnologías (aunque sí las primeras) de componentes complejos incrustados en el navegador. Otras tecnologías similares pueden ser: ActiveX de Microsoft, Flash, Java Web Start, etc.

En sistemas de servidor

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga

computacional o de memoria y propensión a errores por su interpretación dinámica).

Los servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es muy sencilla, flexible y extensible.

- Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.

- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks (pe Struts, Webwork) que se sobreponen sobre los servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la especialización de roles sea posible (desarrolladores, diseñadores gráficos,...) y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas.

Este modelo de trabajo se ha convertido en uno de los estándar *de-facto* para el desarrollo de aplicaciones web dinámicas de servidor.

En aplicaciones de escritorio

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

Plataformas soportadas

Una versión del entorno de ejecución Java JRE (Java Runtime Environment) está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de GNU/Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

JRE

El **JRE** (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

[1] Pagina web:
http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29

7.1.2. PHP

PHP es un lenguaje de programación interpretado o framework para HTML, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor (*server-side scripting*) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

PHP es un acrónimo recursivo que significa *PHP Hypertext Pre-processor* (inicialmente *PHP Tools*, o, *Personal Home Page Tools*). Fue creado originalmente por Rasmus Lerdorf en 1994; sin embargo la implementación principal de PHP es producida ahora por The PHP Group y sirve como el estándar de facto para PHP al no haber una especificación formal. Publicado bajo la PHP License, la Free Software Foundation considera esta licencia como software libre.

Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. El lenguaje PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores, el número de sitios en PHP ha compartido algo de su preponderante dominio con otros nuevos lenguajes no tan poderosos desde agosto de 2005. El sitio web de Wikipedia está desarrollado en PHP. Es también el módulo Apache más popular entre las computadoras que utilizan Apache como servidor web.

El gran parecido que posee PHP con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones.

Aunque todo en su diseño está orientado a facilitar la creación de sitios webs, es posible crear aplicaciones con una interfaz gráfica para el usuario, utilizando la extensión PHP-Qt o PHP-GTK. También puede ser usado desde la línea de órdenes, de la misma manera como Perl o Python pueden hacerlo; a esta versión de PHP se la llama PHP-CLI (*Command Line Interface*).

Cuando el cliente hace una petición al servidor para que le envíe una página web, el servidor ejecuta el intérprete de PHP. Éste procesa el script solicitado que generará el contenido de manera dinámica (por ejemplo obteniendo información de una base de datos). El resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente. Mediante extensiones es también posible la generación de archivos PDF, Flash, así como imágenes en diferentes formatos.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite.

PHP también tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos, tales como Unix (y de ese tipo, como Linux o Mac OS X) y Microsoft

Windows, y puede interactuar con los servidores de web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

PHP es una alternativa a las tecnologías de Microsoft ASP y ASP.NET (que utiliza C# y Visual Basic .NET como lenguajes), a ColdFusion de la empresa Adobe, a JSP/Java y a CGI/Perl. Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia GNU, existe además un entorno de desarrollo integrado comercial llamado Zend Studio. CodeGear (la división de lenguajes de programación de Borland) ha sacado al mercado un entorno de desarrollo integrado para PHP, denominado 'Delphi for PHP. También existen al menos un par de módulos para Eclipse, uno de los entornos más populares.[1]

Características

- Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- Es considerado un lenguaje fácil de aprender, ya que en su desarrollo se simplificaron distintas especificaciones, como es el caso de la definición de las variables primitivas, ejemplo que se hace evidente en el uso de php arrays.
- El código fuente escrito en PHP es invisible al navegador web y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Capacidad de expandir su potencial utilizando módulos (llamados *ext's* o extensiones).
- Posee una amplia documentación en su sitio web oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite aplicar técnicas de programación orientada a objetos. Incluso aplicaciones como Zend framework, empresa que desarrolla PHP, están totalmente desarrolladas mediante esta metodología.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Tiene manejo de excepciones (desde PHP5).
- Si bien PHP no obliga a quien lo usa a seguir una determinada metodología a la hora de programar, aun haciéndolo, el programador puede aplicar en su trabajo cualquier técnica de programación o de desarrollo que le permita escribir código ordenado, estructurado y manejable. Un ejemplo de esto son los desarrollos que en PHP se han hecho del patrón de diseño Modelo Vista Controlador (MVC), que permiten separar el tratamiento y acceso a los

datos, la lógica de control y la interfaz de usuario en tres componentes independientes.

Inconvenientes

- Como es un lenguaje que se interpreta en ejecución, para ciertos usos puede resultar un inconveniente que el código fuente no pueda ser ocultado. La ofuscación es una técnica que puede dificultar la lectura del código pero no necesariamente impide que el código sea examinado.
- Debido a que es un lenguaje interpretado, un script en PHP suele funcionar considerablemente más lento que su equivalente en un lenguaje de bajo nivel, sin embargo este inconveniente se puede minimizar con técnicas de cache tanto en archivos como en memoria.
- Las variables al no ser tipadas dificulta a los diferentes IDEs para ofrecer asistencias para el tipeado del código, aunque esto no es realmente un inconveniente del lenguaje en sí. Esto es solventado por Zend Studio añadiendo un comentario con el tipo a la declaración de la variable.

XAMPP, LAMP, WAMP, MAMP

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor Web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, MySQL, PHP, Perl. El programa está liberado bajo la licencia GNU y actúa como un servidor Web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP esta disponible para Microsoft Windows, GNU/Linux, Solaris, y MacOS X.

LAMP presenta una funcionalidad parecida a XAMP, pero enfocada en Linux, y WAMP lo hace enfocado en Windows.

Principales sitios desarrollados con PHP

PHP es utilizado en millones de sitios, entre los mas destacados se encuentran wikipedia.org, facebook.com y Wordpress.com.

[2]Página web: http://es.wikipedia.org/wiki/Php

7.1.3. Android

Android, es la excepción de la sección, ya que no es un lenguaje en sí mismo, es un sistema operativo, pero programar para éste sistema supone tantos cambios al lenguaje original (que en éste caso sería Java) que creo conveniente dedicar una sección independiente para este pequeño compañero:

Figura 54. "Andy" mascota del sistema.

Android es el sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, y también para relojes inteligentes, televisores y automóviles, inicialmente desarrollado por Android Inc., que Google respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles.10 El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008.

El éxito del sistema operativo se ha convertido en objeto de litigios sobre patentes en el marco de las llamadas «Guerras por patentes de teléfonos inteligentes» (en inglés *Smartphone patent wars*) entre las empresas de tecnología. Según documentos secretos filtrados en 2013 y 2014, el sistema operativo es uno de los objetivos de las agencias de inteligencia internacionales.

Historia

Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005 Es el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.10 Las unidades vendidas de teléfonos inteligentes con Android se ubican en el primer puesto en los Estados Unidos, en el segundo y tercer trimestres de 2010 con una cuota de mercado de 43,6% en el tercer trimestre. A escala mundial alcanzó una cuota de mercado del 50,9% durante el cuarto trimestre de 2011, más del doble que el segundo sistema operativo (iOS de Apple, Inc.)

Tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se ha llegado ya al 1.000.000 de aplicaciones (de las cuales, dos tercios son gratuitas y en comparación con la App Store más baratas) disponibles para la tienda de aplicaciones oficial de Android: Google Play

El anuncio del sistema Android se realizó el 5 de noviembre de 2007 junto con la creación de la Open Handset Alliance, un consorcio de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles. Google liberó la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto.

En julio de 2005, Google adquirió Android Inc., una pequeña compañía de Palo Alto, California fundada en 2003. Entre los cofundadores de Android que se fueron a trabajar a Google están Andy Rubin (co-fundador de Danger), Rich Miner (co-fundador de Wildfire Communications, Inc.), Nick Sears (alguna vez VP en T-Mobile), y Chris White (quien encabezó el diseño y el desarrollo de la interfaz en WebTV). En aquel entonces, poco se sabía de las funciones de Android Inc. fuera de que desarrollaban software para teléfonos móviles. Esto dio pie a rumores de que Google estaba planeando entrar en el mercado de los teléfonos móviles.

En Google, el equipo liderado por Rubin desarrolló una plataforma para dispositivos móviles basada en el núcleo Linux que fue promocionado a fabricantes de dispositivos y operadores con la promesa de proveer un sistema flexible y actualizable. Se informó que Google había alineado ya una serie de fabricantes de hardware y software y señaló a los operadores que estaba abierto a diversos grados de cooperación por su parte.

La especulación sobre que el sistema Android de Google entraría en el mercado de la telefonía móvil se incrementó en diciembre de 2006. Reportes de BBC y The Wall Street Journal señalaron que Google quería sus servicios de búsqueda y aplicaciones en teléfonos móviles y estaba muy empeñado en ello. Medios impresos y en línea pronto reportaron que Google estaba desarrollando un teléfono con su marca.

En septiembre de 2007, «InformationWeek» difundió un estudio de Evalueserve que reportaba que Google había solicitado diversas patentes en el área de la telefonía móvil.

Etimología

Tanto el nombre *Android* (androide en español) como Nexus One hacen alusión a la novela de Philip K. Dick *¿Sueñan los androides con ovejas eléctricas?*, que posteriormente fue adaptada al cine como *Blade Runner*. Tanto el libro como la película se centran en un grupo de androides llamados *replicantes* del modelo Nexus-6.

Seguridad, privacidad y vigilancia

Según un estudio de Symantec de 2013, demuestra que en comparación con iOS, Android es un sistema menos vulnerable. El estudio en cuestión habla de 13 vulnerabilidades graves para Android y 387 vulnerabilidades graves para iOS. El estudio también habla de los ataques en ambas plataformas, en este caso Android se queda con 113 ataques nuevos en 2012 a diferencia de iOS que se queda en 1 solo ataque. Aún así Google y Apple se empeñan cada vez más en hacer sus sistemas operativos más seguros incorporando más seguridad tanto en sus sistemas operativos como en sus mercados oficiales.

Como parte de las amplias revelaciones sobre vigilancia masiva filtradas en 2013 y 2014, se descubrió que las agencias de inteligencia estadounidenses y británicas, la Agencia de Seguridad Nacional (NSA) y el Cuartel General de Comunicaciones del Gobierno (GCHQ), respectivamente, tienen acceso a los datos de los usuarios de dispositivos Android. Estas agencias son capaces de leer casi toda la información del teléfono como SMS, geolocalización, correos, notas o mensajes. Documentos filtrados en enero de 2014, revelaron que las agencias interceptan información personal a través de Internet, redes sociales y aplicaciones populares, como Angry Birds, que recopilan información para temas comerciales y de publicidad. Además, según *The Guardian*, el GCHQ tiene una wiki con guías de las diferentes aplicaciones y redes de publicidad para saber los diferentes datos que pueden ser interceptados.

Las informaciones revelaron que las agencias realizan un esfuerzo adicional para interceptar búsquedas en Google Maps desde Android y otros smartphones para recopilar ubicaciones de forma masiva. La NSA y el GCHQ insistieron en que estas actividades cumplen con las leyes nacionales e internacionales, aunque The Guardian afirmó que *las últimas revelaciones podrían sumarse a la creciente preocupación pública acerca de cómo se acumula y utiliza la información, especialmente para aquellos fuera de los EE.UU. que gozan de menos protección en temas de privacidad que los estadounidenses*.

Desarrollo

Las aplicaciones se desarrollan habitualmente en el lenguaje Java con Android Software Development Kit (Android SDK), pero están disponibles otras herramientas de desarrollo, incluyendo un Kit de Desarrollo Nativo para aplicaciones o extensiones en C o C++, *Google App Inventor*, un entorno visual para programadores novatos y varios marcos de aplicaciones basadas en la web multiteléfono. También es posible usar las bibliotecas Qt gracias al proyecto *Necesitas SDK*.

El desarrollo de aplicaciones para Android no requiere aprender lenguajes complejos de programación. Todo lo que se necesita es un conocimiento aceptable de Java y estar en posesión del kit de desarrollo de software o «SDK» provisto por Google el cual se puede descargar gratuitamente.

Todas las aplicaciones están comprimidas en formato APK, que se pueden instalar sin dificultad desde cualquier explorador de archivos en la mayoría de dispositivos.

7.1.4. Objective-C:

Su posición en el ranking de lenguajes de programación más utilizados, no cabe duda de que ha sido impulsada gracias a Apple, y la cantidad de aplicaciones para dispositivos iOS que están apareciendo los últimos años.

Objective-C es un lenguaje de programación orientado a objetos creado como un superconjunto de C para que implementase un modelo de objetos parecido al de Smalltalk. Originalmente fue creado por Brad Cox y la corporación StepStone en 1980. En 1988 fue adoptado como lenguaje de programación de NEXTSTEP y en 1992 fue liberado bajo licencia GPL para el compilador GCC. Actualmente se usa como lenguaje principal de programación en Mac OS X, iOS y GNUstep.

Historia

A principios de los 80, el software se desarrollaba usando programación estructurada. La programación estructurada se estableció para ayudar a dividir los programas en pequeñas partes, haciendo más fácil el desarrollo cuando la aplicación se volvía muy grande. Sin embargo, como los problemas seguían creciendo al pasar el tiempo, la programación estructurada se volvió compleja dado el desorden de algunos programadores para invocar instrucciones repetitivamente, llevando a código spaghetti y dificultando la reutilización de código.

Muchos vieron que la programación orientada a objetos sería la solución al problema. De hecho, Smalltalk ya tenía solucionados muchos de estos problemas: algunos de los sistemas más complejos en el mundo funcionaban gracias a Smalltalk. Pero Smalltalk usaba una máquina virtual, lo cual requería mucha memoria para esa época, y era demasiado lento.

Objective-C fue creado principalmente por Brad Cox y Tom Love a inicios de los 80 en su compañía Stepstone. Ambos fueron iniciados en Smalltalk mientras estaban en el Programming Technology Center de ITT en 1981. Cox se vio interesado en los problemas de reutilización en el desarrollo de software. Se dio cuenta de que un lenguaje como Smalltalk sería imprescindible en la construcción de entornos de desarrollo potentes para los desarrolladores en ITI Corporation. Cox empezó a modificar el compilador de C para agregar algunas de las capacidades de Smalltalk. Pronto tuvo una extensión para añadir la programación orientada a objetos a C la cual llamó «OOPC» (*Object-Oriented Programming in C*). Love mientras tanto, fue contratado por Shlumberger Research en 1982 y tuvo la oportunidad de adquirir la primera copia de Smalltalk-80, lo que influyó en su estilo como programador.

Para demostrar que se hizo un progreso real, Cox mostró que para hacer componentes de software verdaderamente intercambiables sólo se necesitaban unos pequeños cambios en las herramientas existentes. Específicamente, estas necesitaban soportar objetos de manera flexible, venir con un conjunto de bibliotecas que fueran utilizables, y permitir que el código (y cualquier recurso

necesitado por el código) pudiera ser empaquetado en un formato multiplataforma.

Cox y Love luego fundaron una nueva empresa, Productivity Products International (PPI), para comercializar su producto, el cual era un compilador de Objective-C con un conjunto de bibliotecas potentes.

En 1986, Cox publicó la principal descripción de Objective-C en su forma original en el libro *Object-Oriented Programming, An Evolutionary Approach*. Aunque él fue cuidadoso en resaltar que hay muchos problemas de reutilización que no dependen del lenguaje, Objective-C frecuentemente fue comparado detalladamente con otros lenguajes.

Mensajes

El modelo de programación orientada a objetos de Objective-C se basa en enviar mensajes a instancias de objetos. Esto es diferente al modelo de programación al estilo de Simula, utilizado por C++ y ésta distinción es semánticamente importante. En Objective-C uno no *llama a un método*; uno *envía un mensaje*, y la diferencia entre ambos conceptos radica en cómo el código referido por el nombre del mensaje o método es ejecutado. En un lenguaje al estilo Simula, el nombre del método es en la mayoría de los casos atado a una sección de código en la clase objetivo por el compilador, pero en Smalltalk y Objective-C, el mensaje sigue siendo simplemente un nombre, y es resuelto en tiempo de ejecución: el objeto receptor tiene la tarea de interpretar por sí mismo el mensaje. Una consecuencia de esto es que el mensaje del sistema que pasa no tiene chequeo de tipo: el objeto al cual es dirigido el mensaje (conocido como *receptor*) no está inherentemente garantizado a responder a un mensaje, y si no lo hace, simplemente lo ignora y retorna un puntero nulo.

Enviar el mensaje method al objeto apuntado por el puntero obj requeriría el siguiente código en C++:

```
obj->method(parameter);
```

mientras que en Objective-C se escribiría como sigue:

```
[obj method:parameter];
```

Ambos estilos de programación poseen sus fortalezas y debilidades. La POO al estilo Simula permite herencia múltiple y rápida ejecución utilizando ligadura en tiempo de compilación siempre que sea posible, pero no soporta ligadura dinámica por defecto. Esto fuerza a que todos los métodos posean su correspondiente implementación, al menos que sean virtuales (aún así, se requiere una implementación del método para efectuar la llamada). La POO al estilo Smalltalk permite que los mensajes no posean implementación - por ejemplo, toda una colección de objetos pueden enviar un mensaje sin temor a producir errores en tiempo de ejecución. El envío de mensajes tampoco requiere que un objeto sea

definido en tiempo de compilación. (Ver más abajo la sección tipado dinámico) para más ventajas de la ligadura dinámica.

Sin embargo, se debe notar que debido a la sobrecarga de la interpretación de los mensajes, un mensaje en Objective-C toma, en el mejor de los casos, tres veces más tiempo que una llamada a un método virtual en C++.

Instanciación

Una vez que una clase es escrita en Objective-C, puede ser instanciada. Esto se lleva a cabo primeramente alojando la memoria para el nuevo objeto y luego inicializándolo. Un objeto no es completamente funcional hasta que ambos pasos sean completados. Esos pasos típicamente se logran con una simple línea de código:

```
MyObject * o = [[MyObject alloc] init];
```

La llamada a alloc aloja la memoria suficiente para mantener todas las variables de instancia para un objeto, y la llamada a init puede ser anulada para establecer las variables de instancia con valores específicos al momento de su creación. El método init es escrito a menudo de la siguiente manera:

```
-(id) init  {
        self = [super init];
        if (self)     {
                ivar1 = '"value1"';
                ivar2 = value2;
                .
                .
                .
        }
        return self;
        }
```

Protocolos

Objective-C fue extendido en NeXT para introducir el concepto de herencia múltiple de la especificación, pero no la implementación, a través de la introducción de protocolos. Este es un modelo viable, ya sea como una clase base abstracta multi-heredada en C++, o como una "interfaz" (como en Java o C#). Objective-C hace uso de protocolos ad-hoc, llamados protocolos informales, y el compilador debe cumplir los llamados protocolos formales.

[3]Página web: http://es.wikipedia.org/wiki/Objective_C

7.2. REST

La **Transferencia de Estado Representacional** (Representational State Transfer) o **REST** es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

Si bien el término *REST* se refería originalmente a un conjunto de principios de arquitectura —descritos más abajo—, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP, aunque este no fuera el planteamiento inicial y correcto de uso, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto pero sin usar SOAP. Estos dos usos diferentes del término *REST* causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST.

Los sistemas que siguen los principios REST se llaman con frecuencia *RESTful*; los defensores más acérrimos de REST se llaman a sí mismos *RESTafaris*.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un **protocolo cliente/servidor sin estado**: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)

- Un conjunto de **operaciones bien definidas** que se aplican a todos los *recursos* de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son **POST**, **GET**, **PUT** y **DELETE**. Con frecuencia estas operaciones se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema. Aún así, POST es válido siempre y cuando funcione conforme a la RFP de HTTP.

- Una **sintaxis universal** para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El **uso de hipermedios**, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son **típicamente JSON o XML**. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

7.2.1. Recursos

Un concepto importante en REST es la existencia de *recursos* (elementos de información), que pueden ser accedidos utilizando un identificador global (un Identificador Uniforme de Recurso). Para manipular estos recursos, los *componentes* de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian *representaciones* de estos recursos (los ficheros que se descargan y se envían) - es cuestión de debate, no obstante, si la distinción entre *recursos* y sus *representaciones* es demasiado platónica para su uso práctico en la red, aunque es popular en la comunidad RDF.

La petición puede ser transmitida por cualquier número de *conectores* (por ejemplo clientes, servidores, cachés, túneles, etc.) pero cada uno lo hace sin "ver más allá" de su propia petición (lo que se conoce como separación en capas, otra restricción de REST, que es un principio común con muchas otras partes de la arquitectura de redes y de la información) Así, una aplicación puede interactuar con un recurso conociendo el identificador del recurso y la acción requerida, no necesitando conocer si existen cachés, proxys, cortafuegos, túneles o cualquier otra cosa entre ella y el servidor que guarda la información. La aplicación, sin embargo, debe comprender el formato de la información devuelta (la *representación*), que es por lo general un documento JSON o XML, aunque también puede ser una imagen o cualquier otro contenido.

7.2.2. REST frente a RPC

Una aplicación web REST requiere un enfoque de diseño diferente a una aplicación basada en llamadas a procedimientos remotos. En RPC, se pone el énfasis en la diversidad de operaciones del protocolo, o *verbos*; por ejemplo una aplicación RPC podría definir operaciones como:

- `getUser()`
- `addUser()`
- `removeUser()`
- `updateUser()`
- `getLocation()`
- `addLocation()`
- `removeLocation()`
- `updateLocation()`
- `listUsers()`

- `listLocations()`
- `findLocation()`
- `findUser()`

En REST, al contrario, el énfasis se pone en la diversidad de recursos, o los *nombres*; por ejemplo, una aplicación REST podría definir los siguientes tipos de recursos:

- `Usuario {}`
- `Localización {}`

Cada recurso tendría su propio identificador, como http://www.example.org/locations/us/ny/new_york_city. Los clientes trabajarían con estos recursos a través de las operaciones estándar de HTTP, como GET para descargar una copia del recurso. Obsérvese cómo cada objeto tiene su propia URL y puede ser fácilmente cacheado, copiado y guardado como marcador. POST se utiliza por lo general para acciones con efectos laterales, como enviar una orden de compra o añadir ciertos datos a una colección.

Por ejemplo, el registro para un usuario podría tener el siguiente aspecto:

<usuario>
<nombre>María Juana</nombre>
<sexo>mujer</sexo>
<localización
href="http://www.example.org/locations/us/ny/new_york_city">Nueva York, NY, US</localización>
</usuario>

Para actualizar la localización del usuario, un cliente REST podría primero descargar el registro XML anterior usando GET. El cliente después modificaría el fichero para cambiar la localización y lo subiría al servidor utilizando HTTP PUT.

Nótese, sin embargo, que los verbos HTTP no proporcionan ningún recurso estándar para descubrir recursos -- no hay ninguna operación LIST o FIND en HTTP, que se corresponderían con las operaciones list*() y find*() en el ejemplo RPC. En su lugar, las aplicaciones basadas en datos REST resuelven el problema tratando una colección de resultados de búsqueda como otro tipo de *recurso*, lo que requiere que los diseñadores de la aplicación conozcan URLs adicionales para mostrar o buscar cada tipo de recurso.

Por ejemplo, una petición GET HTTP sobre la URL http://www.example.org/locations/us/ny/ podría devolver un enlace a una lista de ficheros en XML con todas las localizaciones posibles en Nueva York, mientras que una petición GET a la URL http://www.example.org/users?surname=Michaels podría devolver una lista de enlaces a todos los usuarios con el apellido "Michaels".

REST proporciona algunas indicaciones sobre cómo realizar este tipo de acciones como parte de su restricción "hipermedia como el medio de estado de la

aplicación", lo que sugiere el uso de un lenguaje de formularios (tales como un formulario HTML) para especificar consultas parametrizadas.

La iniciativa OpenSearch de A9.com intenta estandarizar las búsquedas usando REST estableciendo especificaciones para descubrir recursos y un formato genérico para utilizar con sistemas basados en REST, incluyendo el RDF, XTM, Atom, RSS (en sus varias formas) y XML con XLink para gestionar los enlaces.

7.2.3. Implementaciones públicas

Dado que la definición de REST es muy amplia, es posible afirmar que existe un enorme número de aplicaciones REST en la red (prácticamente cualquier cosa accesible mediante una petición HTTP GET). De forma más restrictiva, en contraposición a los servicios web y el RPC, REST se puede encontrar en diferentes áreas de la web:

- La blogosfera -el universo de los blogs- está, en su mayor parte, basado en REST, dado que implica descargar ficheros XML (en formato RSS o Atom) que contienen listas de enlaces a otros recursos.
- Amazon.com ofrece su interfaz para desarrolladores tanto en formato REST como en formato SOAP (siendo la versión REST la que recibe mayor tráfico).
- eBay ofrece una interfaz REST para desarrolladores.
- El Proyecto "Seniors Canada On-line" del Gobierno de Canadá ofrece una interfaz REST descrito aquí.
- Bloglines ofrece un API basado REST para desarrolladores.
- Yahoo! ofrece un API en REST para desarrolladores.
- El mecanismo de enrutamiento de Ruby on Rails soporta aplicaciones REST utilizando el patrón de diseño MVC.
- Microsoft tiene su implementación en ADO.NET Data Services Framework (anteriormente conocido como “Astoria”) [1].
- El mismo mecanismo en Catalyst también soporta aplicaciones REST mediante MVC.
- El publicador de objetos de Zope.
- Implementación REST para Java: RestLet.

Probablemente existan muchas otras implementaciones similares.

En cualquier caso debe tenerse en cuenta que muchas de las implementaciones descritas arriba, no son totalmente RESTful, esto es, no respetan todas las restricciones que impone la arquitectura REST. Sin embargo todas están inspiradas en REST y respetan los aspectos más significativos y restrictivos de su arquitectura, en particular la restricción de "interfaz uniforme". Estos servicios han sido denominados "Accidentalmente RESTful".

[4]Página web: http://es.wikipedia.org/wiki/REST

7.2.4. SOAP vs REST

Empecemos por el principio: **¿Que es un servicio SOAP y un servicio REST?**

SOAP

- SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.
- Las operaciones son definidas como puertos WSDL.
- Dirección única para todas las operaciones.
- Componentes fuertemente acoplados.
- Múltiples instancias del proceso comparten la misma operación.

REST

- Es un estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la Web. Se centra en los estándares HTTP y XML.
- Paradigma creado en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de http.
- REST se refiere únicamente a una colección de principios para el diseño de arquitecturas en red que se centran en como los recursos son definidos y diseccionados.
- El objetivo principal es transmitir un conjunto de datos de un domino sobre el protocolo HTTP sin la necesidad de contar con una capa adicional, como hace SOAP.
- Las operaciones se definen como mensajes.
- Una dirección única para cada instancia del proceso.
- Componentes débilmente acoplados.

¿Donde es mejor utilizar SOAP?

- Cuando se establece un contrato formal donde se describe todas las funciones de la interfaz. El lenguje WSDL(*Web Services Description Language*) nos permite definir cualquier detalle.

- Cuando es necesario que la arquitectura aborde requerimientos complejos no funcionales. Por ejemplo en el uso de transacciones, seguridad o direccionamiento, casos donde es necesario mantener la información contextual y el estado.

- En casos donde la arquitectura necesita manejar un procesado asíncrono debido al tiempo que necesita para realizar una parte del procesado de la petición.

¿Donde es mejor utilizar REST?

- Cuando el servicio web no necesita tener estado.
- Cuando buscamos mejorar el rendimiento, una infraestructura caching puede mejorarlo.
- En momentos donde el productor como el consumidor conocen el contexto y contenido que van a intercambiar.
- REST es muy util para el consumo como servicio web en dispositivos móviles donde tenemos escasos recursos.
- REST es un acierto en la creación de servicios que se utilizaran de agregadores de información en mashup o sitios web existentes. También cuando usemos tecnologías como AJAX o framewoks javascript.

7.2.5. REST y JSON – Combinación Ganadora

Los llamados web services (servicios web) pueden ser implementados usando diferentes tecnologías, protocolos y formatos de datos, en mi caso, de cara a éste proyecto, decidí apostar por tecnologías que, o han aparecido junto con la entrada en escena del sector de las aplicaciones móviles, o con la aparición de éstas han ganado un interés que antes no tenían.

Es el caso de REST y JSON, las cuales han sido descritas anteriormente.

Con el paso del tiempo, se está viendo que ésta combinación es bastante equilibrada, y tiene las características necesarias para convertirse en el nuevo estándar.

El motivo principal por el cual la comunidad ha decidido adoptarlas de una manera bastante amplia, es que el concepto es mucho más sencillo de comprender que sus competidores: SOAP y XML.

Otro motivo, es que REST no necesita una capa intermedia como SOAP, sino que transmite los datos directamente a través de http. Si a eso le juntamos que JSON permite enviar datos, ocupando menos espacio que XML y no por ello renunciando a ningún aspecto de legibilidad o flexibilidad, pues tenemos motivos suficientes como para inclinarnos a pensar que no es una mala combinación.

Para poner un ejemplo de cómo ha sido la adopción por la comunidad de éstas tecnologías, me remito a la web "http://www.programmableweb.com/", una web referencia en el tema de las API's públicas.

Si en esta web nos vamos a la sección de estadísticas, comprobamos la relación de protocolos y formato de datos más utilizados:

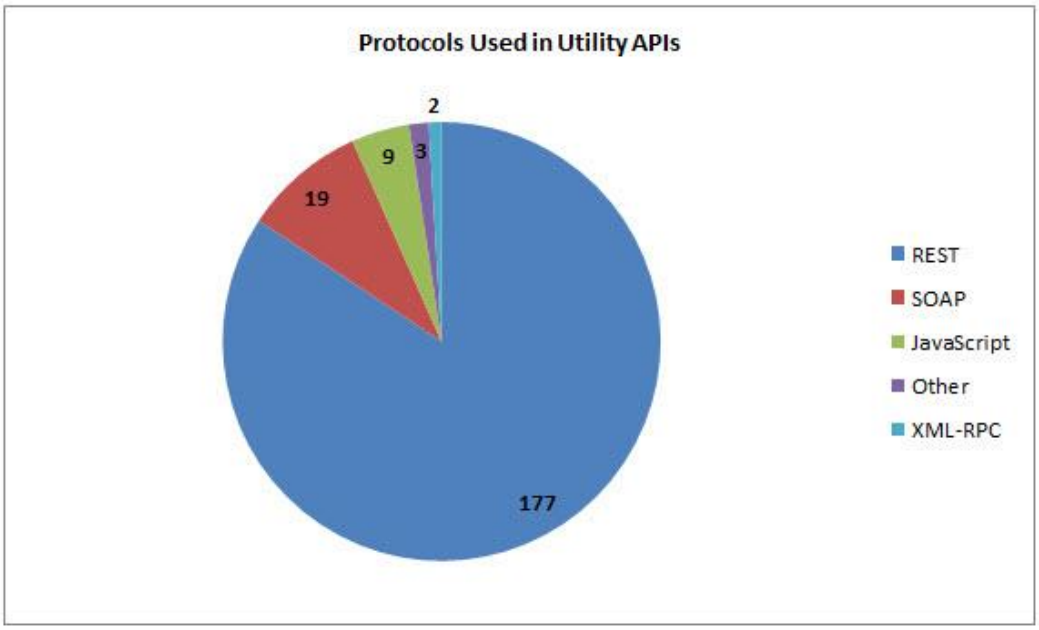


Figura 55. Utilización de protocolos en el repertorio de API's públicas de la web

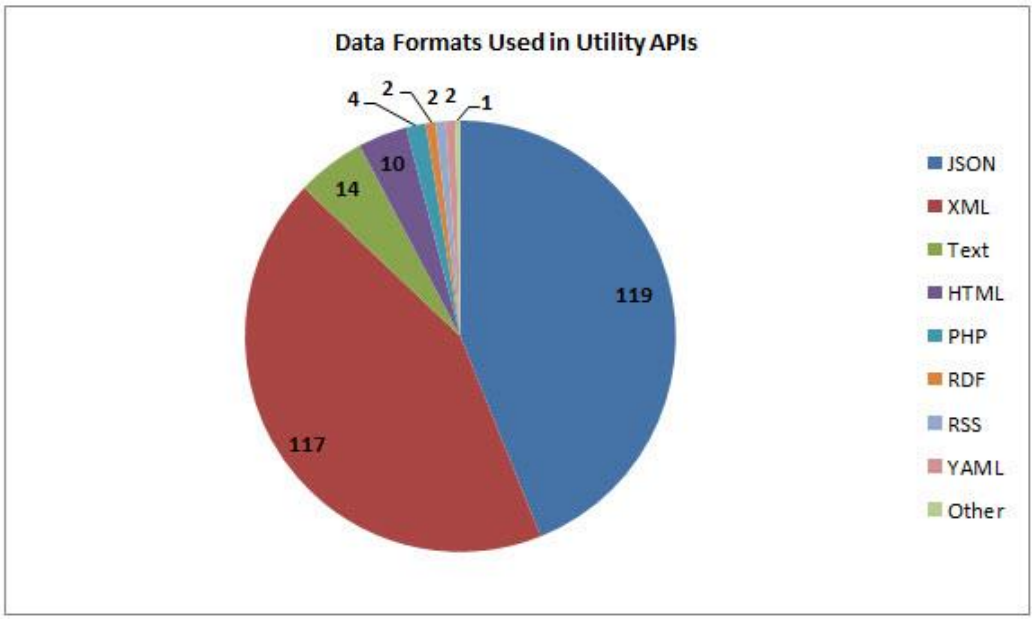


Figura 56. Utilización de formato de datos en el repertorio de API's públicas de la web

Si nos referimos a las mismas variables en periodos anteriores, observamos el claro crecimiento de las tecnologías escogidas para este proyecto:

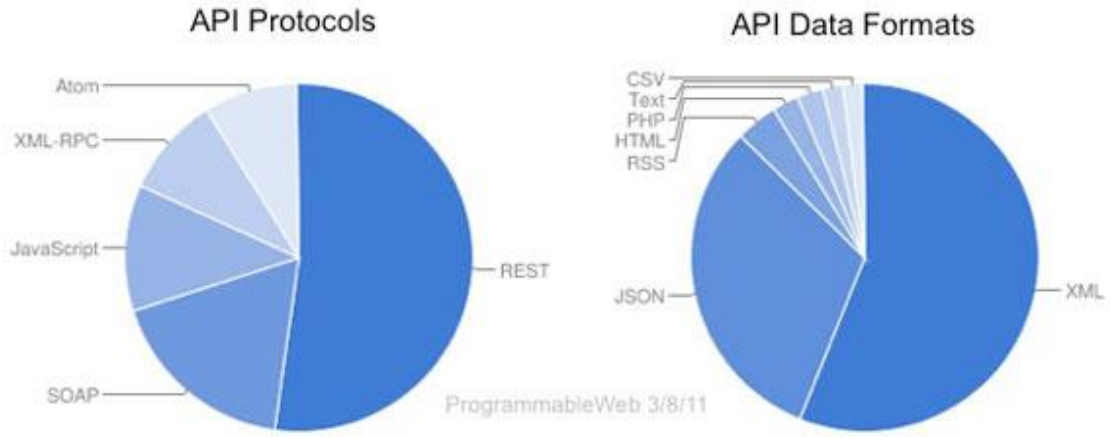


Figura 57. Comparativa con la situación de 2011

Como se puede apreciar, los desarrolladores están apostando por este conjunto, y yo, al igual que ellos, será la combinación que tendré en cuenta de cara a futuros proyectos a no ser que lleguen otros agentes a modificar el entorno, entonces, habrá que estudiar las posibilidades.