Escuela técnica superior de ingeniería de telecomunicación

PROYECTO FIN DE CARRERA

Autor:
Miguel Sánchez del Águila
Director:
Fernando Losilla López

Septiembre 2012

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 iOS 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 iPhone encargada de clasificar toda esa información de manera útil para el usuario, representando en un mapa la ubicación de los vehículos seleccionados.

Titulación:
Ingeniería Técnica de Telecomunicación. Especialidad en Telemática

Departamento:
Depto. TIC

Fecha de presentación:
Septiembre 2012

Índice:

1. Descripción general del proyecto

1.1 Motivación y Objetivo

Como estudiante universitario de ingeniería de telecomunicación, siento especial interés por la tecnología, tanto actual, cómo desfasada. A esto se añade, que hoy en día casi todo está informatizado, y centralizado en la nube (Cloud Computing), dado que es muy eficiente en algunos casos y ámbitos.

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, teníamos muchos dispositivos, los cuáles cada uno de ellos realizaban su función, hoy en día, es muy común que el teléfono inteligente que llevamos en el bolsillo sea más potente que todos esos antiguos dispositivos juntos. Por esta razón, se dice que estamos en el momento de las "apps", es decir, pequeños programas para los teléfonos inteligentes (smartphones) que cumplen las necesidades del usuario.

En el mundo policial, se necesita una tecnología más especializada. Y no me refiero en muchos casos a nuevo hardware que realice nuevas funciones. Si no aprovechar la tecnología actual, y enfocarla en torno a las necesidades y nuevos horizontes.

Éste proyecto es una muestra de lo mencionado hasta ahora, juntar la potencia de los nuevos smartphones, junto con el hardware actual, y darle un enfoque para que cumpla con una función, que hasta ahora no se ha implementado.

El objetivo de éste proyecto es el control mediante un terminal iPhone, 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 capturas del vehículo.

Para ello, aparte de la aplicación, debe haber:

- 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.

Por supuesto, se podría realizar la implementación con otros componentes, y otros procesos, pero esto sería una solución particular diseñada por mí, para la resolución del problema antes mencionado "Seguimiento de vehículos".

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, surge como la expansión de otro proyecto que ha sido desarrollado paralelamente a éste 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 vehículos, 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

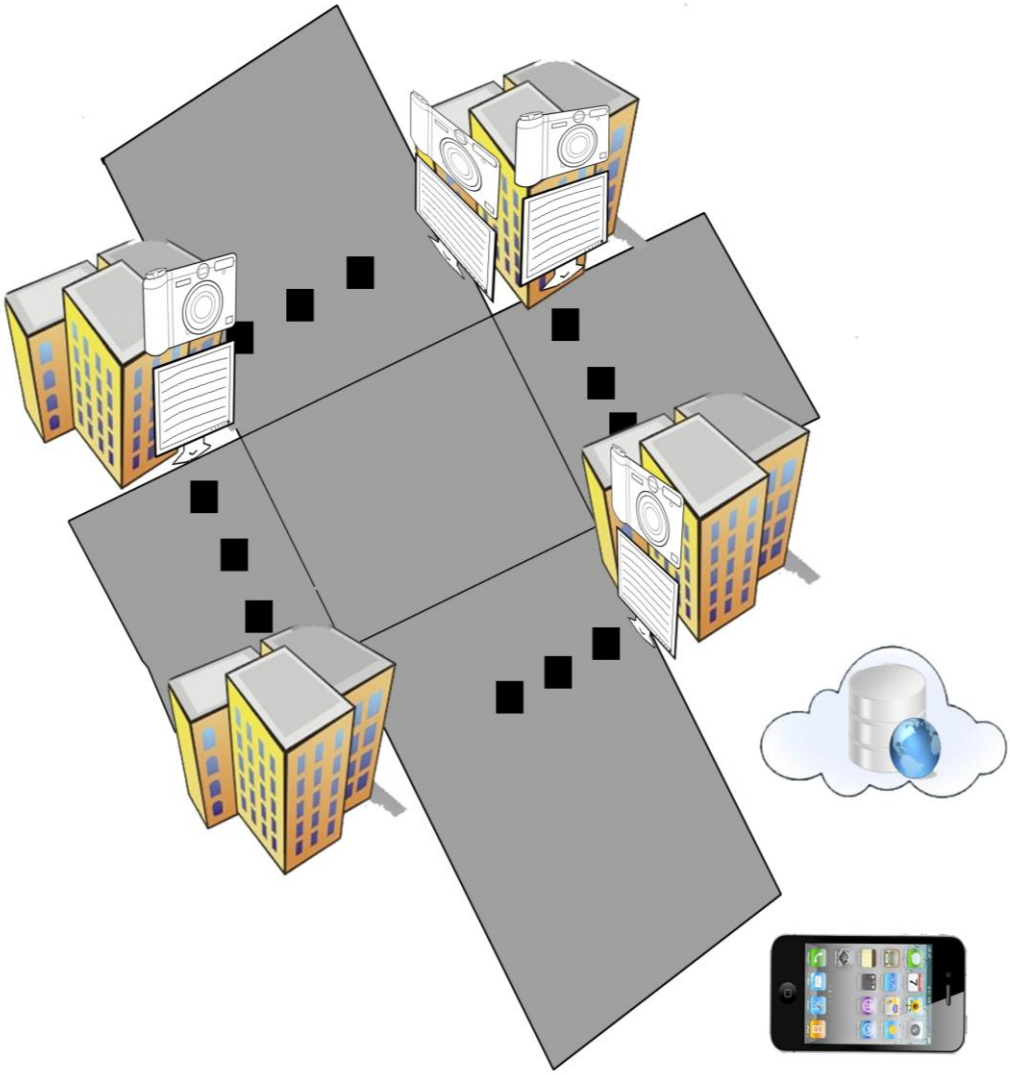


Figura 1. Boceto del entorno general del sistema

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 iPhone

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, ejecuta un script, el cuál le permite enviar y recibir determinados mensajes tanto de los nodos sensores como del servidor. Posee también una cámara, que utilizará para obtener más información de los coches que pasen por los nodos sensores.

El servidor web posee unos scripts PHP que le permiten actuar de pasarela entre la base de datos MySQL, la aplicación y los equipos encargados de gestionar los nodos sensores.

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

La aplicación para iPhone, es 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 ejecutar el software diseñado para el equipo gestor de los nodos sensores por cada una de las calles que componen el sistema.

La siguiente interacción depende de cuando el usuario (policía patrulla) inicie la aplicación para iPhone. En ese momento obtiene la información de las calles registradas.

El usuario selecciona una de las calles registradas.

Aparece una lista con los coches que circulan por la calle seleccionada.

En ese 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, que muestra las diferentes calles por las que el coche ha pasado desde ese momento.

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, creo que sería una aplicación para 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 Otras plataformas con las que puede interactuar el sistema

Otras plataformas podrían ser, los centros de estadística, o la DGT, recopilando la cantidad de vehículos que circulan por cada calle por día u hora.

También podría interactuar con otra posible aplicación móvil para plataformas Android, la cuál no queda recogida en este proyecto.

2.2 Descripción de los lenguajes utilizados:

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

- Para la aplicación que es ejecutada en los equipos que coordinan los nodos sensores con el servidor, 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 scripts 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.

- 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.

  SQL aparece 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.

- 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:

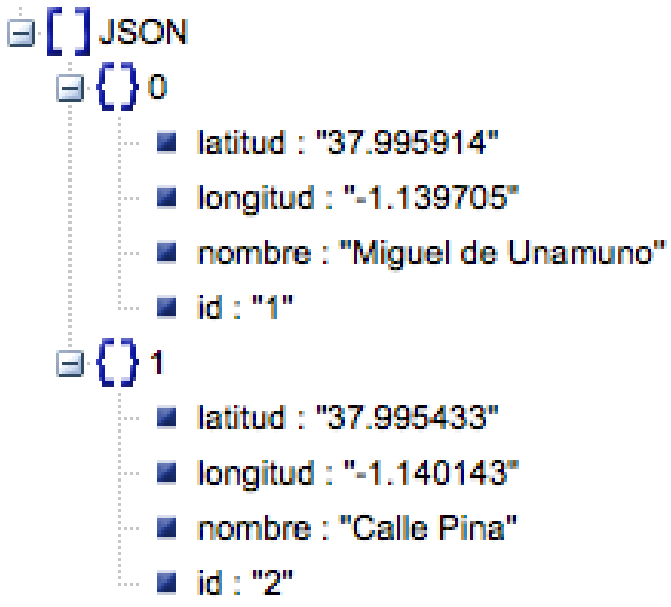


Figura 2. Ejemplo de estructura de objeto JSON

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

- Para la aplicación móvil 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 (ObjC), 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 ObjC el programador no llama a métodos, sino que envía mensajes.
  Al igual que en C++ en ObjC 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 ObjC 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.

La sección "Anexos" contiene más información sobre cada uno de los lenguajes utilizados para la implementación del sistema, aparte de un ejemplo tutorial para el desarrollo de una aplicación para dispositivos iOS.

3. Descripción de la implementación

3.1 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 3. 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 4. 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 y se envían a un nuevo hilo.

En este punto puede empezar a recibir mensajes correspondientes a las firmas de los vehículos, las cuáles son mediciones del campo magnético creado por el objeto 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.

Puede recibir cinco tipos de mensaje:

- *Sensor:X(entero)*
- *FirmaTx:Y*
- *FirmaSelec:X|Y*
- *FirmaRx:X*
- *CorrelacionOK:X*

- El primer tipo de mensaje sirve para avisar a los equipos gestores del sensor escogido por el usuario (policía que selecciona la calle desde donde parte el infractor), 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 envía el servidor a los equipos gestores cuando un usuario 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 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.

  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:

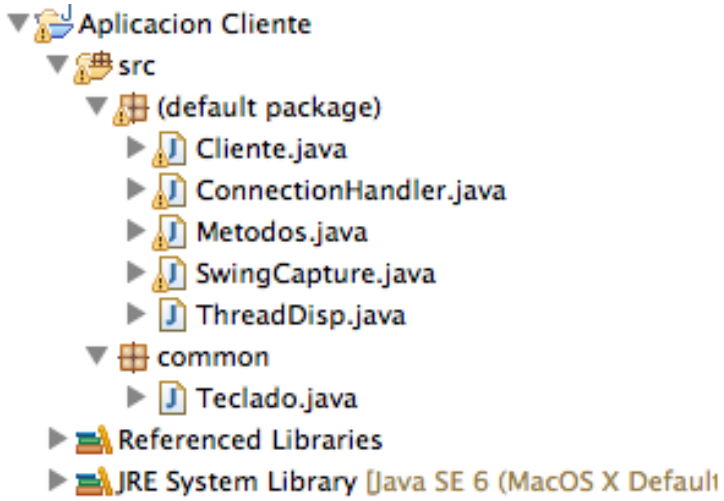


Figura 5. Estructura de los archivos del programa para los equipos gestores

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, pero es la opción más favorable.

Un posible equipo para esta función es un hardware del tipo "Raspberry Pi" . Este dispositivo es un SBC que se ha puesto a la venta este año.

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 6. 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.

El precio también puede disminuir con una posible fabricación en masa de estos dispositivos, aunque ya es bastante ajustado para los componentes que contiene.

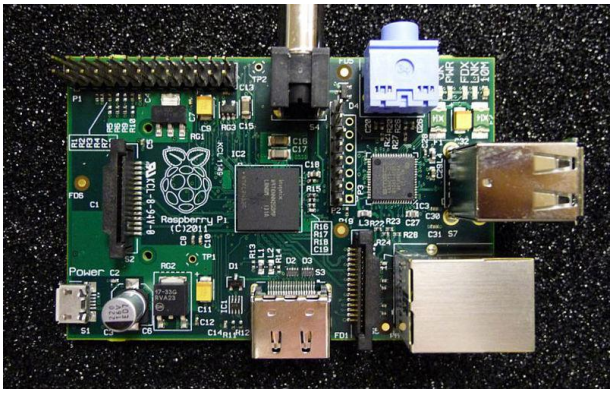


Figura 7. Imagen del dispositivo Raspberry Pi

3.2 Servidor web y base de datos

3.2.1 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.

Las funciones principales de estos scripts son:

- Consultar la información de nodos sensores y coches alojada en la base de datos.
- Enviar los mensajes dependientes del usuario a los equipos gestores de los conjuntos de nodos sensores.

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", 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.

- El tercero, "*txFirmas.php"* se ejecuta cuando el usuario ha seleccionado un sensor, 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 8. 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 9. 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".

  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

- "*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 11. 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 12. 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.

3.2.2 Base de datos

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

- "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 13. Estructura de la tabla coordenadas

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 14. 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.

- "firmasX":

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

Figura 15. Estructura de la tabla firmas

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

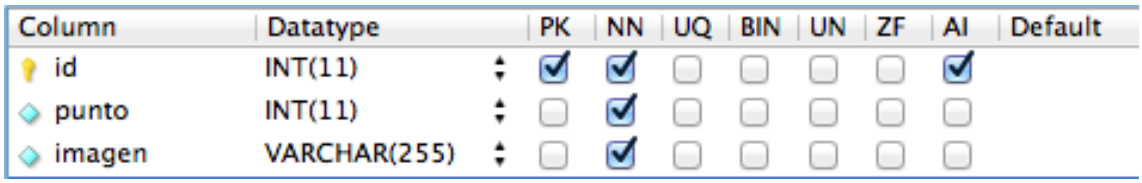
Figura 16. Ejemplo de la tabla firmas rellena con datos

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

El campo imagen es el path donde está almacenada la imagen captada por la cámara del equipo gestor, y el campo número, contiene un valor único por cada coche que representa a ese coche concreto en el sistema.

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.

- “recorridoX”:



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

Figura 17. Estructura de la tabla recorrido

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 18. 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.

3.3 Aplicación iOS

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

La estructura del programa es la siguiente:

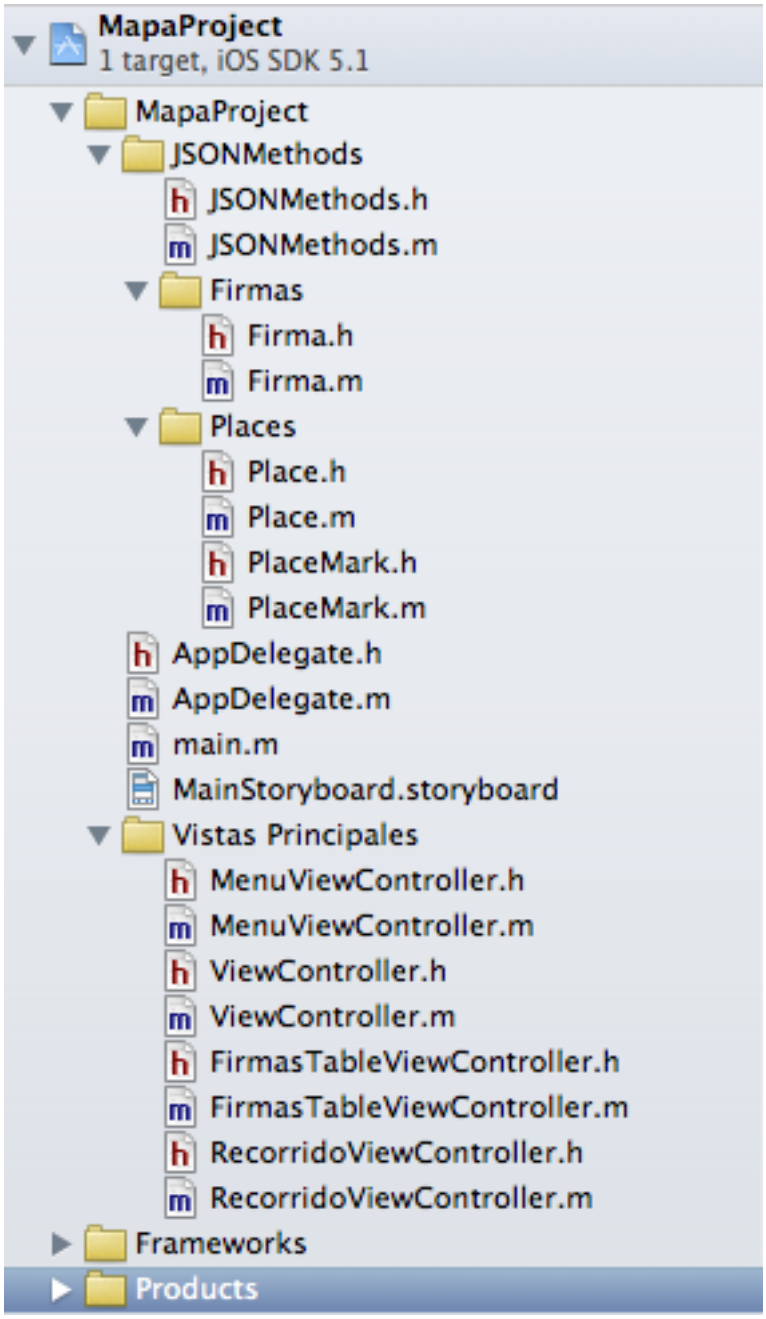


Figura 19. Estructura de la aplicación para iPhone

En ObjC es obligatorio definir cada clase con el archivo interfaz ".h", el cuál contiene las propiedades de la clase (variables de instancia), y los métodos o acciones que puede ejecutar.

El contenido de la interfaz "JSONMethods.h" se describe a continuación:

```
//
//  JSONMethods.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 04/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "Place.h"
#import "PlaceMark.h"
#import "Firma.h"
#import "FirmasTableViewController.h"

@interface JSONMethods : NSObject

@property (nonatomic,strong) NSMutableArray *resp;

+(NSData *) getResponseFrom:(NSString *)url;
+(NSMutableArray *) getDataJSON:(NSData *)response;
+(NSMutableArray *) getRecorridoJSON:(NSData *)response;
+(NSMutableArray *)obtenerSensores;
+(NSMutableArray *)refreshRecorrido:(NSInteger *) firma;
-(NSString *)sendPostMsg:(NSString *) url:(NSString *)argument;
-(void)sendPostMsgWithArrayArguments:(NSArray *)myArgs;
+(NSMutableArray *)convertirArrayFirmas:(NSString *)firmas;

@end
```

Figura 20. Contenido de la interfaz JSONMethods

En primer lugar se aprecia la declaración "*#import"* al igual que "*#define*" en C nos permite añadir objetos o métodos de otras clases a la clase objetivo.

Casi todos los objetos que se usan cuando se realiza una aplicación para iPhone comienzan por "NS", esos objetos pertenecen a la API de Cocoa Touch, mencionada en el punto 2.2 de la memoria.

Como se puede apreciar en la imagen de arriba, la sintaxis de los métodos en ObjC es heredada de *SmallTalk* siendo el propio nombre del método bastante descriptivo sobre su función.

A la hora de hacer la declaración de un método, el signo "+" o "-" que se coloca al comienzo, indica si es un método estático o de clase.

- "+" declara al método como estático.
- "-" indica que debe ejecutarse sobre una instancia del objeto.

A continuación se describe la función de los métodos de la interfaz:

- "*getResponseFrom:*" Hace una HTTP request simple sin argumentos, y recibe el response como el valor de retorno.
- "*getDataJSON:*" Convierte el string JSON de los equipos gestores registrados en objetos que se pueden utilizar en la vista determinada.
- "getRecorridoJSON" Realiza una función similar a la anterior, con la diferencia de que convierte los objetos, a objetos útiles a la hora de realizar el recorrido del coche infractor.

- "*obtenerNodos sensores*" Devuelve a la vista donde aparecen los nodos sensores registrados, la información necesaria para que rellene su tabla, y realice las anotaciones sobre el mapa.
- "*refreshRecorrido:"* Devuelve los puntos por donde pasa una firma concreta caracterizada por su ID, el cuál se le pasa al método como argumento.
- "*sendPostMsg::"* Recibe como primer argumento una URL, y como segundo argumento un parámetro POST del tipo: "parámetro=X". Como valor de retorno devuelve el código HTML de la página resultante.
- "*sendPostMsgWithArrayArguments:"* Actúa como una capa superior para el método anterior, y recibe como argumentos un array con la URL y los parámetros POST que queramos enviar. Esto es necesario a la hora de ejecutar un método después de un delay, pues la única manera de hacerlo es creando un objeto de tipo "*@selector*" el cual puede recibir únicamente un argumento.
- "*ConvertirArrayFirmas:"* Tiene una función similar a "*getDataJSON*" ó "*getRecorridoJSON*", sólo que crea elementos del tipo "Firma" necesarios para la tabla de las firmas que pasan por un sensor determinado.

La estructura de la clase Firma:

```
//
//  Firma.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 17/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Firma : NSObject{
    int ID;
    UIImage *imagen;
    int numero;
}

@property (nonatomic) int ID;
@property (nonatomic,strong) NSString *urlimagen;
@property (nonatomic, retain) UIImage* imagen;
@property (nonatomic) int numero;

@end
```

Figura 21. Estructura de la interfaz Firma

Como se aprecia, los elementos principales que la componen son:

- Un ID que la representa en la tabla
- La URL de la imagen captada por la cámara
- El elemento imagen que contendrá la que hay almacenada en la URL
- Un número único que caracteriza la firma.

El elemento "*Place.h*" contiene la información necesaria para representar la información de los nodos sensores en la "UITableView" (tabla de datos)

```
//
//  Place.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 01/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>


@interface Place : NSObject {

    NSString* name;
    NSString* imagen;
    NSString* description;
    double latitude;
    double longitude;
}

@property (nonatomic, retain) NSString* name;
@property (nonatomic, retain) NSString* imagen;
@property (nonatomic, retain) NSString* description;
@property (nonatomic) double latitude;
@property (nonatomic) double longitude;
@property (nonatomic) int ID;

@end
```

Figura 22. Estructura de la interfaz Place

Los elementos que lo forman son:

- "name" El nombre de la calle donde está situado.
- "imagen" La URL de una imagen accesoria para la tabla.
- "description" Una posible descripción del punto.
- "latitude" La latitud de la coordenada donde está situado.
- "longitude" La longitud de la coordenada.
- "ID" El ID que ocupa en la tabla del sensor determinado.

Para poder utilizar un elemento como anotación dentro de un elemento del tipo "*MKMapView*" (elemento mapa), debe heredar de la clase "*MKAnnotation*". Para ello se ha creado el objeto "*PlaceMark*" con la siguiente estructura:

```
//
//  PlaceMark.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 01/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>
#import "Place.h"

@interface PlaceMark : NSObject <MKAnnotation> {

    CLLocationCoordinate2D coordinate;
    Place* place;
}

@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
@property (nonatomic, retain) Place* place;

-(id) initWithPlace: (Place*) p;

@end
```

Figura 23. Estructura de la interfaz PlaceMark

Siguiendo un patrón decorador, el elemento "*PlaceMark*" posee un objeto "CLLocationCoordinate2D" (coordenada) necesario, y otro objeto del tipo "Place" que utilizamos para rellenar la coordenada.

El "*AppDelegate*" es la superclase que regenta al programa.

Cuando se inicia el "*main*", éste llama al "*AppDelegate*", que será el encargado de mostrar la vista principal del programa, y ejecutar las funciones que le indiquemos antes de comenzar.

```
//
//  AppDelegate.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 01/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

Figura 24. Contenido del archivo AppDelegatFigura e.h

Los delegates (delegados) son superclases que regentan los objetos que están por debajo.

Por ejemplo, cuando se maneja un objeto "UITableView" se tiene que añadir a su clase dos delegados:

- "<UITableViewDelegate>"
- "<UITableViewDataSource>"

Cuando se añade a su clase, dentro de ella se pueden utilizar nuevos métodos referentes a los objetos tipo "UITableView" y "UITableViewCell" (celdas), los cuáles permiten darle a la tabla y a sus celdas, la estructura que se necesita.

Por norma general, el archivo "*main.m*" en una aplicación para iPhone no se modifica, se trabaja con las clases que quedan por debajo, normalmente, el punto de partida es la clase "*AppDelegate*".

La estructura por defecto del archivo "*main.m*" es la siguiente:

```
//
//  main.m
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 01/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

#import "AppDelegate.h"

int main(int argc, char *argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}
```

Figura 25. Contenido del archivo maiFigura n.m

Como se puede comprobar, lo único que hace es una llamada al "*AppDelegate*" para comenzar el proceso.

El siguiente elemento del programa es el StoryBoard.

Esta herramienta se encarga de definir la parte gráfica de cada una de las ventanas del programa, y parte de las transiciones entre ellas, principalmente el orden, y los desencadenantes encargados de pasar de una vista a otra.

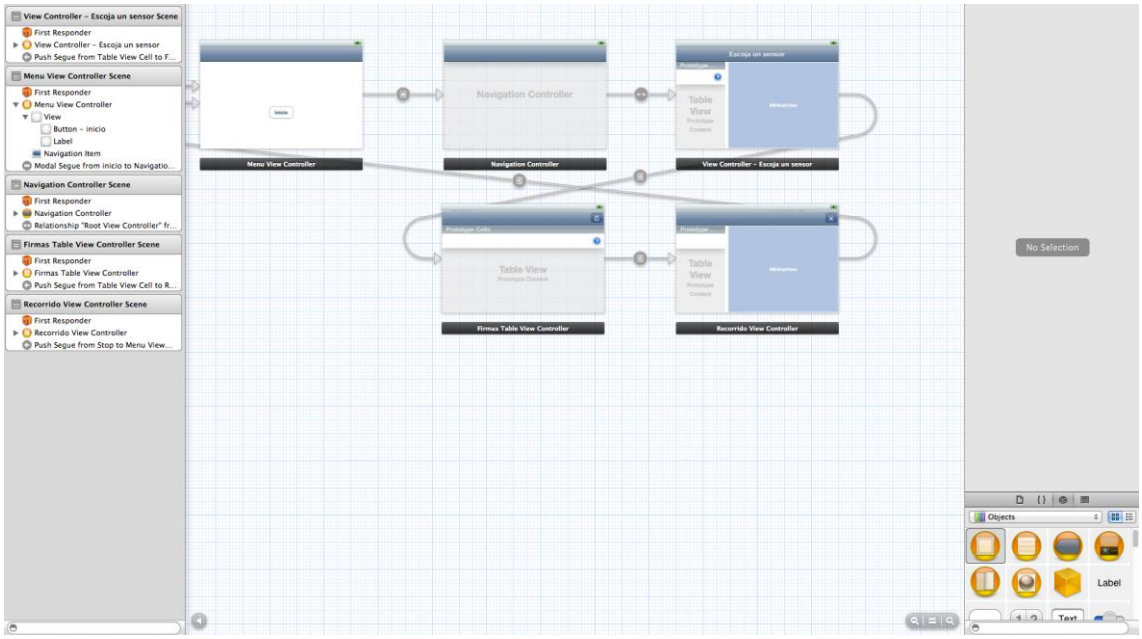


Figura 26. Vista del StoryBoard de la aplicación

A la izquierda, en modo lista aparecen las diferentes vistas que tenemos en el StoryBoard, y los elementos que contienen.

En la ventana de en medio, se puede crear el estilo de cada una de las vistas, y las transiciones entre ellas.

La parte de la derecha, es la barra de herramientas de los objetos, donde se pueden definir muchas de sus propiedades.

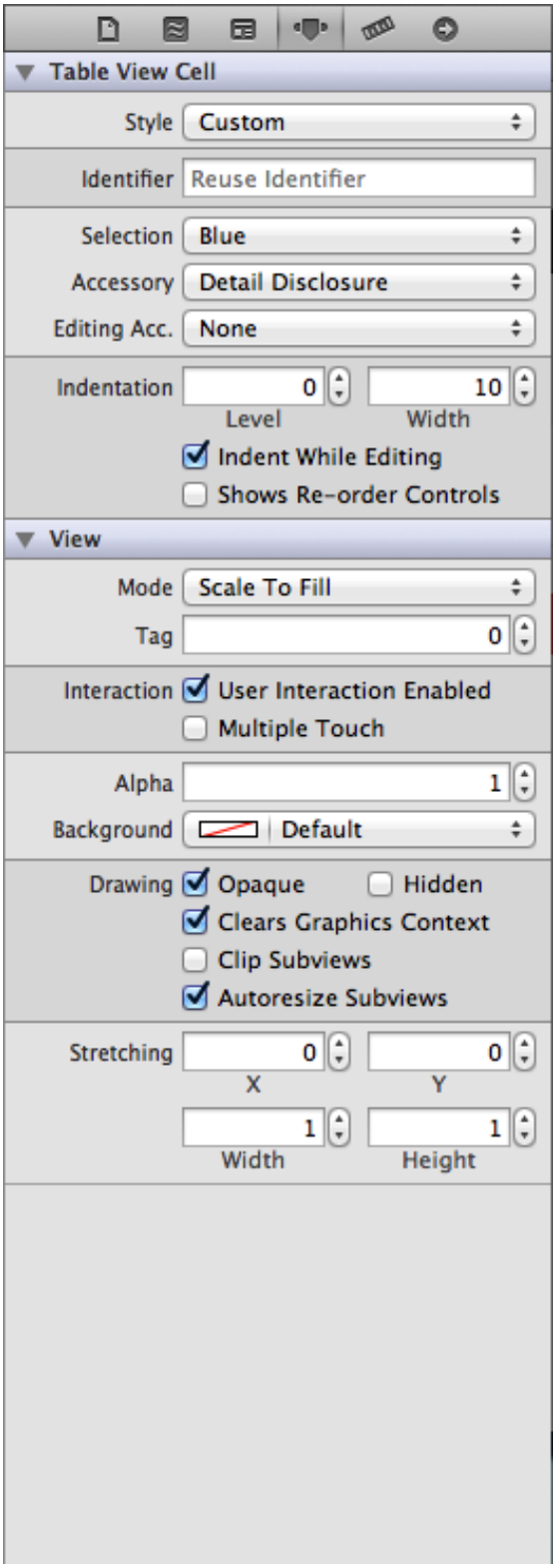


Figura 27. Vista de la barra de herramientas de un objeto "UITableViewCell"

La parte inferior derecha contiene los objetos que se pueden añadir a cualquiera de nuestras vistas:

Figura 28. Vista de la barra de herramientas de UIKit

El punto 7 de la memoria contiene una explicación mas detenida sobre el uso del StoryBoard

Se procede a describir el contenido de las vistas principales de la aplicación.

En primer lugar se aprecia una vista simple que da comienzo a la aplicación llamada "*MenuViewController*". El contenido de su interfaz es el siguiente:

```
//
//  MenuViewController.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 25/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface MenuViewController : UIViewController

@property (nonatomic,strong) IBOutlet UIButton *boton;
@property (nonatomic,strong) IBOutlet UINavigationBar *navbar;
@property (nonatomic,strong) IBOutlet UIImage *fondo;

@end
```

Figura 29. Contenido de la interfaz de la vista "MenuViewController"

Posee un botón que da comienzo a la aplicación, el objeto "*UINavigationBar*" para darle la propiedad de "*hidden"* (oculto), pues no es necesario que aparezca en esta vista.

También contiene una imagen de fondo en el elemento "UIImage".

La siguiente vista a la que se da paso es del tipo "UI*ViewController*". Ya que es la primera vista que se diseñó, se titula "ViewController".

Contiene los siguientes elementos:

```
//
//  ViewController.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 01/07/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import "JSONMethods.h"

@interface ViewController : UIViewController <UITableViewDelegate, UITableViewDataSource,MKMapViewDelegate>

@property (nonatomic,strong) IBOutlet MKMapView *oMapa;
@property (nonatomic,strong) IBOutlet UITableView *tabla;
@property (nonatomic) BOOL firmaSeleccionada,iniciado;
@property (nonatomic,strong) NSMutableArray *celdas;
@property (nonatomic) int titleNumber;;
@property (nonatomic,strong) PlaceMark *calleMark;

-(void)actualizarSensores;

@end
```

Figura 30. Contenido de la interfaz de la vista "ViewController"

Y el prototipo es:

Figura 31. Prototipo de la vista ViewController

A simple vista en el prototipo se puede diferenciar el "*UITableView*", el "*MKMapView*", y también la "UINavigationBar" heredada del "UINavigationController" que tiene incrustada la vista.

El método "*actualizarNodos sensores*" es un método recurrente que se llama a sí mismo cada medio segundo, se encarga de obtener los equipos gestores registrados en el sistema, y actualizar la tabla en caso de que se registre alguno nuevo.

```
-(void)actualizarSensores
{
    celdas = [JSONMethods obtenerSensores];
    if(iniciado){
        Boolean flag = false;
        for(int i=0;i<celdas.count;i++){
            flag = false;
            PlaceMark *aux = [celdas objectAtIndex:i];
            for (int j = 0;j<oMapa.annotations.count;j++){
                PlaceMark *aux2 = [oMapa.annotations objectAtIndex:j];
                if ([aux.place.name isEqualToString:aux2.place.name]) flag = true;
            }
            if(!flag){
                [oMapa addAnnotation:aux];
            }
        }
    }else{
        for(int i=0;i<celdas.count;i++){
            PlaceMark *aux = [celdas objectAtIndex:i];

            [oMapa addAnnotation:aux];
        }
        iniciado = true;

    }
    [self.tabla reloadData];

    [self performSelector:@selector(actualizarSensores)withObject:nil afterDelay:0.5];

}
```

Figura 32. Función actualizarNodos sensores

La clase "*ViewController*" al heredar de "*UIViewController*" obtiene también métodos para manejar la vista como los siguientes:

```
//#############################################################################
//MÉTODOS PARA LA VISTA
//#############################################################################
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {...}

- (void)viewDidLoad
{...}

-(void)actualizarSensores
{...}

- (void)viewDidUnload
{...}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{...}
```

Figura 33. Contenido de la implementación de "ViewController"

"*ViewDidLoad*" es el método que prepara la vista antes de que cargue, por tanto si queremos asignarle algo antes de que aparezca, debemos insertarlo ahí.

El contenido del método para esta vista viene a continuación:

```
- (void)viewDidLoad
{
    iniciado = false;
    CLLocationCoordinate2D Coordenada;
    MKCoordinateSpan Span;
    MKCoordinateRegion Region;
    Coordenada.latitude=37.983445;
    Coordenada.longitude=-1.12989;
    Span.latitudeDelta=0.02;
    Span.longitudeDelta=0.02;
    Region.span=Span;
    Region.center=Coordenada;
    oMapa.region=Region;

    [super viewDidLoad];
    self.navigationItem.hidesBackButton = YES;
    [self actualizarSensores];
}
```

Figura 34. Función inicial de "ViewController"

La primera parte del código se encarga de centrar el mapa en la región correspondiente a las coordenadas introducidas.

Como se ha explicado anteriormente, al poseer un elemento "UITableView" se han añadido los delegados correspondientes a la clase, lo cual permite acceder a los siguientes métodos, entre otros:

```
//#############################################################################
//MÉTODOS PARA EL TABLEVIEW
//#############################################################################

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{...}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{...}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath{...}

-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath{...}

-(void)tableView:(UITableView *)tableView accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath{...}
```

Figura 35. Implementación de UITableView de "ViewController"

La siguiente vista principal es: "*FirmasTableViewController*" que hereda de "*UITableViewController*". Su prototipo es:

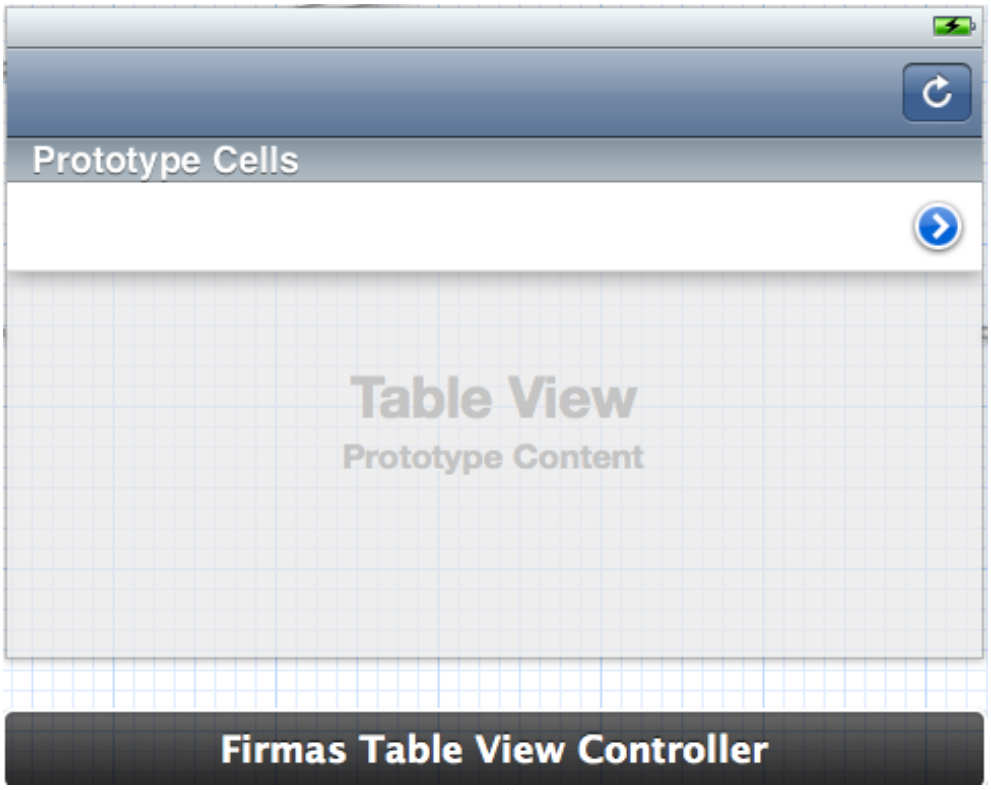


Figura 36. Prototipo de la vista "FirmasTableViewController"

El contenido de su interfaz:

```
//
//  FirmasTableViewController.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 21/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Firma.h"
#import "JSONMethods.h"
#import "RecorridoViewController.h"

@interface FirmasTableViewController : UITableViewController

@property (nonatomic) NSInteger tableViewNumber;
@property (nonatomic,strong) NSMutableArray *respuesta;
@property (nonatomic,strong) Firma *oFirma;
@property (nonatomic) int firmaSelec;

- (IBAction)actualizarFirmas:(id)sender;

- (void) actualizarTabla;

@end
```

Figura 37. Contenido de la interfaz de "FirmasTableViewController"

Como se puede observar, importa el objeto "*Firma.h*" pues principalmente son los objetos con los que trabaja la tabla.

Los métodos de su implementación son:

```
//
//  FirmasTableViewController.m
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 21/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "FirmasTableViewController.h"

@interface FirmasTableViewController ()

@end

@implementation FirmasTableViewController

@synthesize tableViewNumber,respuesta,oFirma,firmaSelec,tableView;

- (id)initWithStyle:(UITableViewStyle)style
{...}

- (void)viewDidLoad
{...}

-(void)viewWillAppear:(BOOL)animated{...}

- (void)viewDidUnload
{...}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{...}


#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{...}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{...}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{...}

- (void)actualizarTabla{...}

-(IBAction)actualizarFirmas:(id)sender
{...}

#pragma mark - Table view delegate

-(void)tableView:(UITableView *)tableView accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
{...}

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {...}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{...}

@end
```

Figura 38. Contenido de la implementación de "FirmasTableViewController"

El método "*accessoryButtonTappedForRowWithIndexPath*" será el que se ejecute cuando se pulse el botón azul circular que posee cada una de las celdas de la tabla.

Aquí aparece también un elemento nuevo, "*IBAction*". Este tipo de métodos van enlazados a los elementos del tipo "*IBOutlet*".

Los "*IBOutlets*" son los elementos visuales que podemos añadir a los prototipos de las vistas, y cuando ocurre una acción sobre esos elementos, podemos invocar una acción "*IBAction*".

En este caso, la "*IBAction*" se invoca cuando se presiona el botón actualizar contenido en la "*UINavigationBar*" de la vista.

En el anexo tutorial de desarrollo de una aplicación para dispositivos iOS se puede entender mejor éste concepto.

La siguiente y última vista del programa es similar a la segunda. Lleva como título "*RecorridoViewController*".

Su prototipo viene a continuación:

Figura 39. Prototipo de "RecorridoViewController"

La diferencia sustancial con la segunda vista, es que ésta no muestra los nodos sensores registrados, busca la información de la tabla de la base de datos "recorridoX", y a partir del ID introducido en esa tabla, inserta la información del sensor correspondiente en la tabla y lo añade como anotación en el mapa.

La interfaz posee los siguientes métodos y propiedades:

```
//
//  RecorridoViewController.h
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 22/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import "JSONMethods.h"
#import "PlaceMark.h"

@interface RecorridoViewController : UIViewController <UITableViewDataSource,UITableViewDelegate,MKMapViewDelegate>

@property (nonatomic) NSInteger *titleNumber;
@property (nonatomic,strong) IBOutlet MKMapView *oMapa;
@property (nonatomic,strong) IBOutlet UITableView *tableView;
@property (nonatomic) Boolean iniciado;
@property (nonatomic,strong) NSMutableArray *celdas;
@property (nonatomic,strong) PlaceMark *calleMark;

-(void)actualizarRecorrido;

@end
```

Figura 40. Interfaz de "RecorridoViewController"

Como se aprecia en la imagen, tiene añadidos los delegados de la tabla y del elemento mapa. Así se consigue acceso a más métodos además del que tiene como título "*actualizarRecorrido*".

"*actualizarRecorrido"* es llamado a través de otro método, "*performSelector:@selector() withObject: afterDelay:*" ése método, permite ejecutar el método convertido a selector, después de que haya pasado un retardo (double que se introduce en el argumento "*afterDelay:*").

Ésta ejecución no se hace en el hilo principal del dispositivo, si no que éste método, pasa el @selector a otro hilo con un contador de espera (con el valor iniciar el retardo), ésto provoca que la ejecución del resto del programa no se bloquee en el tiempo que el usuario está esperando a que se ejecute la acción.

Si el retardo introducido es muy pequeño, y el método invocado se encarga de realizar una comprobación y añadir algún elemento visual al programa, da al usuario la sensación de que la comprobación se está haciendo a tiempo real.

Los métodos que forman la implementación de la vista son los siguientes:

```
//
//  RecorridoViewController.m
//  MapaProject
//
//  Created by Miguel Sánchez del Águila on 22/08/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "RecorridoViewController.h"

@interface RecorridoViewController ()

@end

@implementation RecorridoViewController

@synthesize titleNumber,oMapa,tableView,iniciado,celdas,calleMark;


- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{...}

- (void)viewDidLoad
{...}

- (void)viewDidUnload
{...}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{...}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{...}

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{...}

-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{...}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{...}

-(void)actualizarRecorrido{...}


@end
```

Figura 41. Implementación de "RecorridoViewController"

El método "*cellForRowAtIndexPath:*" es el que marca el contenido que van a llevar las celdas de la tabla.

En este caso, el contenido es el siguiente:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    //Código eficiente
    static NSString *identificadorCelda = @"CeldaPunto";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:identificadorCelda];

    if(cell == nil){
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:identificadorCelda];
    }
    // Configure the cell...
    calleMark = [celdas objectAtIndex:indexPath.row];
    cell.textLabel.text = calleMark.place.name;
    UIFont *miFuente = [UIFont fontWithName:@"Helvetica" size:12.0f];
    cell.textLabel.font = miFuente;
    NSString *imag = calleMark.place.imagen;
    [cell.detailTextLabel setText:imag];

    return cell;
}
```

Figura 42. Función "cellForRowAtIndexPath" de "RecorridoViewController"

Para que la tabla se rellene de forma dinámica, el elemento de donde proviene la fuente de datos, debe ser un array.

Para el ejemplo, el array utilizado es el elemento "celdas".

Ésta vista posee también un botón que cierra el ciclo del programa. Cuando el usuario haya terminado de ver el recorrido deseado, si pulsa ese botón, volverá a la vista principal inicial, y tendrá la opción de salir del programa, o de volver a iniciar el proceso.

La siguiente carpeta que aparece en la jerarquía del programa "*Frameworks*" contiene los elementos extra que forman parte de la API "*Cocoa Touch*" pero que no están contenidos en la librería estándar.

Un ejemplo de ello es el componente mapa "*MKMapKit*" y el conjunto de elementos y herramientas para trabajar con él.

El contenido de la carpeta "*Frameworks*" del proyecto es:

Figura 43. Contenido de la carpeta "Frameworks"

El "*UIKit.framework*" contiene el conjunto de elementos básicos que conforman una aplicación para iPhone, desde cajas de texto, botones, etiquetas, hasta los gestos que reconoce el digitalizador del dispositivo.

La última carpeta de la jerarquía del programa "*Products*" contiene el paquete funcional de la aplicación (para el proyecto, el paquete resultante es "*MapaProject.app*").

Para testear esta aplicación en un dispositivo real, se necesita una licencia de desarrollador de Apple que esté inscrita en alguno de los programas para desarrolladores.

Los programas que oferta Apple son los siguientes:

- iOS Developer Program Individual
- iOS Developer Program Company
- iOS Developer University Program

Los dos primeros cuestan cien dólares al año, el tercero es gratuito, pero debe registrarse en él una persona con capacidad para registrarse en nombre de una escuela universitaria.

La ETSIT está inscrita en este programa, por lo que se podría probar esta aplicación en dispositivos reales.

3.4 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.

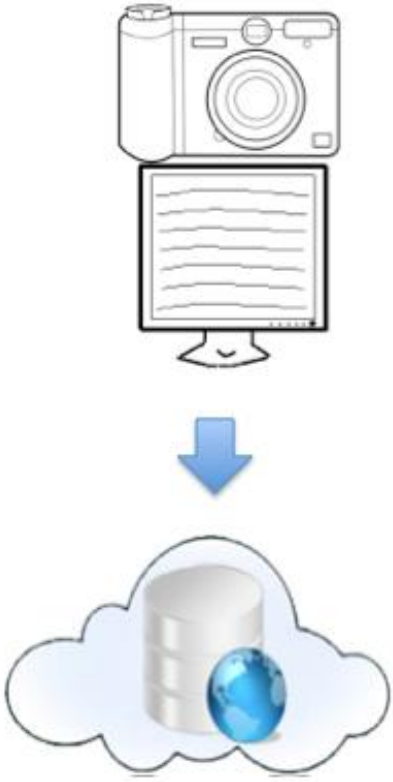


Figura 44. Interacción 1

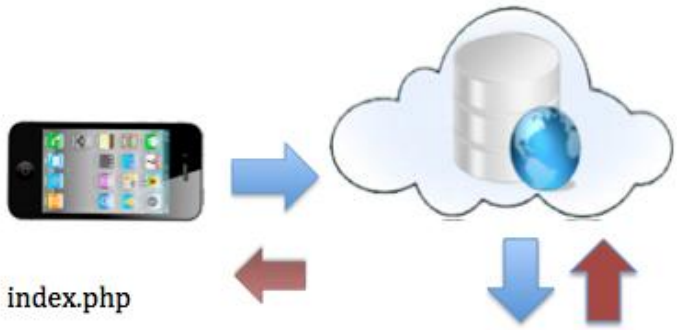
El servidor comprueba el nombre introducido en la tabla "coordenadas" de la base de datos:

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 45. Tabla coordenadas

En caso afirmativo, 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”.



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 46. Interacción 2

En la aplicación se representa la información de la tabla “coordenadas”, cuando el usuario decide desde que equipo comienza la búsqueda. El usuario transmite al servidor ese dato, y éste lo retransmite a los equipos.

El equipo gestor objetivo, indica a sus nodos sensores que comiencen a almacenar las firmas de los vehículos que circulen sobre ellos, también se crea la tabla “firmasX” en la base de datos.

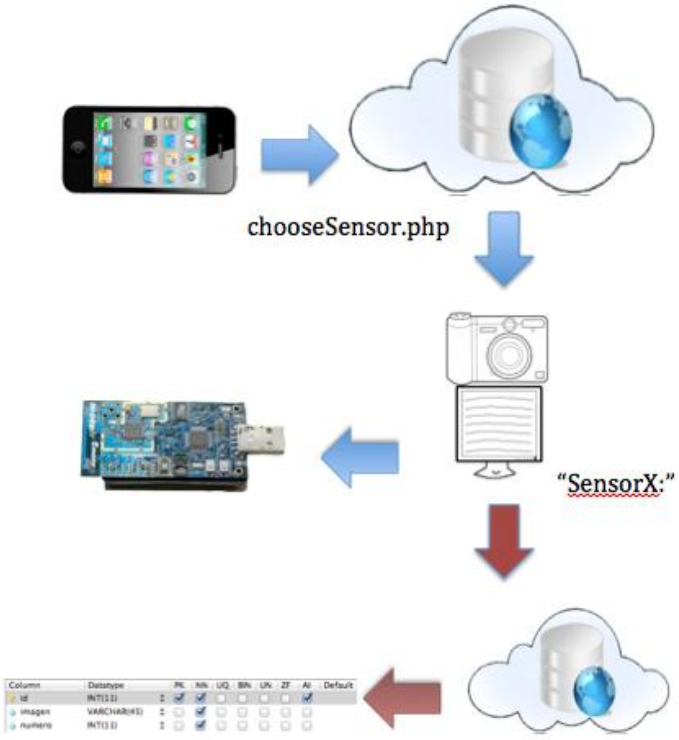


Figura 47. Interacción 3

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”.

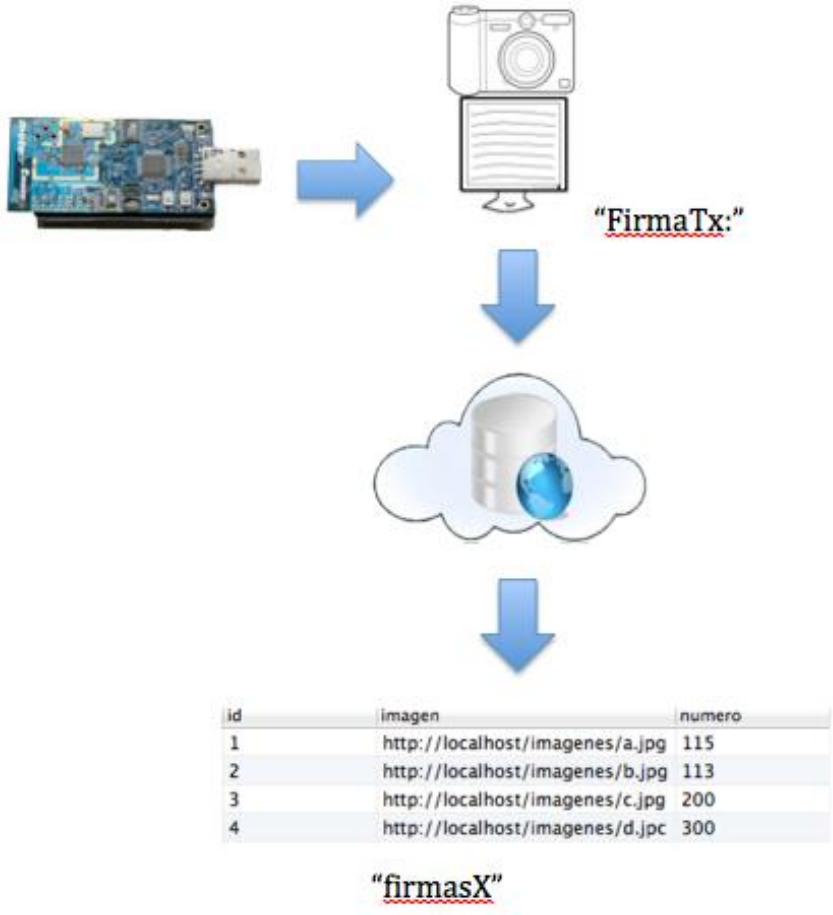


Figura 48. Interacción 4

A los tres segundos desde que el usuario selecciona el sensor objetivo, se hace una consulta al script que envía las firmas almacenadas en la tabla “firmasX” (donde X es un entero que se envía tipo POST) hasta la aplicación.

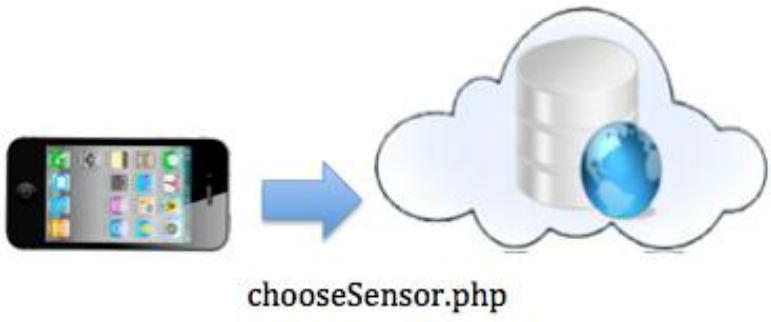


Figura 49. Interacción 5

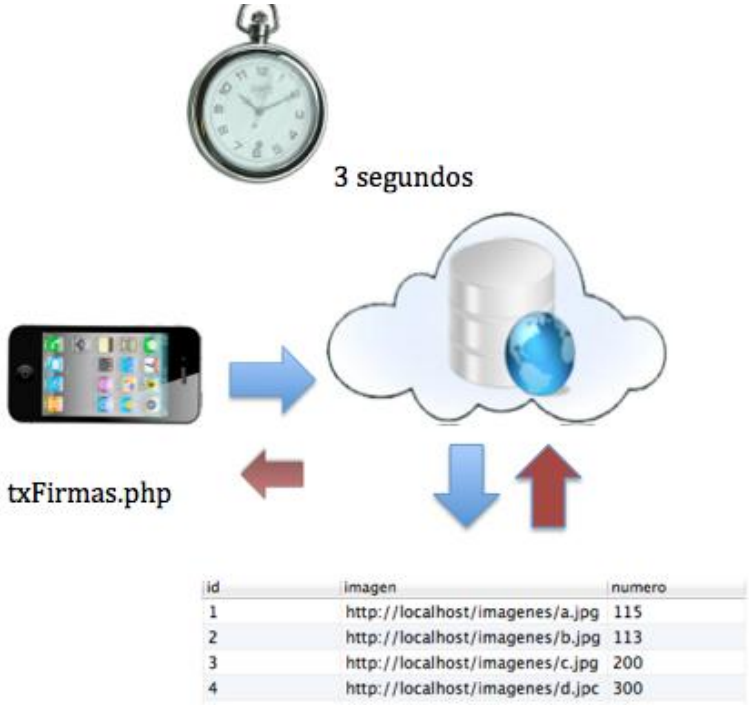


Figura 50. Interacción 6

Si el usuario presiona el botón "refrescar" se vuelve a esperar los tres segundos.
En cambio, si 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.

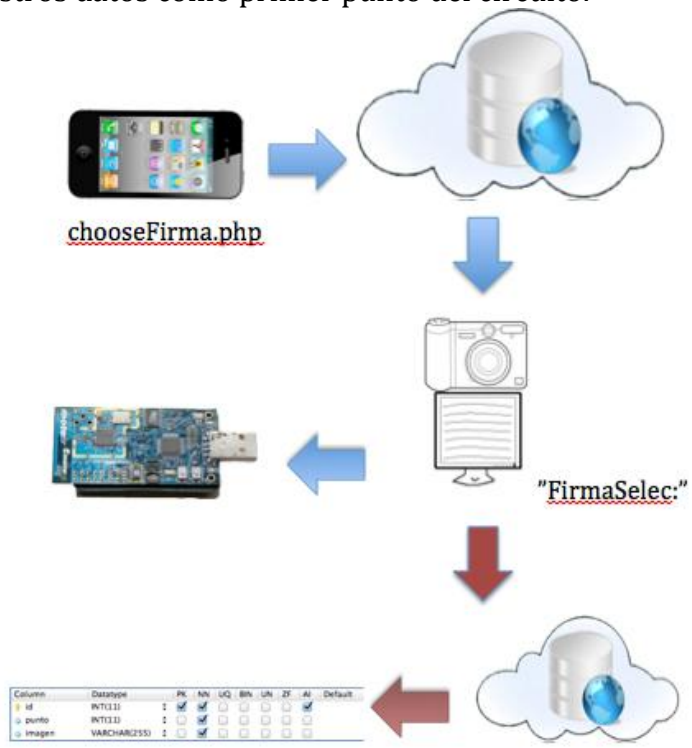


Figura 51. Interacción 7

Después de esto, el usuario 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 que están capturando en ese momento.

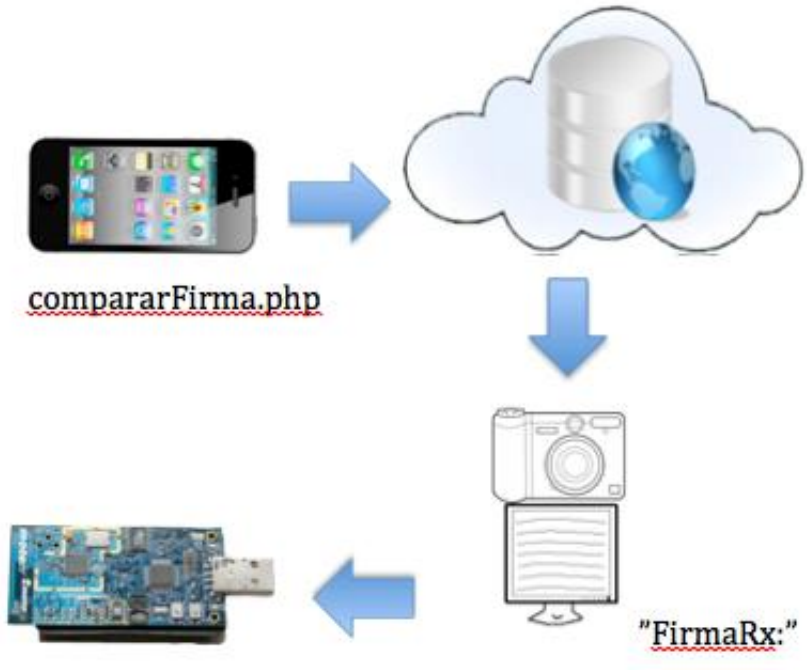


Figura 52. Interacción 8

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

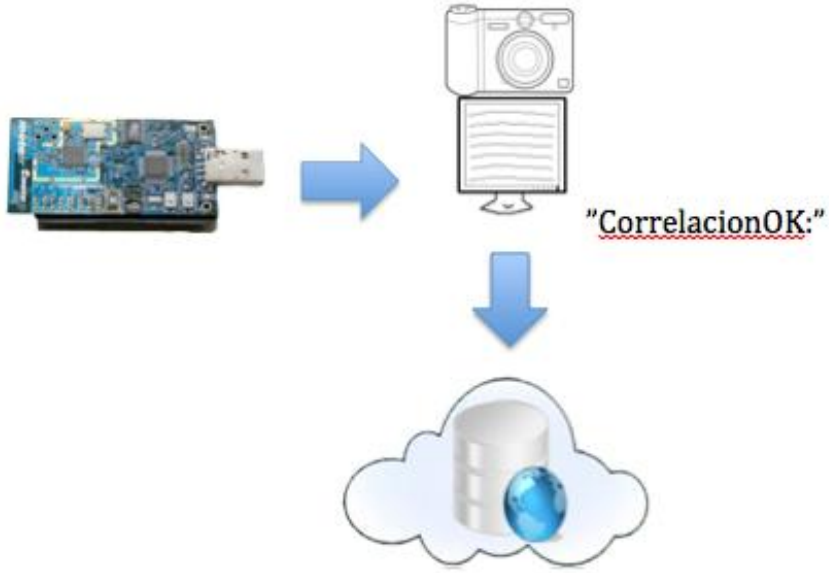
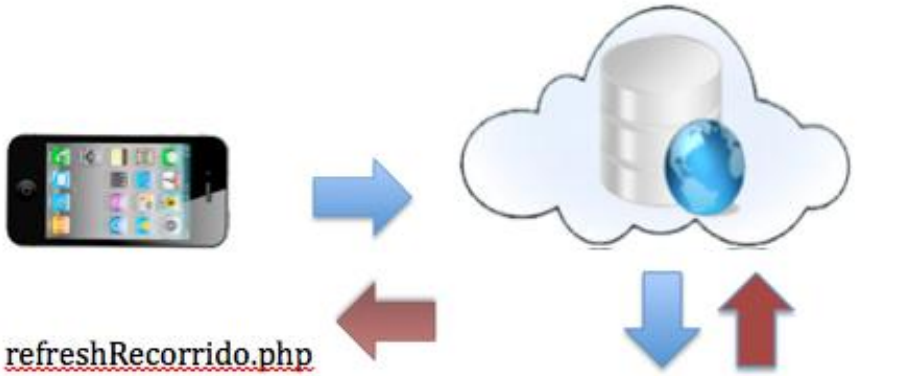


Figura 53. Interacción 9

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 54. Interacción 10

4. Manual de usuario

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 55. 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.

Puede cancelar el proceso de cierre de la aplicación pulsando el botón "Volver" que aparece en esa ventana.

Figura 56. Saliendo del programa

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.

La única manera de navegar entre vistas es pulsando los botones azules circulares que hay en cada una de las entradas de la tabla, junto con los botones que aparecen en la barra de navegación superior del programa.

Figura 57. 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”.

Pasados tres segundos, aparecen las firmas de los coches que haya almacenado ese sensor.

Figura 58. Lista de firmas de un sensor

Contenido de la tabla:

- Foto del vehículo
- ID del vehículo en la lista de firmas del sensor
- Identificador único para cada firma

El usuario puede darle al botón de refrescar, y tres segundos después vuelve a aparecer la vista con la tabla actualizada.

Cuando se selecciona una de las firmas, se pasa a la siguiente ventana, muy similar a la vista de selección del sensor inicial:

Figura 59. 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.

Con los botones de la barra superior, puede volver hacia atrás si pulsa el de la izquierda, o puede finalizar el recorrido y volver a la vista inicial, si presiona el de la derecha.

Figura 60. Barra de navegación del programa

El resto de programas no poseen interacción por parte del usuario, son procesos automáticos que dependen de los valores que reciben, por tanto no es necesario hacer un manual sobre el funcionamiento de éstos procesos.

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".

Aún está pendiente la unificación de los dos proyectos.

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.

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.
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 una demostración de una posible comunicación entre un servidor y un iPhone.

- ¿Se le ocurre alguna ampliación al sistema?

Se me ocurren varias, en primer lugar, esta 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 logueo, 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.

Otra ampliación, podría ser, implementar un mecanismo para que cuando un usuario seleccionase una firma determinada de uno de los equipos gestores, en vez de enviar la señal a comparar a todos los nodos sensores, enviarla a los nodos sensores que estuviesen dentro de un radio circular de determinado rango, con

esto se conseguiría un ahorro energético bastante considerable, y una mayor fluidez del sistema.

Otra posible ampliación, podría ser mejorar la API de las funciones que se utiliza en la aplicación móvil para que respeten los principios de una aplicación RESTFUL-

Para obtener más información sobre un programa RESTFUL, se puede consultar la sección "*Anexos*" de la memoria.

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. Probablemente ya exista un sistema similar.

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, el contenido que circula por 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, se podría portar la aplicación a los terminales Android o Windows Phone, para que con dispositivos económicos y comunes, 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] 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 Desarrollo de una aplicación sencilla para iPhone

Éste sería un tutorial para dar los primeros pasos en el desarrollo de aplicaciones para dispositivos iOS.

Para el desarrollo de esta aplicación se va a utilizar el entorno de desarrollo proporcionado por Apple "XCode 4" el cuál implementa la nueva herramienta para el diseño de las vistas y sus transiciones, los llamados "StoryBoards".

El primer paso para crear una aplicación para iPhone es descargar las herramientas necesarias.

Para ello, vamos al App Store e ingresamos con nuestro Apple ID y nuestra contraseña, en la barra de búsqueda, buscamos "XCode" que es el entorno de programación que proporciona Apple, una herramienta más que útil para cualquiera que posea un equipo de la marca.

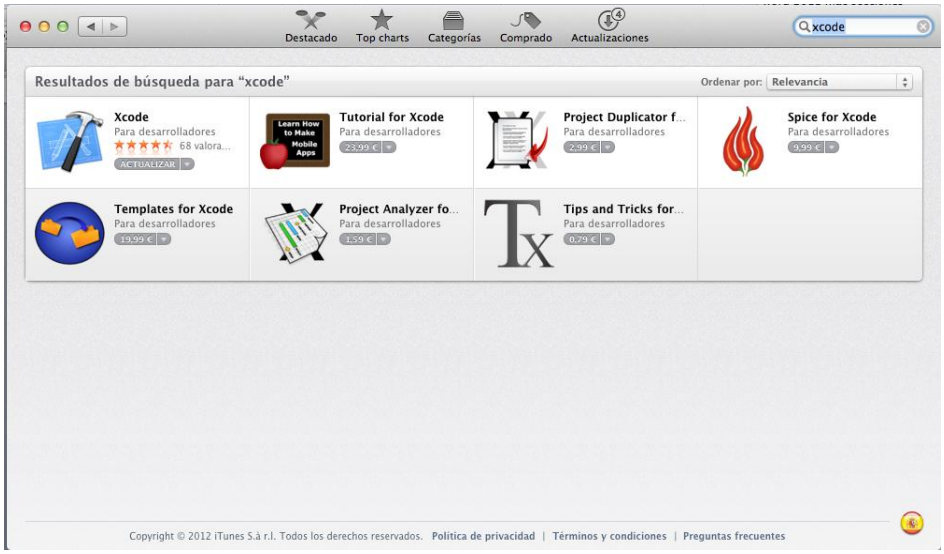


Figura 61. Resultado de la búsqueda XCode en la App Store de Apple

Pulsamos sobre el icono del martillo y el plano azul:

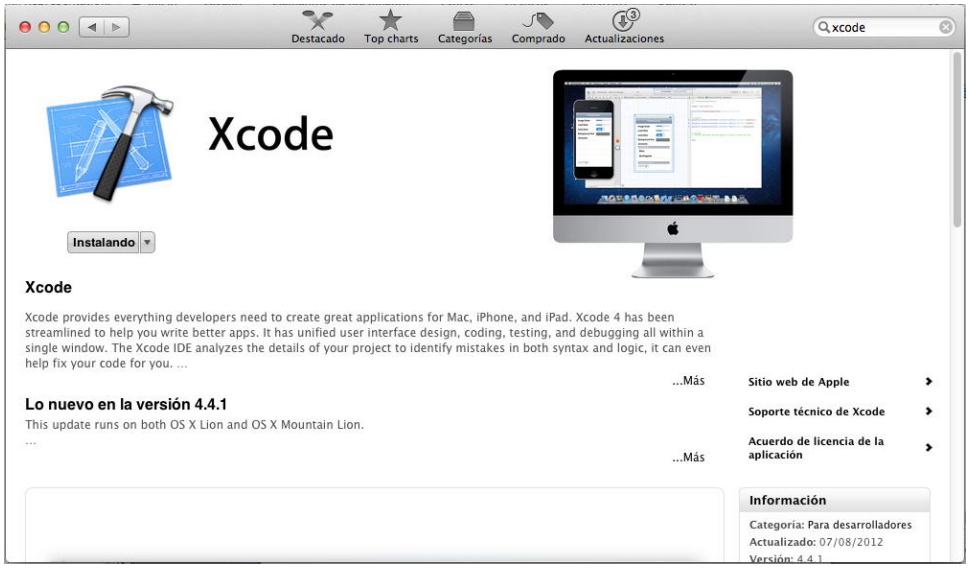


Figura 62. Sección de XCode de la App Store

Mientras se instala, aparece una barra de progreso:

Figura 63. Barra de progreso de la instalación

Una vez instalado, aparece una ventana similar a esta, pero con la lista de proyectos recientes vacía:

Figura 64. Vista inicial de XCode

Pulsamos sobre “Create a new XCode Project” y aparece la siguiente ventana.

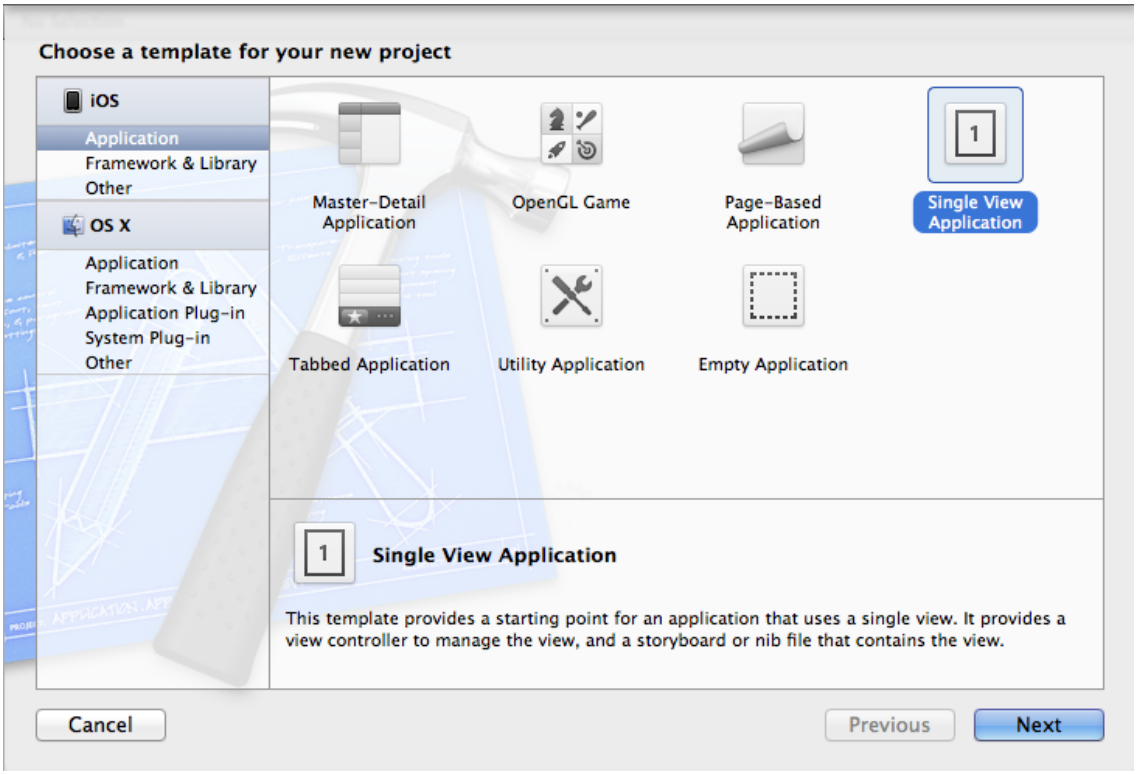


Figura 65. Selección de plantilla

En esta vista podemos seleccionar una plantilla para nuestro proyecto, lo cuál facilita la labor de hacer la base cuando se trata de un gran proyecto.

Para comenzar, seleccionamos “Single View Application” que es la plantilla que permite hacer la aplicación más básica.

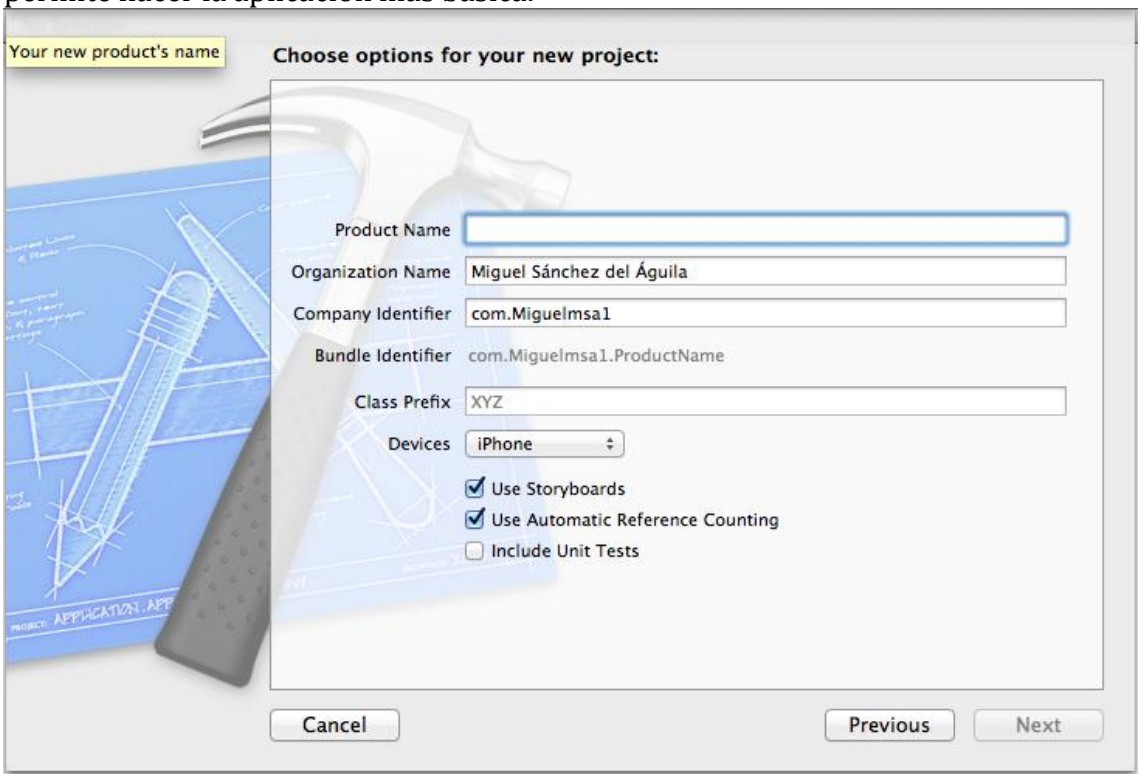


Figura 66. Nombre del proyectFigura o.

Muy importante, en esa ventana marcar las dos casillas de la parte inferior que se ven en la captura.

La primera nos permite utilizar los StoryBoards para el diseño de la aplicación, y la segunda activa también una nueva cualidad que posee ésta versión de XCode, “ARC” un recolector de basura automático, similar al de Java.

En este ejemplo vamos a hacer una calculadora sencilla, por tanto, el nombre del proyecto puede ser “proyectoCalculadora”, seleccionamos la carpeta donde se almacenará el proyecto, y seguimos.

En este punto, llegamos a la vista principal de XCode:

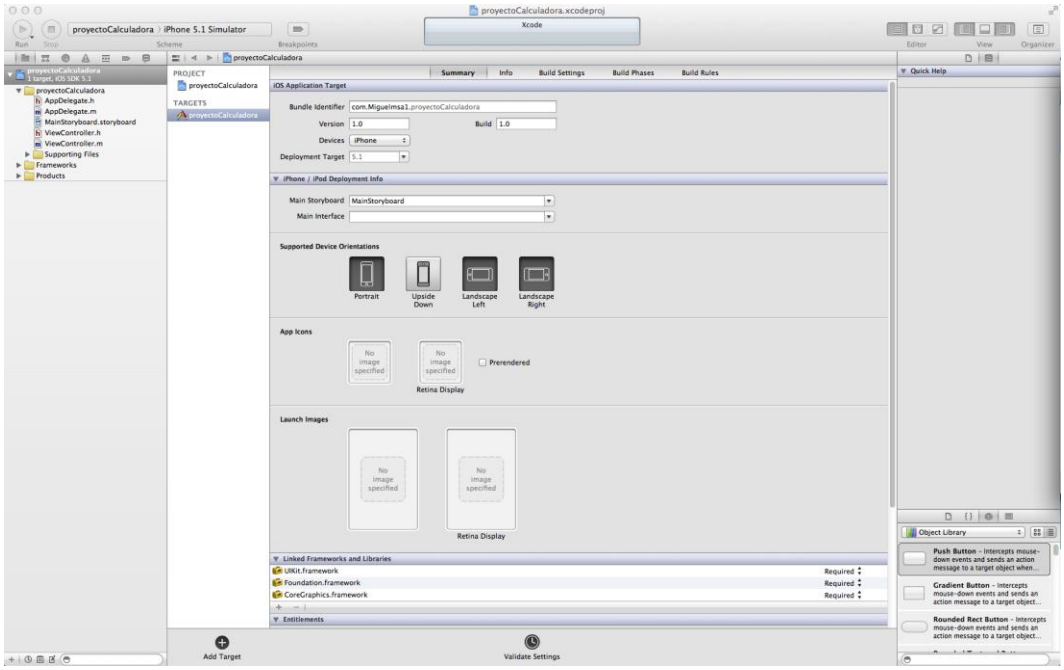


Figura 67. Vista principal de XCode

Desde aquí podemos definir varias características del programa, como la orientación de los acelerómetros que soporta (vertical, horizontal), la imagen de icono de la aplicación, los frameworks que necesitaremos, y un montón de opciones más complejas.

El primer paso será diseñar el apartado gráfico de la aplicación, para ello, desde la barra de la izquierda (barra de navegación a partir de ahora) seleccionamos "MainStoryboard.storyboard".

Aquí encontramos una ventana en blanco, la que será el aspecto de la vista inicial de nuestra aplicación.

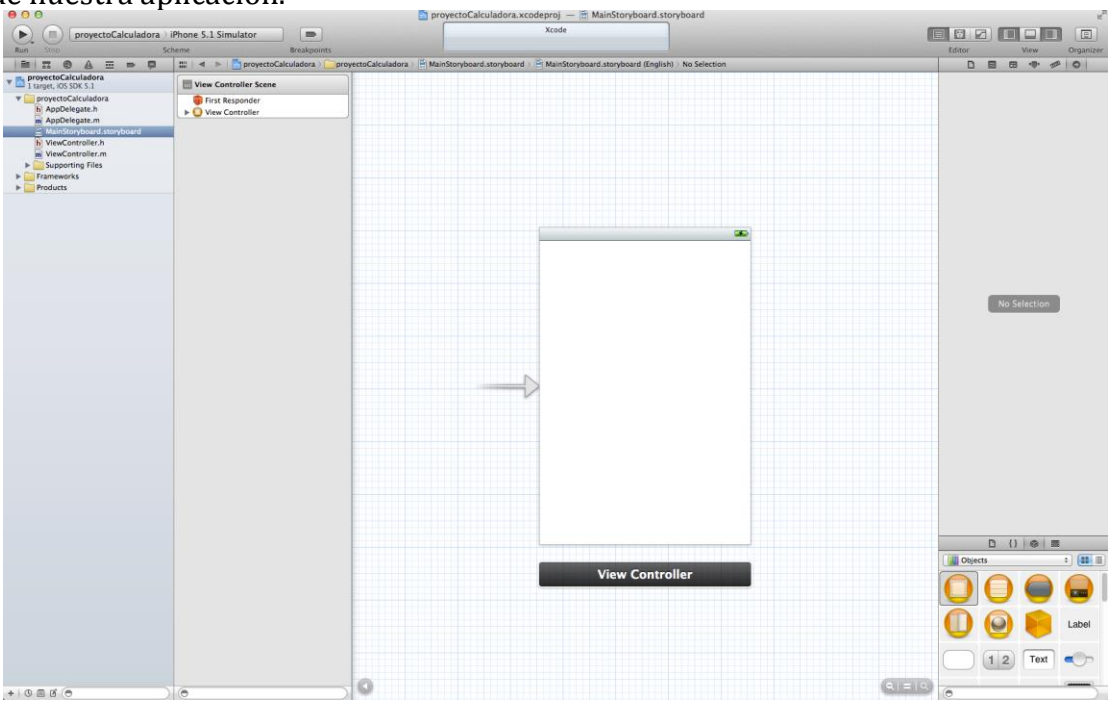


Figura 68. Vista inicial del Storyboard

Como se ha dicho anteriormente, el objetivo de éste ejemplo es hacer una calculadora, por tanto, vamos a darle un aspecto más atractivo a nuestra vista. Para ello pulsamos una vez sobre la vista, y se activa la barra de herramientas (barra derecha).

Para seleccionar la vista, también podemos hacerlo sobre la barra de elementos:

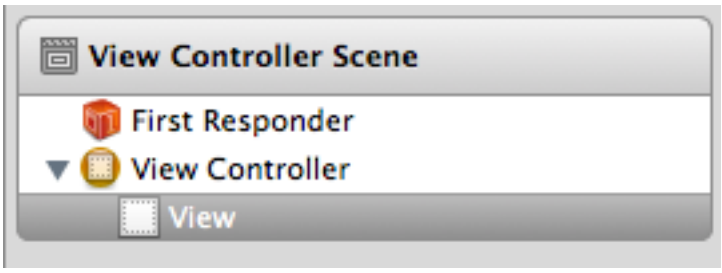


Figura 69. Elementos del Storyboard

En la barra de herramientas, aparece el campo “background” donde seleccionamos el color de fondo de la vista.

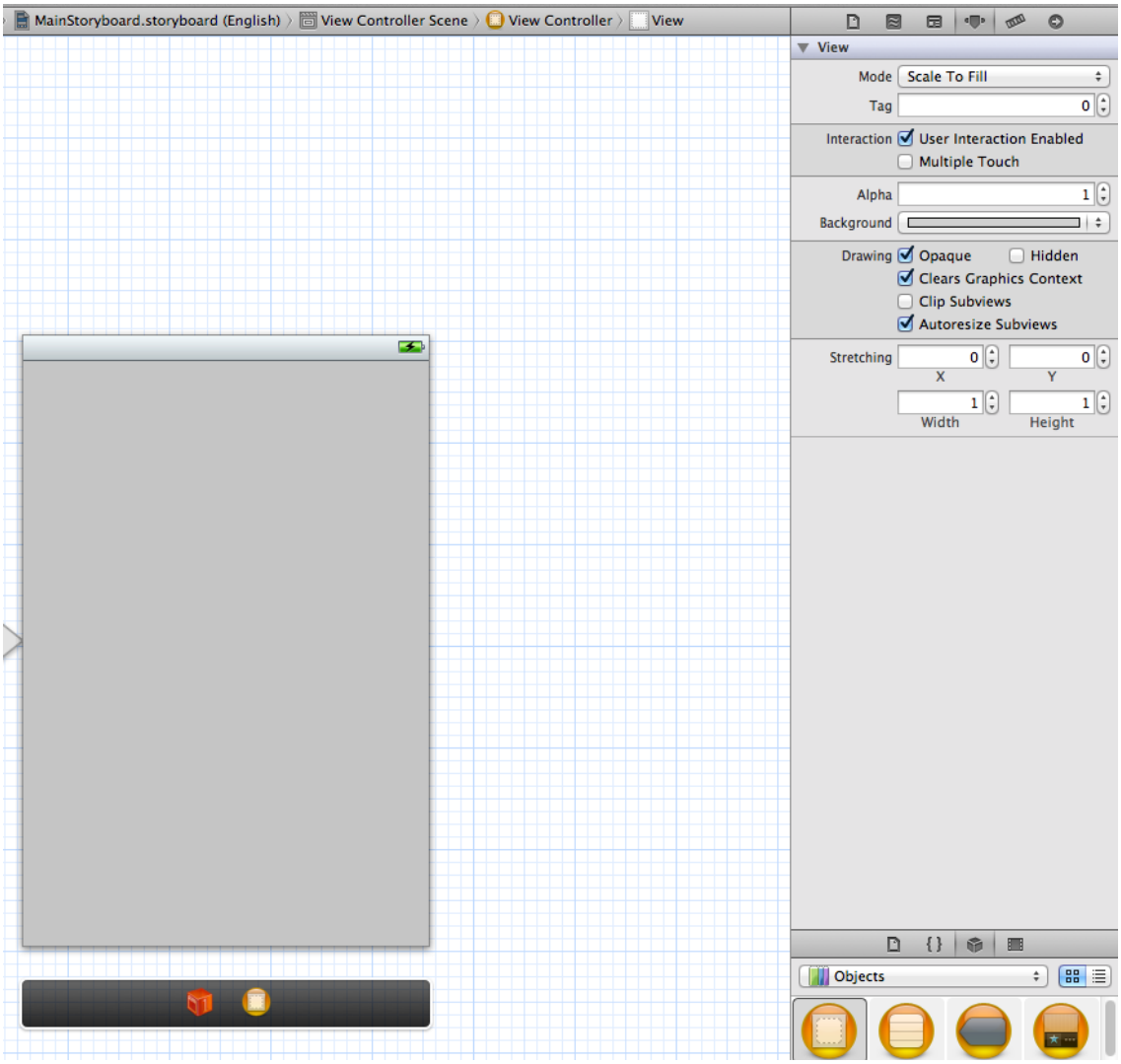


Figura 70. Definir color de fondo de la vista

Después de esto añadimos los botones necesarios para nuestra aplicación, para ello arrastramos desde el UIKit el elemento UIButton hasta nuestra vista.

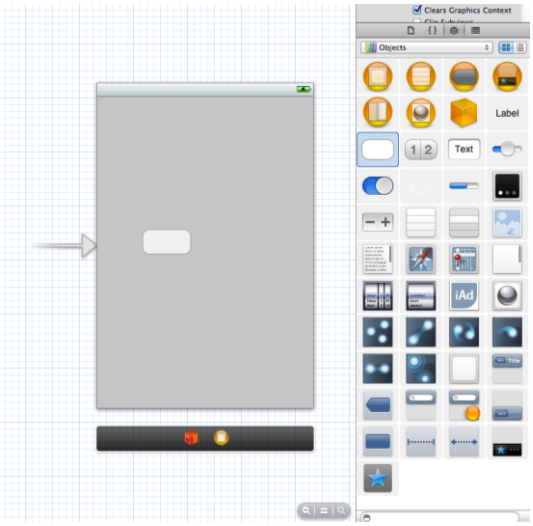


Figura 71. Añadiendo botones a nuestra vista

Una vez hemos repartido los botones a nuestro gusto, añadimos el elemento UILabel, que será la caja de texto donde aparecerán los números pulsados y los resultados de las operaciones:

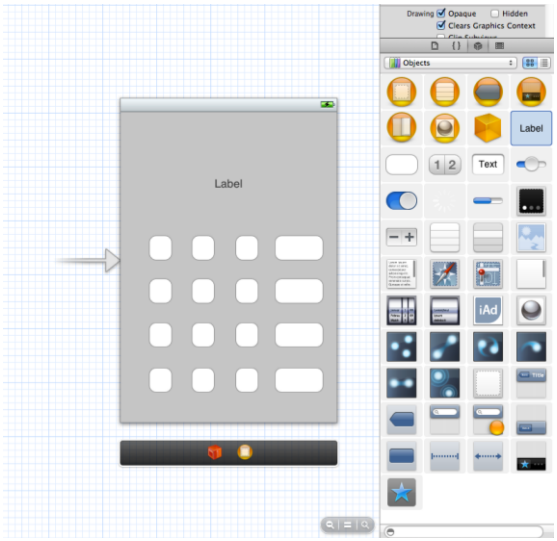


Figura 72. Agregando el UILabel a nuestra vista

Haciendo doble click sobre los elementos, podemos definir el texto que contienen, intentamos darle un formato así a nuestra calculadora:

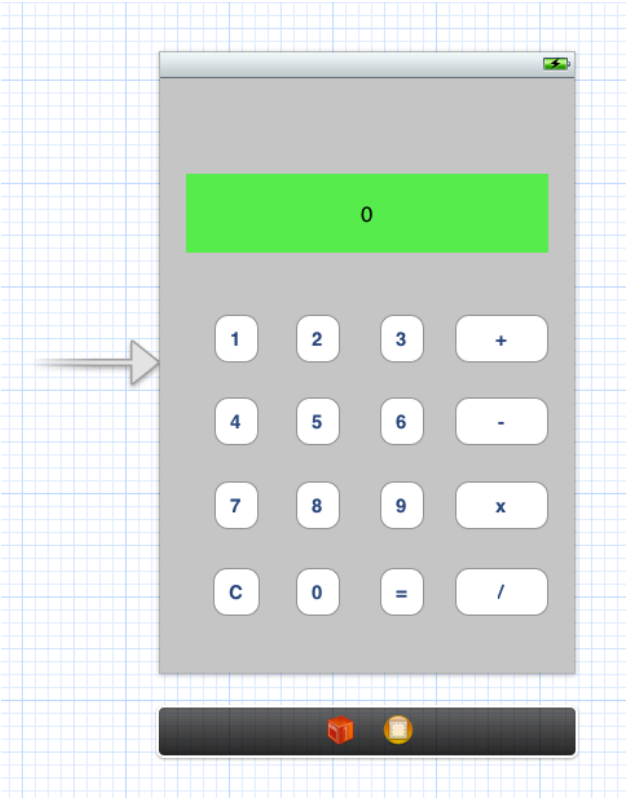


Figura 73. Parte gráfica de la calculadora terminada

La clase que va a regir nuestra vista se llama “ViewController” para confirmarlo o cambiarlo, tenemos el siguiente campo:

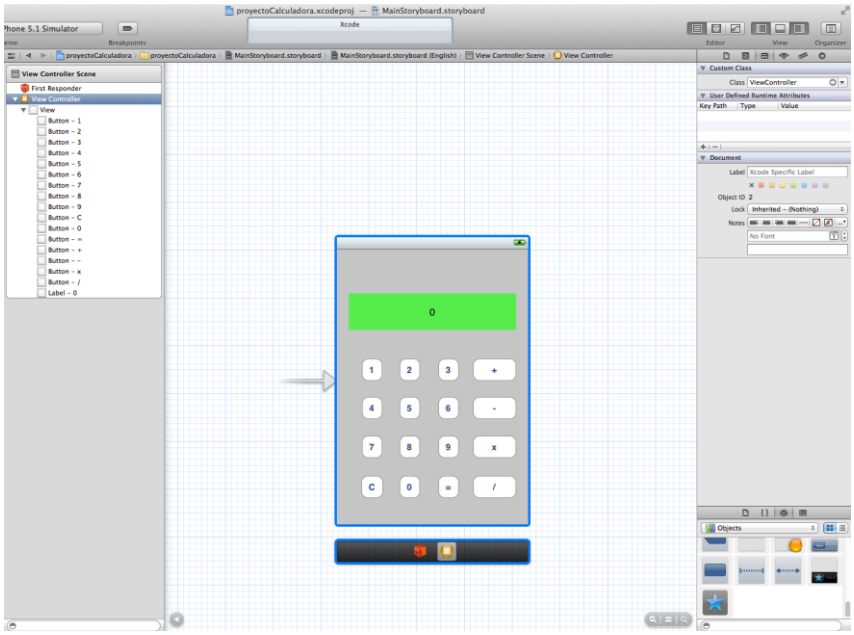


Figura 74. Selección de la clase de nuestra vista

Arriba a la derecha aparece el cuadro de diálogo que selecciona la clase de la vista.

Pasemos ahora a rellenar la interfaz de la clase, añadimos los mismos objetos que hemos añadido en el Storyboard en la interfaz “ViewController.h”, añadimos también un entero y un string auxiliar, más adelante entenderemos el por qué:

```
//
//  Created by Miguel Sánchez del Águila on 30/08/12.
//  Copyright (c) 2012 Miguel Sánchez del Águila. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (nonatomic,strong) IBOutlet UIButton *but0,*but1,*but2,*but3,*but4,*but5,*but6,*but7,*but8;
@property (nonatomic,strong) IBOutlet UIButton *but9,*butC,*butIgual,*butMas,*butMenos,*butMult,*butDiv;

@property (nonatomic,strong) IBOutlet UILabel *caja;

@property (nonatomic,strong) NSString *cadena;
@property (nonatomic) int entero;
```

Figura 75. Interfaz de ViewController

Después de esto, el siguiente paso es añadir estos objetos a nuestra implementación utilizando "synthesize"

```
//  Created by Miguel Sánchez del Águila on 30/08/12.
//  Copyright (c) 2012 Miguel Sánchez del Águila. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

@synthesize but0,but1,but2,but3,but4,but5,but6,but7,but8,but9,butC,butDiv;
@synthesize butIgual,butMas,butMenos,butMult,caja,cadena,entero;

- (void)viewDidLoad
{
    [super viewDidLoad];
```

Figura 76. Sintetizando objetos

Pasamos a enlazar los objetos agregados de manera visual, con los objetos declarados en la interfaz, para ello, nos vamos al Storyboard, y realizamos lo siguiente:

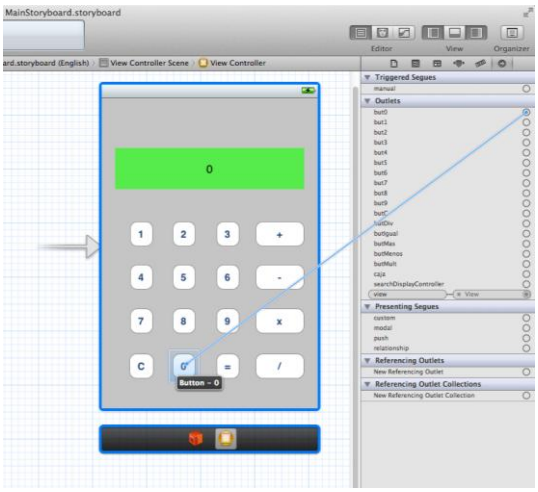


Figura 77. Enlazando el código con la parte gráfica

Realizamos esto hasta tener enlazados todos los objetos de la vista.

Definir las acciones que realizan los botones, para ello, ponemos el modo "doble editor" con este botón:

Figura 78. Opción doble editoFigura r.

El programa ahora ha quedado dividido en dos partes, tenemos el Storyboard por un lado, y la declaración de la interfaz por otro.

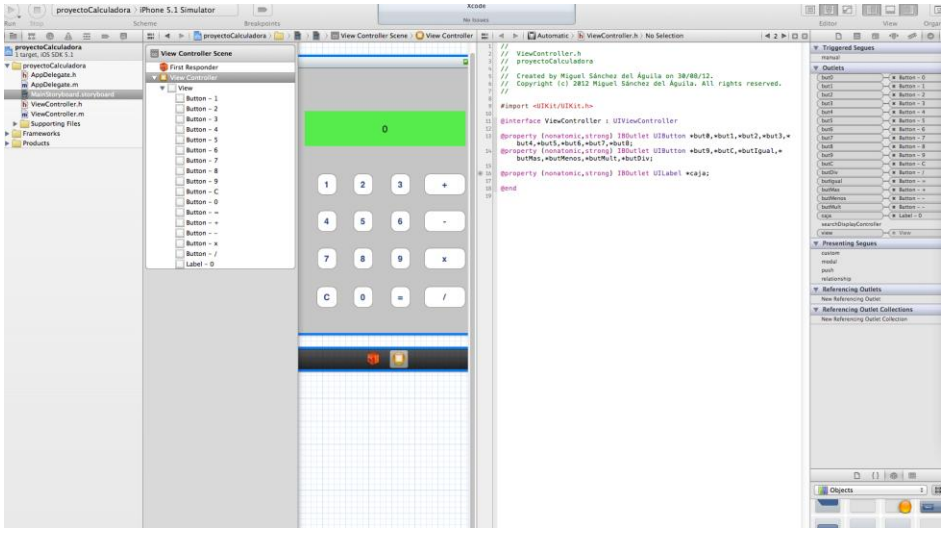


Figura 79. XCode en modo doble editor

Para asignar acciones a los botones, podemos realizarlo de la siguiente manera:

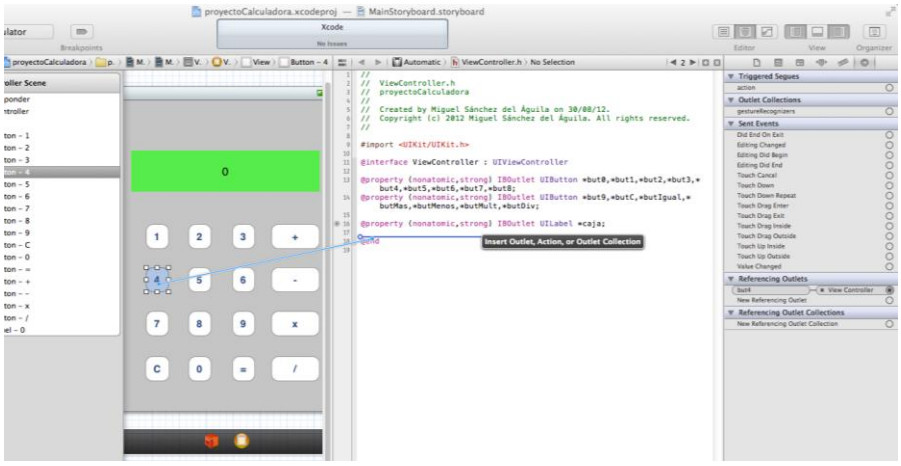


Figura 80. Crear un IBAction (paso 1)

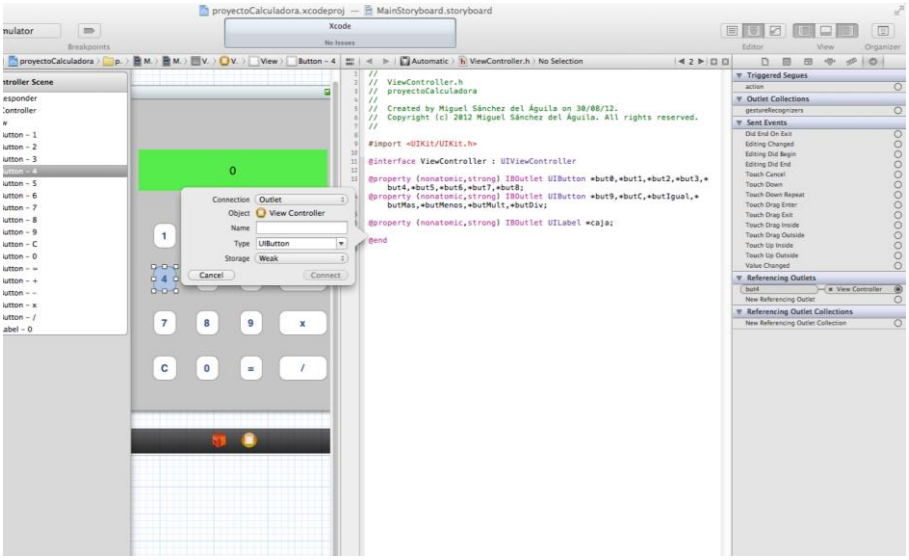


Figura 81. Crear un IBAction (paso 2)

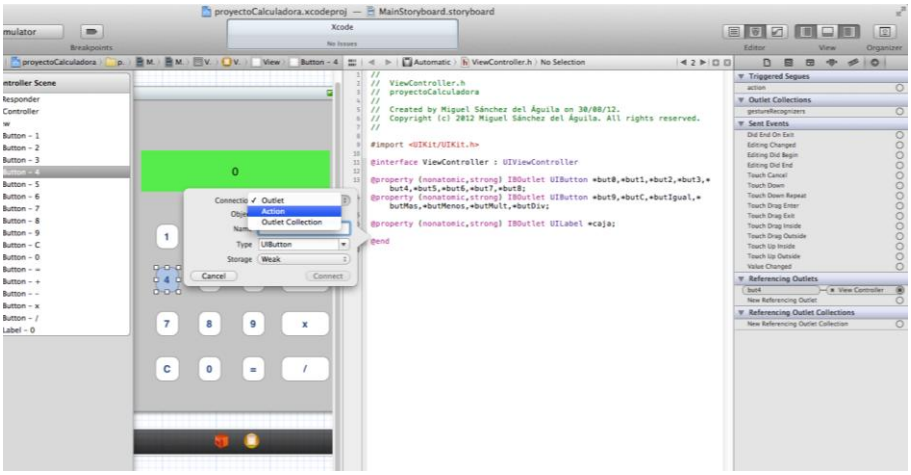


Figura 82. Crear un IBAction (paso 3)

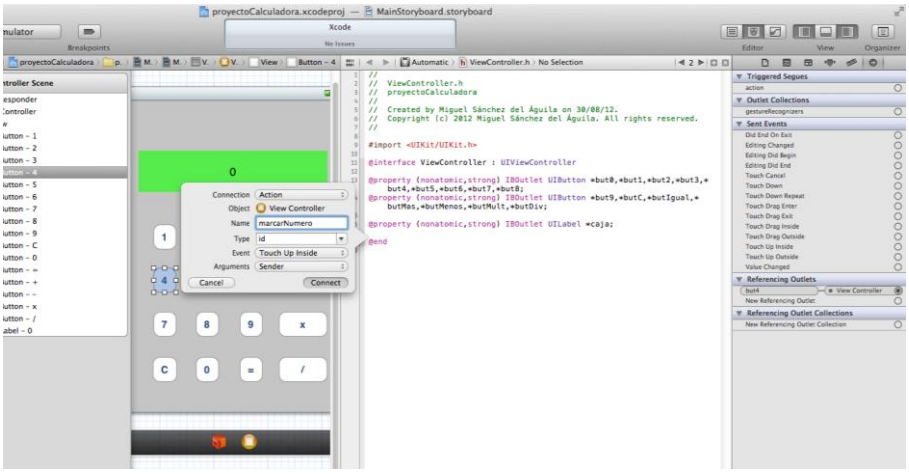


Figura 83. Crear un IBAction (paso 4)

Con esto se crea un IBAction a falta de implementarlo, si vamos a la ventana de la implementación de la clase, también veremos el esqueleto vacío de la IBAction definida.

```
16  @property (nonatomic,strong) IBOutlet
17
18  - (IBAction)marcarNumero:(id)sender;
19  @end
20
```

Figura 84. IBAction creada

Agregamos el resto de números a la función "marcarNumero"

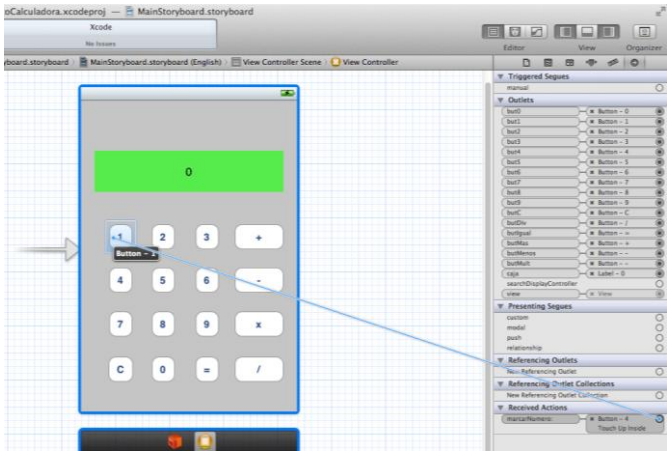


Figura 85. Enlazando la acción a más números (paso 1)

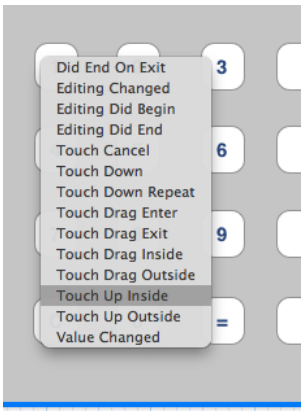


Figura 86. Enlazando la acción a más números (paso 2)

Así añadimos los 10 números a la función.
Siguiendo los mismos pasos, creamos las siguientes funciones:

```
- (IBAction)marcarNumero:(id)sender;
- (IBAction)limpiarPantalla:(id)sender;
- (IBAction)operacion:(id)sender;
- (IBAction)igual:(id)sender;
@end
```

Figura 87. Diferentes acciones creadas para los botones

Asignamos “limpiarPantalla” al botón butC, “operación” a los cuatro botones de operación, e “igual” a butIgual.

Ahora debemos realizar la implementación de las funciones, más abajo tenéis el código para realizarlo vosotros mismos.

Función “marcarNumero”:

```
- (IBAction)marcarNumero:(id)sender{
    UIButton *aux = sender;
    NSString *str = caja.text;
    NSString *send = aux.titleLabel.text;
    NSString *resp = [[NSString alloc]initWithFormat:@""];
    if([str isEqualToString:@"0"]) caja.text = aux.titleLabel.text;
    else {
        resp = [NSString stringWithFormat:@"%@%@",str,send];
        caja.text = resp;
    }
}
```

Figura 88. Función marcarNumero

En esta función realizamos lo siguiente:
Tomamos el botón que ha invocado a la función (sender), almacenamos el valor de nuestra caja de texto actual, y el valor del botón pulsado.

Hacemos la comparación de la cadena contenida en nuestra caja de texto, si es igual a cero, se reemplaza el valor por el pulsado, si por el contrario, el valor de la caja de texto no es cero, se concatena el valor con el que hubiera en el instante anterior.

Función “limpiarPantalla”:

```
- (IBAction)limpiarPantalla:(id)sender{
    caja.text = @"0";
}
```

Figura 89. Función limpiarPantalla

Es una función muy sencilla, coloca un “0” en la caja de texto.

Función “operación”:

```
- (IBAction)operacion:(id)sender{
    UIButton *au = sender;
    cadena = au.titleLabel.text;
    entero = [caja.text integerValue];
    caja.text = @"0";
}
```

Figura 90. Función operacion

Lo primero que realiza ésta función, es almacenar el botón que la ha invocado.
Extrae el signo que contiene ese botón y aquí entran en juego las dos variables que hemos definido en la interfaz, el string y el entero.

El string “cadena” contendrá el signo de operación que ha seleccionado el usuario, y el entero, el primer operando de la operación.

Después de esto se limpia la pantalla para que el usuario pueda introducir el segundo operando.

Una vez que el usuario ha introducido los dos operandos y ha seleccionado la operación que desea que se realice, debe presionar el botón “igual” para obtener la solución.

La función enlazada a ese botón es la siguiente:

```
- (IBAction)igual:(id)sender{
    int op2 = [caja.text integerValue];
    if ([cadena isEqualToString:@"+"]){
        caja.text = [NSString stringWithFormat:@"%d",entero+op2];
    }else if ([cadena isEqualToString:@"-"]){
        caja.text = [NSString stringWithFormat:@"%d",entero-op2];
    }else if ([cadena isEqualToString:@"x"]){
        caja.text = [NSString stringWithFormat:@"%d",entero*op2];
    }else if ([cadena isEqualToString:@"/"]){
        caja.text = [NSString stringWithFormat:@"%f",(float)entero/op2];
    }
}
```

Figura 91. Función igual

Lo que hace ésta función es almacenar el valor del segundo operando, que es el valor que posee en este momento la caja de texto, lo convierte de string a entero, y compara el valor del botón operación que había presionado el usuario anteriormente.

Para el caso de la división, al poder contemplar resultados no enteros al dividir dos números, el formato del resultado lo hemos establecido como “double”, así podremos representar los posibles decimales que resulten de la operación.

En este punto, nuestra calculadora ya es funcional, para hacer la comprobación en el simulador de iOS, sólo tendremos que pulsar el botón “play” situado en la barra de herramientas superior de XCode.

Figura 92. Barra de herramientas superior de XCode

Figura 93. Botones de ejecución o pausa de la simulación del proyecto

Así podremos obtener un resultado similar a éste:

Figura 94. Ejemplo de calculadora funcional

Para continuar con el ejemplo, vamos a añadir a nuestra calculadora una segunda vista, y así explicar el tipo de transición simple entre vistas.

Para ello, vamos al Storyboard, y agregamos una segunda vista:

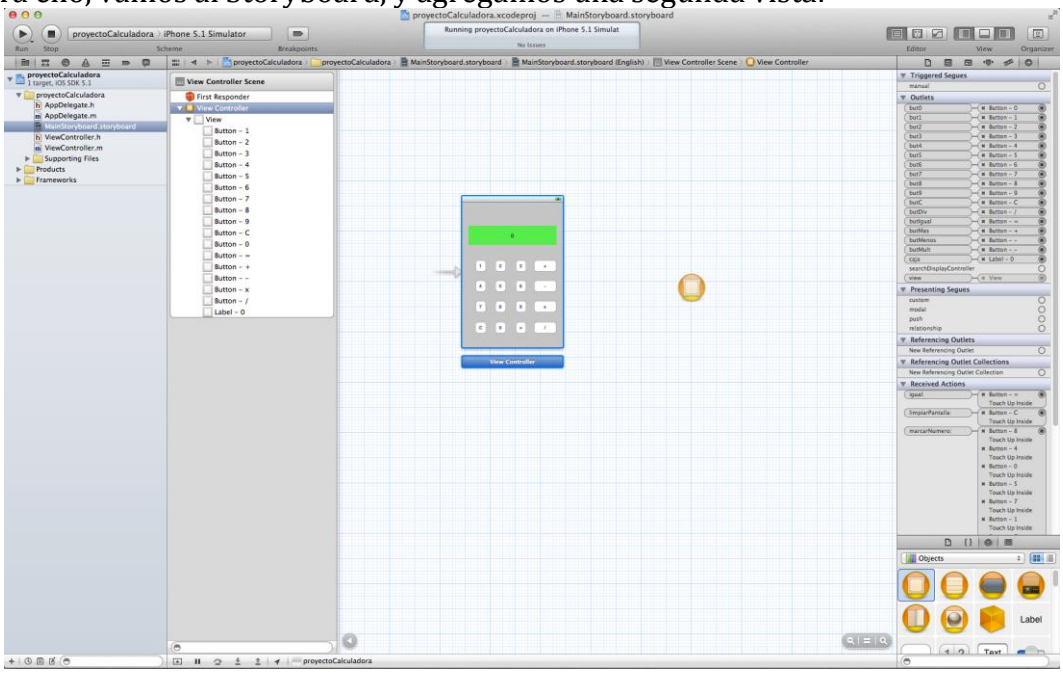


Figura 95. Agregando una segunda vista a nuestro Storyboard (paso 1)

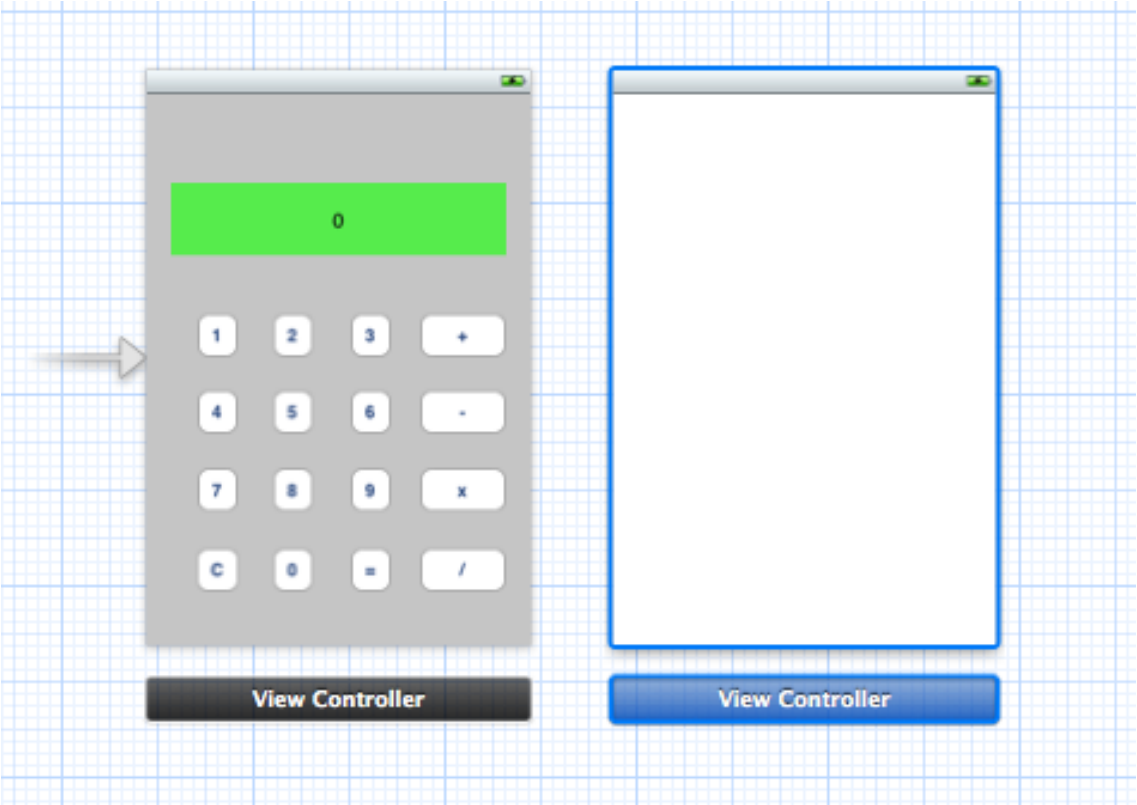


Figura 96. Agregando una segunda vista a nuestro Storyboard (paso 2)

Una vez tenemos las dos vistas, vamos a decidir para que usar la segunda, un posible ejemplo, la información básica de la aplicación: versión, autor, etc.

Le damos el formato deseado a nuestra vista:

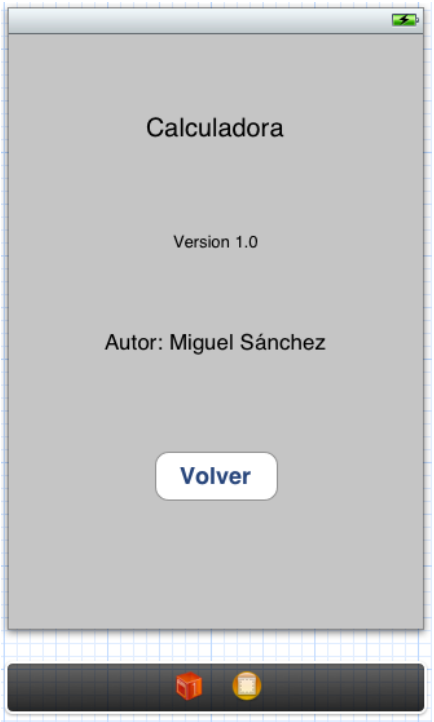


Figura 97. Ventana de información de la calculadora

Debemos añadir también el botón que pasará a esta ventana de información, para ello, arrastramos un UIButton a la vista de la calculadora, y en su barra de herramientas seleccionamos el "Type" "Info Dark"

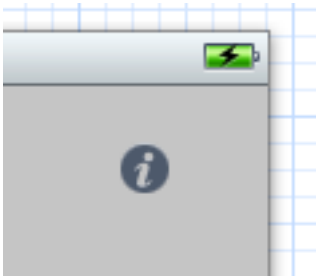


Figura 98. UIButton con estilo "Info Dark"

Para crear transiciones entre vistas, el proceso es similar a crear acciones, pulsamos con el segundo botón sobre el botón que será el invocador de la transición y lo llevamos a la vista correspondiente:

Figura 99. Creando una transición (paso 1)

Aquí aparece un menú contextual, seleccionamos la opción "modal" que será la transición simple.

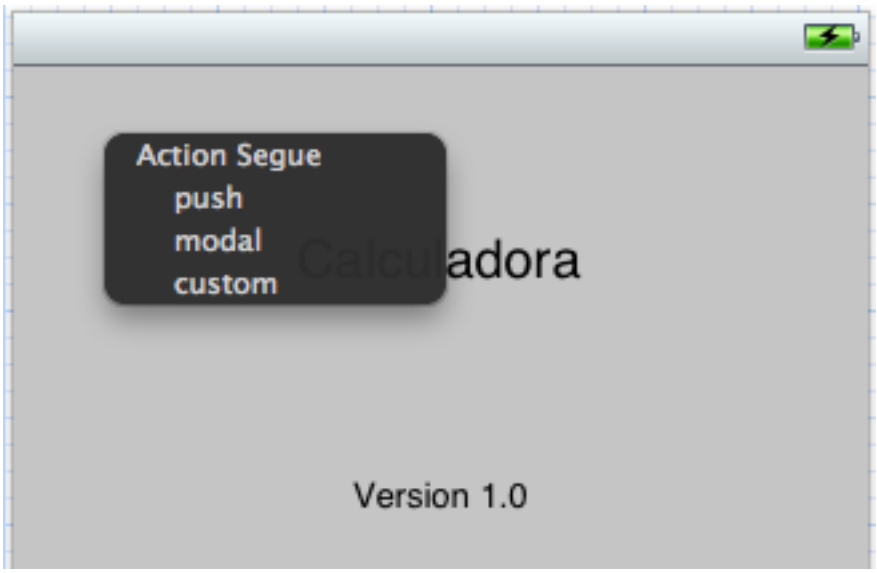


Figura 100. Creando una transición (paso 2)

Ya tenemos la transición creada, podemos ver que ha aparecido un enlace (también conocido como “segue”) entre las dos vistas:

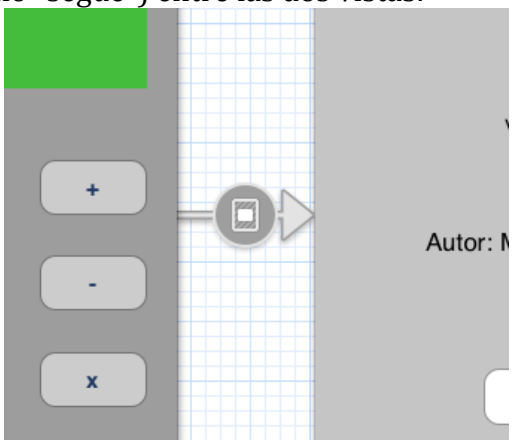


Figura 101. Segue entre dos vistas

Realizamos la misma operación para el botón “Volver” agregado en la vista de información, y lo arrastramos hasta la vista de la calculadora. En el menú volvemos a seleccionar la opción “Modal”.

Con esto damos por finalizado el ejemplo de diseño de una aplicación sencilla para iPhone.

A continuación deposito el código por si el lector desea probar la aplicación.

El diseño del Storyboard tendría que hacerse basándose en la explicación anterior, en resumen:

- Crear una vista
- Agregar los botones correspondientes (números del 0 al 9, operación suma, resta, multiplicación y división, botón de limpiar pantalla y botón igual)
- Agregar un UILabel
- Conceder a esta vista la clase que contenga el código siguiente
- Enlazar los botones y el label añadido, con los objetos de la clase
- Simular la aplicación

Código:

ViewController.h:

```
//
//  ViewController.h
//  proyectoCalculadora
//
//  Created by Miguel Sánchez del Águila on 30/08/12.
//  Copyright (c) 2012 Miguel Sánchez del Águila. All rights reserved.
//

#import <UIKit/UIKit.h>
```

```
@interface ViewController : UIViewController

@property (nonatomic,strong) IBOutlet UIButton
*but0,*but1,*but2,*but3,*but4,*but5,*but6,*but7,*but8;
@property (nonatomic,strong) IBOutlet UIButton
*but9,*butC,*butIgual,*butMas,*butMenos,*butMult,*butDiv;

@property (nonatomic,strong) IBOutlet UILabel *caja;

@property (nonatomic,strong) NSString *cadena;
@property (nonatomic) int entero;


- (IBAction)marcarNumero:(id)sender;
- (IBAction)limpiarPantalla:(id)sender;
- (IBAction)operacion:(id)sender;
- (IBAction)igual:(id)sender;

@end
```

ViewController.m:

```
//
//  ViewController.m
//  proyectoCalculadora
//
//  Created by Miguel Sánchez del Águila on 30/08/12.
//  Copyright (c) 2012 Miguel Sánchez del Águila. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

@synthesize but0,but1,but2,but3,but4,but5,but6,but7,but8,but9,butC,butDiv;
@synthesize butIgual,butMas,butMenos,butMult,caja,cadena,entero;

- (void)viewDidLoad
{
    [super viewDidLoad];
        // Do any additional setup after loading the view, typically from a nib.
}
- (void)viewDidUnload
{
    [super viewDidUnload];
```

```
    // Release any retained subviews of the main view.
}
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrient
ation
{
    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}
- (IBAction)marcarNumero:(id)sender{
    UIButton *aux = sender;
    NSString *str = caja.text;
    NSString *send = aux.titleLabel.text;
    NSString *resp = [[NSString alloc]initWithFormat:@""];
    if([str isEqualToString:@"0"]) caja.text = aux.titleLabel.text;
    else {
        resp = [NSString stringWithFormat:@"%@%@",str,send];
        caja.text = resp;
    }
}
- (IBAction)limpiarPantalla:(id)sender{
    caja.text = @"0";
}

- (IBAction)operacion:(id)sender{
    UIButton *au = sender;
    cadena = au.titleLabel.text;
    entero = [caja.text integerValue];
    [self limpiarPantalla:butIgual];
}

- (IBAction)igual:(id)sender{
    int op2 = [caja.text integerValue];
    if ([cadena isEqualToString:@"+"]){
        caja.text = [NSString stringWithFormat:@"%d",entero+op2];
    }else if ([cadena isEqualToString:@"-"]){
        caja.text = [NSString stringWithFormat:@"%d",entero-op2];
    }else if ([cadena isEqualToString:@"x"]){
        caja.text = [NSString stringWithFormat:@"%d",entero*op2];
    }else if ([cadena isEqualToString:@"/"]){
        caja.text = [NSString stringWithFormat:@"%f",(float)entero/op2];
    }
}

@end
```

Lenguajes utilizados en el proyecto

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

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 102. Ranking lenguajes de programación 2011. Extraída de "http://www.genbetadev.com/"

Los lenguajes utilizados en este proyecto no están mal situados, se aporta un poco más de información sobre ellos:

7.2 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

Véase también: 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

7.3 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

7.4 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

7.5 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, 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.

- 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 HTML 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.

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 HTML o XML, aunque también puede ser una imagen o cualquier otro contenido.

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.

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

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.

¿Estado actual del mercado, a nivel de APIs?

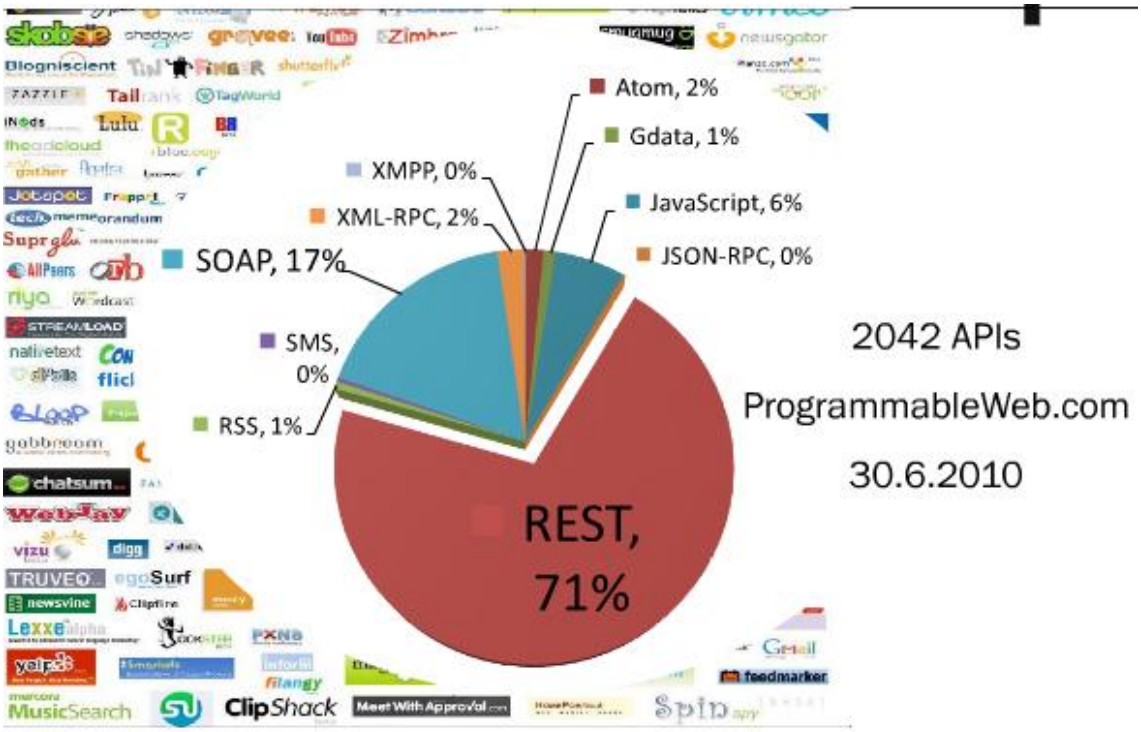


Figura 103. Utilización REST frente a SOAP

[5]Página web: http://www.estebanetayo.es/2011/11/10/webservices-soap-vs-rest-%C2%BFcual-usar-en-cada-caso/