Máster Universitario en Investigación en
Ingeniería de Software y Sistemas Informáticos

Definición de un DSL para la gestión de sensores en dispositivos móviles

Trabajo Fin de Máster
Itinerario de Ingeniería de Software (Código: 31105128)

AUTOR: **Saioa Picado Fernández**

DIRECTOR: **Ismael Abad Cardiel**

Curso Académico 2014/2015

Convocatoria Septiembre

Máster Universitario en Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO: Ingeniería del Software

CÓDIGO DE ASIGNATURA: 31105128

TÍTULO DE TRABAJO: Definición de un DSL para la gestión de sensores en dispositivos móviles

TIPO DE TRABAJO: Tipo A, proyecto específico propuesto por un profesor

AUTOR: Saioa Picado Fernández

DIRECTOR: Ismael Abad Cardiel

HOJA DE CALIFICACIONES

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y el prototipo desarrollado.

Firma del autor

Resumen

El auge de los dispositivos móviles ha permitido que surjan grandes oportunidades de negocio en torno al uso y desarrollo de aplicaciones para estos dispositivos. Esta explosión ha traído consigo una gran diversificación de dispositivos así como de sistemas operativos empleados, dificultando la creación de aplicaciones multiplataforma.

En este trabajo se realiza una definición parcial de un lenguaje específico de dominio (DSL) que permita trabajar con algunos de los sensores de los dispositivos móviles de una forma más sencilla. Además este DSL, junto con un generador, permite generar el código necesario para algunas de las plataformas móviles con mayor cuota de mercado: Android e iOS.

Antes de entrar en los detalles de la gramática, la implementación y ejemplos de uso, se hará un estudio del estado del arte evaluando algunas soluciones existentes para la generación de código multiplataforma, algunas haciendo uso de DSLs y otras de lenguajes de uso general.

Palabras clave: DSL, Lenguaje específico de dominio, multiplataforma, móvil, generación automática de código, Groovy

Abstract

The rise in the use of mobile devices has allowed the appearance of new business opportunities around the use and development of applications for these devices. This explosion in the use of mobile devices has come with a huge diversification of devices and their operating systems, making the development of multiplatform application more difficult.

In this work a partial definition of a Domain Specific Language (DSL) is made to allow an easier work with the different sensors that mobile devices offer. In addition, this DSL, with the accompanying generator, allows generating the code that is needed for the major mobile platforms: Android and iOS.

Before introducing the DSL grammar, the implementation and some use cases, an evaluation of some of the existing solutions for multiplatform code generation is made, some of them using their own DSL and others with different General Purpose Languages (GPL).

Keywords: DSL, Domain Specific Language, multiplatform, mobile, automatic code generation, Groovy

Contenido

Lista de figuras

Lista de tablas

1 INTRODUCCIÓN

En unos años el uso de dispositivos móviles se ha hecho muy común y accesible a prácticamente todo el mundo. Esta extensión del mercado ha provocado también la aparición de diferentes sistemas operativos que den soporte a estos dispositivos móviles, cada uno con sus propias características, peculiaridades, requisitos y lenguajes.

Sin embargo, al incrementarse la diversidad de sistemas operativos que se ejecutan en los dispositivos, el desarrollo de las aplicaciones móviles para los mismos implica mayor carga de trabajo o en algunos casos que se limiten a uno o un subconjunto de dichos sistemas.

Para paliar este problema han surgido en el mercado diversas opciones para la generación de aplicaciones para los distintos sistemas operativos. Estas soluciones pueden dividirse entre aquellas que generan aplicaciones híbridas y aquellas que generan aplicaciones en código nativo. Las ventajas entre estos dos tipos de aplicaciones se analizan más adelante.

A lo largo de este proyecto se analizarán diversas de estas alternativas existentes para la generación de aplicaciones para dispositivos móviles, prestando especial interés a cómo realizan el tratamiento con los distintos sensores que estos utilizan. También se estudiará la forma en que se ha de trabajar con dichos sensores en los distintos sistemas operativos.

El objetivo ideal a conseguir con la realización de este proyecto es la creación de un lenguaje específico de dominio o DSL (Domain Specific Language) que facilite el trabajo con los sensores de los dispositivos para varios sistemas operativos.

2 ÁMBITO DEL PROYECTO Y OBJETIVOS

Un primer paso a realizar será realizar un análisis y estudio del estado del arte y cómo afrontan el problema las diferentes soluciones existentes en el mercado. Se tendrán en especial atención aquellas soluciones que generen código nativo frente aquellas que generan aplicaciones híbridas aunque no se descartarán por su gran extensión en el mercado.

Este análisis se focalizará en el tratamiento de las herramientas respecto a la gestión de los sensores de los dispositivos móviles al ser el punto central del proyecto presentado en esta memoria.

Un siguiente paso consistirá en el análisis de los métodos proporcionados por las distintas APIs para realizar dicha gestión de sensores. Será necesario tener en cuenta posibles peculiaridades de cada sistema operativo tales como requisitos de seguridad, comprobaciones extra o imposibilidad de acceso a los mismos.

Partiendo de estos estudios y análisis, se realizará una propuesta de DSL junto con una breve justificación del lenguaje que se decida utilizar para su desarrollo, explicando también la generación de código que realizará dicho DSL para las distintas plataformas.

Finalmente se incluirán una serie de conclusiones obtenidas durante la realización del proyecto y unas posibles líneas de trabajo futuras para la ampliación y mejora del proyecto.

2.1 LIMITACIÓN DE ÁMBITO

Debido a la existencia de múltiples plataformas para los dispositivos móviles, no se abordará en este proyecto la generación de código para todos ellos sino que se limitará a dos de los sistemas operativos con más uso actualmente: Android e iOS.

Igualmente, al tratarse de un prototipo de DSL, se trabajará con un número limitado de sensores y funcionalidades que se consideran de mayor utilidad o cuyo uso está más extendido.

No entra dentro del alcance de este proyecto la creación de plugins o add-on para diferentes entornos de desarrollo o IDEs ni la creación de una herramienta propia que facilite el uso del DSL propuesto.

3 ESTADO DEL ARTE

En el primer apartado se analizan algunas aplicaciones de generación de código de aplicaciones en dispositivos móviles haciendo énfasis en el tratamiento que realizar para la gestión de los sensores en dichos dispositivos.

En un segundo apartado se recoge un resumen de las características más comunes e interesantes empleadas por los productos analizados en el primer apartado.

En un tercer apartado se analiza el uso de los diferentes sensores disponibles en los dispositivos móviles directamente mediante las APIs proporcionadas por los diferentes sistemas operativos. En concreto el trabajo se centra en los sistemas Android e iOS.

3.1 ANÁLISIS DE PRODUCTOS EXISTENTES

3.1.1 Rhodes

Ilustración 1 - Logo RhoMobile Suite

Rhodes es una librería API que forma parte de la suite RhoMobile. Esta librería puede utilizarse en cualquier aplicación creada con RhoMobile, siendo éste un contenedor de aplicaciones que se ejecuta como una aplicación nativa real pero que permite escribir código multiplataforma de forma sencilla utilizando lenguajes web.

Rhodes proporciona a las aplicaciones acceso a las funcionalidades de los dispositivos, como la cámara o la geolocalización, mediante interfaces JavaScript y Ruby, facilitando el desarrollo como si de una aplicación web se tratase.

Realmente lo que Rhodes crea es una aplicación nativa en la que se incrusta un navegador o WebView, siendo ahí donde se ejecuta la aplicación web desarrollada bien con Rhodes o con el resto de elementos del suite RhoMobile.

Rhodes es de uso gratuito y Open-Source, lo que permite a los desarrolladores contribuir al proyecto de forma continuada.

3.1.2 LiveCode

La plataforma LiveCode permite crear aplicaciones de forma sencilla mediante la combinación de comandos simples con componentes visuales que facilitan el diseño de la aplicación. Permite la creación de aplicaciones para Android e iOS.

Ilustración 2 - Logo LiveCode

Las aplicaciones generadas mediante LiveCode son aplicaciones nativas y no incluyen un WebView en el que se incrusta el HTML y JavaScript creado por los desarrolladores. Esta creación de aplicaciones nativas se realiza mediante el uso de una serie de comandos o DSL propio, que permite indicar el comportamiento de los componentes de una forma más legible.

```
on mouseUp
      set the text of field "messagelabel" to the text of field "newmessage"
end mouseUp
```

Con el ejemplo anterior se asocia una acción, por ejemplo a un botón, que al ser pulsado introduce en el campo *newmessage* el texto introducido en el control con identificador *messagelabel*.

LiveCode tiene una versión gratuita para uso particular pero que no permite la publicación en la App Store. Para poder desarrollar en ambas plataformas sería necesario acceder a la versión indie o a la de empresa, opciones ambas de pago.

3.1.3 Tabris.js

Ilustración 3 - Logo Tabris.js

Tabris es un framework para móviles que permite desarrollar aplicaciones nativas para iOS y Android partiendo de un solo código escrito por completo en JavaScript.

Al contrario que otros frameworks, Tabris no incrusta un WebView en las aplicaciones para poder mostrar y ejecutar el código JavaScript de la aplicación, sino que genera widgets en código nativo mediante la transformación del código JavaScript en código nativo (JavaScript to native bridges).

Un ejemplo de construcción del típico *Hello World!* sería el mostrado a continuación. Como se puede observar dispone de un API que puede verse como un DSL que es transformado en código nativo en el momento de la compilación.

```
var page = tabris.create("Page", {
  title: "Hello, World!",
  topLevel: true
});

var button = tabris.create("Button", {
  text: "Native Widgets",
  layoutData: {centerX: 0, top: 100}
}).appendTo(page);

var textView = tabris.create("TextView", {
  font: "24px",
  layoutData: {
    centerX: 0, top: [button, 50]
  }
}).appendTo(page);

button.on("selection", function() {
  textView.set("text", "Totally Rock!");
});

page.open();
```

Por defecto proporciona acceso al manejo de datos, cámara y otra serie de APIs nativas. Aunque no soporta de forma nativa el uso de todas las APIs de acceso a los elementos de los dispositivos móviles, se

puede lograr acceder a estas APIs nativas mediante plugins de Cordova. De hecho puede emplearse cualquier plugin de Cordova o incluso librerías JavaScript, siempre y cuando no trabajen con el DOM del documento, ya que no existirá código HTML que sea cargado en un WebView.

Tabris no es un producto Open Source pero sí ofrece una versión gratuita que permite realizar la compilación a través de su servicio siempre y cuando se trate de repositorios públicos. Si se quieren utilizar repositorios privados o realizar las compilaciones en local, será necesario acceder a una de las versiones de pago existentes.

3.1.4 APPlause

Applause es una solución Open Source creada inicialmente por Heiko Behrens y Peter Friese, disponiendo del código fuente en Github para realizar cambios y aportaciones según las necesidades que se detecten en las aplicaciones. Sin embargo, parece que se encuentra casi abandonado ya que la última actualización de código es de hace 2 años. La solución Applitude surgida a partir de esta solución parece igualmente abandonada.

Ilustración 4 - Logo APPlause

Se trata de un DSL que permite describir las aplicaciones móviles que servirá para, haciendo uso de una serie de herramientas y generadores, obtener las aplicaciones nativas en algunas plataformas móviles como Android, iOS o Windows Phone. Está basado en Xtext para la creación del DSL y en eXpand para la generación del código, siendo ambas soluciones de Eclipse.

El DSL proporcionado por Applause no sólo nos permite definir la vista sino que se pueden indicar los conectores con los datos y las entidades que representarán las estructuras de datos de los elementos a mostrar. Además permite definir la navegación completa dentro de la aplicación así como enlaces externos.

3.1.5 Titanium SDK

Dentro del suite de AppCelerator se encuentra Titanium SDK. Este SDK permite generar aplicaciones nativas partiendo de código JavaScript y XML, que deberá seguir un esquema (XMLSchema) concreto.

Internamente realiza un proceso parecido a Trabris.js transformando el código JavaScript creado en código nativo para cada una de las plataformas soportadas. También hace uso del XML específico para la creación de las vistas.

Un simple ejemplo de *Hello World!* sería el siguiente. En primer lugar se definiría el XML de la vista:

```
<Alloy>
	<Window class="container">
		<Label id="label" onClick="doClick">Hello, World</Label>
	</Window>
</Alloy>
```

Siendo el controlador el presentado a continuación, que mostrará un mensaje de alerta con el contenido del *Label* con identificador *label*.

```
function doClick(e) {
    alert($.label.text);
}

$.index.open();
```

Se trata de una solución propietaria y de pago pero que ofrece una versión gratuita hasta el momento en que sea necesario publicar las aplicaciones.

3.1.6 Resumen de las soluciones existentes

Solución	**Aplicaciones híbridas/nativas**	**Uso de DSL**	**Plataformas**	**Open Source / propietaria**
Rhodes	híbrida	No (JavaScript, Ruby)	Android, iOS y Windows	Propietaria
LiveCode	nativas	Sí	Android e iOS	Propietaria
Tabris.js	nativas	No (JavaScript)	Android e iOS	Propietaria (versión gratuita)
APPlause	nativas	Sí	Android, iOS y Windows Phone	Open Source
Titanium SDK	nativas	No (JavaScript, XML)	Android e iOS	Propietaria

Tabla 1 - Comparativa de las soluciones existentes

Entre las soluciones evaluadas se puede observar que hay opciones que generan auténtico código nativo y otras que incrustan un navegador en las aplicaciones y se utiliza HTML y JavaScript para la visualización y el funcionamiento.

Dentro del estudio de las soluciones existentes, se han encontrado más soluciones basadas en WebView que aquellas que generan código nativo partiendo de un DSL o sistema similar, aunque aquí se han tratado de plasmar las más relevantes y aquellas que se aproximan más al objetivo de este trabajo.

Es curioso observar cómo las soluciones basadas en la creación de aplicaciones realmente nativas han perdido protagonismo (a excepción quizás de Livecode) frente a las basadas en el uso de HTML y JavaScript dentro de un navegador incrustado en las aplicaciones, aun presentando actualmente un rendimiento y una experiencia de usuario pero que las aplicaciones nativas.

Una posible explicación a este hecho es la inferior curva de entrada necesaria para estas soluciones frente a las basadas en DSLs propios, al requerir que los desarrolladores aprendan este nuevo lenguaje frente a la amplia extensión de HTML y JavaScript.

Sin embargo esta dificultad extra de entrada se traduce en mejores resultados en especial en cuanto a rendimiento, integración e interfaces de usuario que siguen mejor las líneas indicadas para cada uno de los sistemas operativos.

3.2 PLATAFORMAS DE SMARTPHONE

En la actualidad existen multitud de plataformas para dispositivos móviles, sin embargo por su mayor rango de mercado y popularidad, sólo Android e iOS se han tomado en consideración para la elaboración de este prototipo.

En los siguientes apartados se incluyen algunos datos generales de la plataforma y otros más relacionados con la implementación en cada una de las plataformas mencionadas.

3.2.1 Android

Las aplicaciones Android en general se desarrollan utilizando el lenguaje Java. Sin embargo su ciclo de vida difiere en gran medida de las aplicaciones típicas de escritorio escritas en este lenguaje.

Una actividad Android puede encontrarse en cuatro estados diferentes:

- **Activa (Running):** la actividad está visible y en uso, en primer plano.
- **Pausada (Paused):** La actividad es visible pero no tiene el foco. Por ejemplo si se superpone alguna actividad con zona transparente o que ocupa parcialmente la pantalla.
- **Parada (Stopped):** La actividad no está visible pero no ha sido destruida ni reclamada por el manejador de memoria de Android
- **Destruida (Destroyed):** La actividad no se encuentra en ejecución.

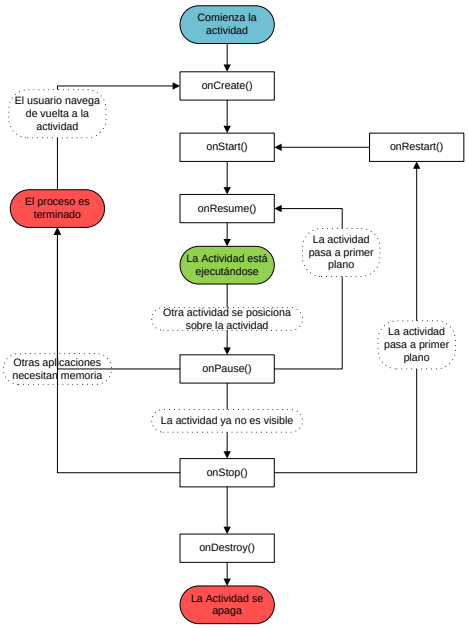

Ilustración 5 - Ciclo de vida en Android

Cada vez que se realiza un cambio de estado, se cambiará de posición en el ciclo de vida de la actividad, ejecutándose los métodos que sean necesarios, tal como se muestra en la siguiente imagen.

Sobrescribiendo estos métodos protegidos en las actividades, se puede controlar el funcionamiento de la aplicación o actividad en cada uno de los estados, almacenando los datos necesarios o mostrando la interfaz que corresponda en cada caso.

3.2.1.1 Desarrollo

Las aplicaciones Android tienen como elemento principal las actividades, clases que heredan de *Activity*, que serán las que se encarguen de cargar las vistas, formadas por distintos componentes visuales comunes, que serán con las que interactúe el usuario. Además de las actividades existen, además de otros elementos, servicios (*Service*) que permiten la ejecución de tareas de larga duración o tareas que deben ejecutarse en segundo plano para no bloquear la aplicación.

Para desarrollar aplicaciones Android es necesario el uso del SDK Android. Este SDK contiene todas las herramientas necesarias para compilar, testear y depurar las aplicaciones.

En función de la versión mínima de Android que se quiera soportar, se deberá hacer uso de las librerías de retrocompatibilidad que Android ofrece: android-support. Igualmente, está disponible una librería que

permite acceder a las funcionalidades proporcionadas por Google Play Services, por ejemplo para facilitar el trabajo con el posicionamiento de los dispositivos.

3.2.2 iOS

Las aplicaciones nativas para dispositivos con iOS se desarrollan empleando el lenguaje Objective-C, un lenguaje de programación de alto nivel, orientado a objetos, altamente influenciado por C.

Uno de los elementos principales son los controladores, siendo el más importante *UIViewController,* que maneja una vista o conjunto de ellas, haciéndose cargo de la interacción del usuario con ellas.

La entrada de una aplicación iOS se especifica implementando el protocolo *UIApplicationDelegate*, siendo en uno de sus métodos donde se crea la interfaz de usuario.

Según el estado del ciclo de vida en que se encuentre el *UIViewController* se ejecutarán una serie de métodos que permitirán controlar el estado de la aplicación. En el siguiente diagrama se muestra el ciclo de vida de un *UIViewController*.

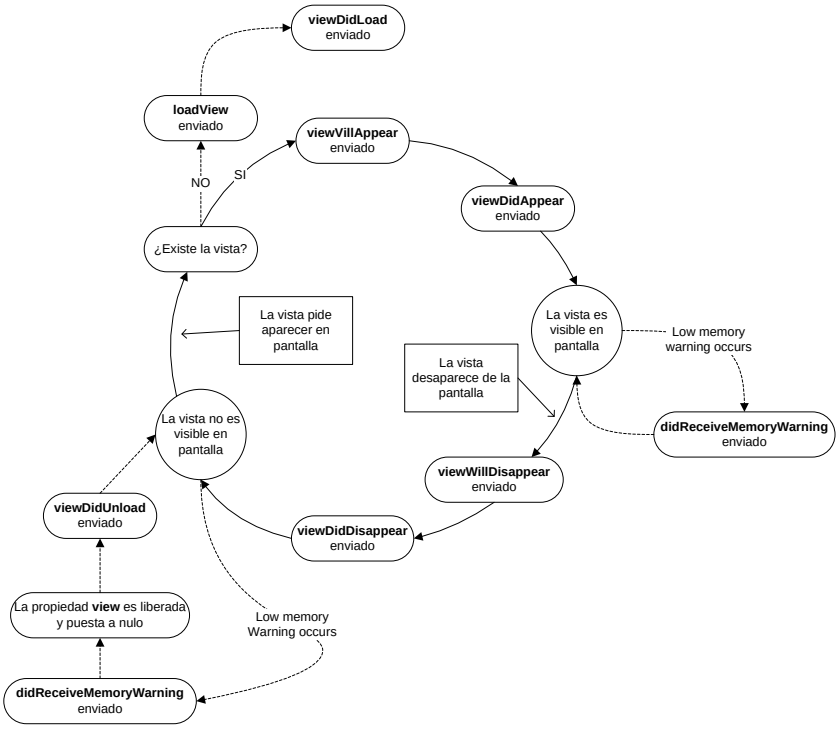

Ilustración 6 - Ciclo de vida de UIViewController

Posteriormente apareció Swift, como alternativa para desarrollar aplicaciones nativas para iOS de forma más interactiva. Su uso no sustituye necesariamente a Objective-C, sino que puede usarse como complemento a él o incluirse en aplicaciones ya desarrolladas con este lenguaje. Sin embargo su adopción de momento no es muy elevada al considerarse que no se encuentra suficientemente desarrollada.

3.2.2.1 Desarrollo

Para desarrollar aplicaciones en iOS empleando el lenguaje Objective-C es necesario emplear el SDK de iOS. Este SDK está compuesto por diferentes frameworks, siendo uno de los más importante Cocoa Touch. Además proporciona un emulador de dispositivos iPhone y iPad.

El IDE más utilizado para el desarrollo en iOS es Xcode, que incluye un compilador gcc, soporte para depuración, constructor de interfaces, etc.

3.3 Elementos de las APIs en Android

En Android hay que tener en cuenta que para poder hacer uso de varios de los sensores será necesario incluir la solicitud de permisos en el archivo manifest de la aplicación, además de emplear las APIs que se requieran en cada caso. En los siguientes apartados se indicará tanto el uso de las APIs como los cambios que se deban realizar en dicho archivo.

3.3.1 Cámara

3.3.1.1 Manifest

En primer lugar, la aplicación ha de solicitar permisos para utilizar la cámara del dispositivo:

```
<uses-permission android:name="android.permission.CAMERA" />
```

También se debe indicar en este archivo que la aplicación utiliza las funcionalidades de la cámara, pudiendo indicar si es requisito obligatorio o no mediante el flag a*ndroid:required*:

```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

En función del uso que se vaya a dar a la cámara será necesario especificar una serie de entradas adicionales:

- **Permiso de almacenamiento:** en caso de que la aplicación almacene imágenes o vídeos en la tarjeta externa de memoria (tarjeta SD) habrá que solicitar permiso para acceder a ella:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- **Permiso para grabar audio:** para capturar audio al realizar una grabación de vídeo, se ha de solicitar permiso especial:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- **Permiso de localización:** si la aplicación etiqueta las imágenes con información de la localización GSP, también ha de solicitarse permiso:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

3.3.1.2 Control de la cámara

Existen dos alternativas para utilizar la cámara: hacer uso de alguna aplicación ya existente e instalada en el dispositivo móvil o bien crear una aplicación propia de cámara. Por simplicidad para este prototipo se hará uso de alguna aplicación ya instalada en los dispositivos.

Esta llamada a otras aplicaciones se realizará mediante el uso de *Intents* y la captura posterior de los resultados devueltos por la aplicación empleada. Los pasos a seguir para realizar este proceso son:

1. *Crear el Intent de la cámara:* pudiendo ser de captura de imagen o de vídeo
2. *Arrancar dicho Intent* pidiendo que devuelva el control con el resultado
3. *Recibir y tratar el resultado*

3.3.1.2.1 Creación de Intent para captura de imagen

Un intent de captura de imágenes puede incluir la siguiente información extra:

- **MediaStore.EXTRA_OUTPUT –** Este parámetro requiere como valor un objeto *Uri* que especifique la ruta y el nombre donde se desea guardar la imagen. No es obligatorio la inclusión de esta información pero sí altamente recomendada ya que si no la aplicación almacenará la imagen en la ruta por defecto con un nombre por defecto, que se indicará en el campo de retorno del Intent obtenible mediante *Intent.getData()*.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private Uri fileUri;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // crear el Intent para capturar la imagen y devolver el control a la aplicación
principal
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    // getOutputMediaFileUri creará el archive donde almacenar la imagen y devolverá la
Uri correspondiente
    fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // indicar la información extra

    // arrancar el Intent de captura de imagen
    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
```

3.3.1.2.2 Creación de Intent para captura de vídeo

A un Intent de captura de vídeo se le puede indicar la siguiente información extra:

- **MediaStore.EXTRA_OUTPUT –** se indicará mediante un objeto *Uri* la ruta y el nombre de archivo donde almacenar el vídeo. Esta información es opcional pero se recomienda ya que si no el vídeo se almacenará en una ruta y con un nombre por defecto, que se podrá obtener mediante la llamada *Intent.getData()*.
- **MediaStore.EXTRA_VIDEO_QUALITY –** siendo 0 la menor calidad y el menor tamaño de archivo y 1 para la mejor y archivo más grande
- **MediaStore.EXTRA_DURATION_LIMIT –** se puede indicar un valor en segundos para limitar la duración del vídeo capturado
- **MediaStore.EXTRA_SIZE_LIMIT –** se puede indicar un valor en bytes para limitar el tamaño del archivo de vídeo

```
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
private Uri fileUri;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Crear el intent de captura de vídeo
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

    fileUri = getOutputMediaFileUri(MEDIA_TYPE_VIDEO);  // crear el archive para guardar
el vídeo
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // indicar la información del
nombre de archivo

    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1); // establecer la calidad de
imagen a alta

    // arrancar el intent de captura de vídeo
    startActivityForResult(intent, CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE);
}
```

3.3.1.2.3 Recepción y tratamiento de resultado

Para obtener los resultados de un Intent, será necesario sobrescribir el método *onActivityResult()* en el *Activity* que realizó la llamada al Intent. En el siguiente trozo de código se indica cómo capturar el resultado tanto del intent de foto como de vídeo.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
private static final int CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Imagen capturada y almacenada en el fileUri indicado en el Intent
            Toast.makeText(this, "Image saved to:\n" +
```

```
                    data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // El usuario ha cancelado la captura de imagen
        } else {
            // La captura de imagen falló, avisar al usuario
        }
    }

    if (requestCode == CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Vídeo capturado y almacenado en el fileUri indicado en el Intent
            Toast.makeText(this, "Video saved to:\n" +
                    data.getData(), Toast.LENGTH_LONG).show();
        } else if (resultCode == RESULT_CANCELED) {
            // El usuario cancel la captura de vídeo
        } else {
            // La captura de vídeo falló, avisar al usuario
        }
    }
}
```

Una vez que el *Activity* de la aplicación recibe un resultado satisfactorio, la imagen o vídeo capturado está disponible en la localización que se haya indicado.

3.3.2 Acelerómetro

El sensor de acelerómetro mide la aceleración aplicada al dispositivo, incluyendo la fuerza de la gravedad.

En primer lugar se deberá obtener una instancia del sensor de acelerómetro por defecto empleando el siguiente código:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Es importante que la clase donde se quieran escuchar los cambios en el sensor de aceleración implemente la interfaz *SensorEventListener* así como realizar la subscripción del observador y la cancelación de la subscripción. Por ejemplo en un Activity se recomienda registrarlo en el método *onResume()* y pararlo en el método *onPause()*.

```
public class SensorsTester implements SensorEventListener {
      @Override
      public void onResume() {
               super.onResume();
               if (mAccelerometerSensor != null) {
                        mSensorManager.registerListener(this, mAccelerometerSensor,
SensorManager.SENSOR_DELAY_NORMAL);
               }
      }
      @Override
```

```
        public void onPause(){
                super.onPause();
                if (mSensorManager != null) {
                        mSensorManager.unregisterListener(this);
                }
        }
}
```

Con el siguiente código se muestra cómo obtener los valores de la aceleración sobre los tres ejes, aplicando tanto un filtro de paso alto como uno de paso bajo para aislar la aceleración real eliminando la fuerza de la gravedad u obtener únicamente la fuerza de la gravedad, respectivamente.

```
@Override
public void onSensorChanged(SensorEvent event){
  // In this example, alpha is calculated as t / (t + dT),
  // where t is the low-pass filter's time-constant and
  // dT is the event delivery rate.

  final float alpha = 0.8;

  // Isolate the force of gravity with the low-pass filter.
  gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
  gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
  gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

  // Remove the gravity contribution with the high-pass filter.
  linear_acceleration[0] = event.values[0] - gravity[0];
  linear_acceleration[1] = event.values[1] - gravity[1];
  linear_acceleration[2] = event.values[2] - gravity[2];
}
```

3.3.3 Giroscopio

El giroscopio mide la rotación en radianes alrededor de los ejes x, y, z del dispositivo. A continuación se muestra cómo obtener una instancia del giroscopio por defecto:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

Como en el caso del acelerómetro, es necesario que donde se quieran escuchar los cambios en el sensor se implemente la interfaz *SensorEventListener* así como realizar la subscripción del observador y la cancelación de la subscripción.

```
public class SensorsTester implements SensorEventListener {
	@Override
	public void onResume() {
		super.onResume();
		if (mGyroscope != null) {
			mSensorManager.registerListener(this, mGyroscope,
SensorManager.SENSOR_DELAY_NORMAL);
		}
	}
	@Override
	public void onPause(){
		super.onPause();
		if (mSensorManager != null) {
			mSensorManager.unregisterListener(this);
		}
	}
}
```

Para obtener los valores se podría utilizar el siguiente código:

```
public void onSensorChanged(SensorEvent event) {
  // Axis of the rotation sample, not normalized
  float axisX = event.values[0];
  float axisY = event.values[1];
  float axisZ = event.values[2];
}
```

El giroscopio estándar proporciona datos de rotación en crudo sin aplicar ningún filtro o corrección de ruido o bias. En la práctica, esto producirá errores que necesitan ser compensados.

3.3.4 Localización

A partir de la versión 2.2 (API 8) de Android, está disponible a través de Google Play Services un servicio de localización que facilita la detección y escucha de cambios de localización de las aplicaciones.

A continuación se muestran ejemplos de cómo obtener la última posición conocida y subscribirse a cambios en la localización del dispositivo.

3.3.4.1 Permisos

En primer lugar será necesario incluir en el archivo *Manifest* de la aplicación la entrada de solicitud de permisos necesaria para habilitar la localización.

Android proporciona dos permisos de localización: ACCESS_COARSE_LOCATION y ACCESS_FINE_LOCATION. El valor que se indique determinará la precisión con que el API dará la localización. Con el valor ACCESS_COARSE_LOCATION, proporciona una localización con una precisión aproximada equivalente a un bloque de edificios. En función de la precisión que requiera la aplicación se

deberá indicar uno u otro valor, teniendo en cuenta que una mayor precisión irá acompañada de un mayor gasto de batería del dispositivo.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

3.3.4.2 Conectarse a Google Play Services

Para poder hacer uso del API de localización es necesario obtener una instancia del cliente de Google Play Services. Se puede utilizar el siguiente método desde el *onCreate* para obtener la instancia.

```
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
}
```

3.3.4.3 Última localización conocida

Una vez se haya conectado a Google Play Services y el servicio de localización, se puede obtener la última localización conocida del usuario mediante el método *getLastLocation(),* obteniendo un objeto *Location* del que se pueden obtener las coordenadas de latitud y longitud. La precisión de la localización dependerá de la que se haya indicado en el manifest de la aplicación.

La llamada a este método deberá hacerse en el callback *onConnected()* proporcionado por el cliente de API de Google, que es llamado cuando el cliente está listo.

```
public class MainActivity extends ActionBarActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    @Override
    public void onConnected(Bundle connectionHint) {
        mLastLocation = LocationServices.FusedLocationApi.getLastLocation(
                mGoogleApiClient);
        if (mLastLocation != null) {
            mLatitudeText.setText(String.valueOf(mLastLocation.getLatitude()));
            mLongitudeText.setText(String.valueOf(mLastLocation.getLongitude()));
        }
    }
}
```

3.3.4.4 Recibir cambios en la localización

3.3.4.4.1 Construir la petición

Se debe crear un *LocationRequest* que almacene los parámetros que se quieren enviar en la petición al proveedor de localización, que servirán para determinar los niveles de precisión solicitados. Algunas de las propiedades que se pueden establecer son las siguientes:

- **setInterval() –** indica cada cuantos milisegundos quiere la aplicación recibir actualizaciones en la localización. Hay que tener en cuenta que las actualizaciones pueden ser más frecuentes si hay otra aplicaciones recibiéndolas a una mayor velocidad. También puede ocurrir que no lleguen o lleguen más lentas si por ejemplo el dispositivo no tiene conectividad
- **setFastestInterval() –** indica el rango mínimo en milisegundos que la aplicación puede tratar las actualizaciones para no verse afectado por los rangos de otras aplicaciones.
- **setPriority() –** establece la prioridad de la petición, lo cual proporciona al servicio de localización una pista importante sobre qué recurso de localización utilizar. Se pueden indicar los siguientes valores:
    - *PRIORITY_BALANCED_POWER_ACCURACY* - precisión aproximada de 100 metros. Lo más probable es que se utilice las redes WIFI y las antenas de telefonía para posicionar
    - *PRIORITY_HIGH_ACCURACY* – solicita la localización más precisa posible. Es más probable que se utilice el GPS para el posicionamiento
    - *PRIORITY_LOW_POWER* – solicita precisión a nivel de ciudad, lo que se traduce en aproximadamente 10 km. Se supone que provocará un menor gasto de batería
    - *PRIORITY_NO_POWER* – se debe utilizar este valor si no se quiere tener impacto en la batería pero se quiere tener actualizaciones de la localización cuando esté disponible. En este modo las aplicaciones no provocan actualizaciones de localización pero sí reciben las que otras aplicaciones puedan originar.

Un ejemplo de creación de una petición sería la siguiente:

```
protected void createLocationRequest() {
    LocationRequest mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(10000);
    mLocationRequest.setFastestInterval(5000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

3.3.4.4.2 Solicitar actualizaciones de localización

El siguiente paso es solicitar las actualizaciones de la localización, dentro del método *onConnected()* proporcionado por el cliente del API de Google.

Una primera opción es utilizar un observador (*LocationListener*), que funciona mejor en situaciones en que se requiere la localización únicamente cuando la aplicación está en primer plano.

```
@Override
public void onConnected(Bundle connectionHint) {
    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
protected void startLocationUpdates() {
    LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
}
```

Si se quieren recibir actualizaciones cuando la aplicación no esté en primer plano, se recomienda utilizar la versión en la que se emplea un *PendingIntent*. Primero se deberá registrar el receptor en el archivo Manifest de la aplicación:

```
<receiver android:name=".LocationReceiver" />
```

Realizar la petición de recibir las actualizaciones indicando el receptor en lugar del observador y crear el receptor que tratará las actualizaciones:

```
@Override
public void onConnected(Bundle bundle) {
  Intent intent = new Intent(this, LocationReceiver.class);
  PendingIntent locationIntent = PendingIntent.getBroadcast(getApplicationContext(),
INTENT_LOCATION_UPDATE, intent, PendingIntent.FLAG_CANCEL_CURRENT);
  LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, locationIntent);
  ...
}
```

3.3.4.4.3 Tratar las actualizaciones

En caso de haber indicado un observador al solicitar las actualizaciones de localización, habrá que implementar el método *onLocationChanged* de nuestro observador.

```
public class MainActivity extends ActionBarActivity implements
        ConnectionCallbacks, OnConnectionFailedListener, LocationListener {
    ...
    @Override
    public void onLocationChanged(Location location) {
        mCurrentLocation = location;
    }
}
```

Si por el contrario se ha utilizado la versión con *PendingIntent*, será necesario crear la clase que se encargue de recibir las peticiones y que ha sido registrado en el archivo Manifest de la aplicación.

```
public class LocationReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Location location = (Location)
intent.getExtras().get(LocationClient.KEY_LOCATION_CHANGED);
    }
}
```

3.3.4.4.4 Detener actualizaciones de localización

Una posibilidad para dejar de escuchar las actualizaciones en la localización es detener en el método *onPause* y reactivarlo en el método *onResume*, suponiendo que sólo requerimos actualizaciones si la aplicación está en primer plano.

```
@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

protected void stopLocationUpdates() {
    LocationServices.FusedLocationApi.removeLocationUpdates(
            mGoogleApiClient, this);
    //Esta otra version si se usan PendingIntents
    LocationServices.FusedLocationApi.removeLocationUpdates(
            mGoogleApiClient, pendingIntent);
}
@Override
public void onResume() {
    super.onResume();
    if (mGoogleApiClient.isConnected() && !mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
```

3.4 Elementos de las APIs en iOS

A diferencia de en Android, no es necesario pedir permisos a la hora de instalar las aplicaciones para utilizar ciertos sensores y capacidades. Sin embargo si se trata de funcionalidades básicas para el funcionamiento de las aplicaciones, se pueden indicar como requisitos de la aplicación en la propiedad *UIRequiredDeviceCapabilities* del archivo Info.plist. Esto provocará que no se pueda instalar la aplicación en los dispositivos que no cumplan con estos requisitos.

3.4.1 Cámara

Para el uso de la cámara hay dos opciones: utilizar las funcionalidades por defecto de captura de imágenes y vídeos o desarrollar las funcionalidades por completo, que ofrece más posibilidades de personalización. Para este piloto se usarán las funcionalidades por defecto por su mayor simplicidad.

3.4.1.1 Info.plist

Si el uso de la cámara es obligatorio para el funcionamiento correcto de la aplicación, habrá que indicarlo en la clave *UIRequiredDeviceCapabilities*. Se puede indicar tanto en forma de array como de diccionario, teniendo que indicar en el segundo caso valor *true* si tiene que existir o *false* si se requiere que no esté.

```
//This form would be for using an array
<key>UIRequiredDeviceCapabilities</key>
<array>
    <string>still-camera</string> //picture camera
    <string>video-camera</string>  //video camera
</array>
//This form would be for using a dictionary
<key>UIRequiredDeviceCapabilities</key>
<dict>
    <key>still-camera</key> //picture camera
    <true/>
    <key>video-camera</key>  //video camera
    <true/>
</array>
```

3.4.1.2 Presentar la interfaz de cámara o video

Se debe crear y configurar un controlador de captura de imágenes indicando algunas opciones:

- **Source type –** indicar como valor *UIImagePickerControllerSourceTypeCamera* para indicar que se use la captura de imagen en lugar del navegador de archivos multimedia.
- **Media types –** para indicar si el usuario puede tomar imágenes, vídeos o ambos. Se indicarán en *mediaTypes* la elección mediante un array con los valores *kUTTypeImage* o *kUTTypeMovie* respectivamente.
- **Editing controls –** permite indicar si la interfaz de la cámara debe mostrar controles para mover y escalar la imagen capturada, o para cortar el vídeo obtenido. Si se indica el valor *YES* se mostrarán los mencionados controles
- **Delegate object –** finalmente hay que asignar el objeto delegado a la propiedad *delegate* del controlador de captura de imágenes

Es importante comprobar siempre antes la disponibilidad y existencia de la cámara mediante el método *isSourceTypeAvailable* de la clase *UIImagePickerController*.

```
- (BOOL) startCameraControllerFromViewController: (UIViewController*) controller
               usingDelegate: (id <UIImagePickerControllerDelegate,
                                   UINavigationControllerDelegate>) delegate {
     if (([UIImagePickerController isSourceTypeAvailable:
                 UIImagePickerControllerSourceTypeCamera] == NO)
           || (delegate == nil)
           || (controller == nil))
        return NO;

     UIImagePickerController *cameraUI = [[UIImagePickerController alloc] init];
    cameraUI.sourceType = UIImagePickerControllerSourceTypeCamera;

    // Displays a control that allows the user to choose picture or
    // movie capture, if both are available:
```

```
    cameraUI.mediaTypes =
        [UIImagePickerController availableMediaTypesForSourceType:
            UIImagePickerControllerSourceTypeCamera];

    // Hides the controls for moving & scaling pictures, or for
    // trimming movies. To instead show the controls, use YES.
    cameraUI.allowsEditing = NO;

    cameraUI.delegate = delegate;

    [controller presentModalViewController: cameraUI animated: YES];
    return YES;
}
```

El ejemplo anterior mostrará un cuadro de diálogo para que el usuario pueda elegir entre captura de cámara o vídeo si se encuentran ambos disponibles. En caso de querer utilizar sólo el vídeo, habrá que indicar en el *mediaTypes* lo siguiente:

```
cameraUI.mediaTypes = [[NSArray alloc] initWithObjects: (NSString *) kUTTypeMovie, nil];
```

Sustituyendo *KUTTypeMovie* por *kUTTypeImage* para presentar únicamente captura de imágenes.

3.4.1.3 Recepción de resultado

Cuando el usuario pulsa sobre un botón en la interfaz de la cámara para aceptar la nueva captura de imagen o video, o cuando cancela la operación, el sistema notifica al elemento delegado u observador la elección del usuario. Sin embargo el sistema no cierra la interfaz de la cámara, siendo responsabilidad del elemento que ha realizado la llamada su cierre mediante la llamada al método *dismissModalViewControllerAnimated*.

El siguiente ejemplo muestra posibles implementaciones de métodos delegados para un controlador de captura de imágenes.

```
@implementation CameraViewController (CameraDelegateMethods)
 // For responding to the user tapping Cancel.
- (void) imagePickerControllerDidCancel: (UIImagePickerController *) picker {
    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
}
// For responding to the user accepting a newly-captured picture or movie
- (void) imagePickerController: (UIImagePickerController *) picker
            didFinishPickingMediaWithInfo: (NSDictionary *) info {
    NSString *mediaType = [info objectForKey: UIImagePickerControllerMediaType];
    UIImage *originalImage, *editedImage, *imageToSave;
    // Handle a still image capture
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeImage, 0)
            == kCFCompareEqualTo) {
        editedImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerEditedImage];
        originalImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerOriginalImage];

        if (editedImage) {
            imageToSave = editedImage;
        } else {
            imageToSave = originalImage;
        }
    // Save the new image (original or edited) to the Camera Roll
        UIImageWriteToSavedPhotosAlbum (imageToSave, nil, nil , nil);
    }
    // Handle a movie capture
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeMovie, 0)
            == kCFCompareEqualTo) {
        NSString *moviePath = [[info objectForKey:
                    UIImagePickerControllerMediaURL] path];

        if (UIVideoAtPathIsCompatibleWithSavedPhotosAlbum (moviePath)) {
            UISaveVideoAtPathToSavedPhotosAlbum (
                    moviePath, nil, nil, nil);
        }
    }
    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
}
@end
```

3.4.2 Acelerómetro y Giroscopio

El objeto *CMMotionManager* es el que permitirá acceder a los distintos servicios de movimiento proporcionados por iOS, encontrándose entre ellos el acelerómetro y el giroscopio. Por tanto un primer paso será conseguir un objeto de este tipo, que deberá ser instanciado una única vez en cada aplicación.

```
CMMotionManager *mManager = [(AppDelegate *)[[UIApplication sharedApplication] delegate]
sharedManager];
```

Para recibir actualizaciones de los valores de estos sensores de movimiento hay dos opciones: solicitando actualizaciones cada cierto intervalo de tiempo o realizando comprobaciones periódicas de estos valores.

3.4.2.1 Recibir actualizaciones en intervalos

Para recibir actualizaciones de movimiento en intervalos específicos, la aplicación llama a un método de arranque que recibe una cola de operaciones (una instancia de *NSOperationQueue*) y un bloque encargado de procesar esas actualizaciones. La frecuencia de las actualizaciones se determina por el valor de una propiedad *interval*:

- **Acelerómetro:** se trata de la propiedad *accelerometerUpdateInterval*. Se deberá llamar al método *startAccelerometerUpdatesToQueue:withHandler:* pasando un bloque de tipo *CMAccelerometerHandler*. Este bloque recibirá los datos del acelerómetro en objetos del tipo *CMAccelerometerData*.
- **Giroscopio:** se indica el valor del intervalo en la propiedad *gyroUpdateInterval*. La llamada al método de arranque será *startGyroUpdatesToQueue:withHandler* pasando un bloque de tipo *CMGyroHandler*. Los datos de velocidad de rotación de pasarán al bloque en objetos de tipo *CMGyroData*.

Para parar la recepción de actualizaciones se deberán llamar a los métodos de parada correspondientes: *stopAccelerometerUpdates* y *stopGyroUpdates*.

```
if ([self.motionManager isAccelerometerAvailable])
{
      NSOperationQueue *queue = [[NSOperationQueue alloc]init];
      [self.motionManager startAccelerometerUpdatesToQueue:queue
withHandler:^(CMAccelerometerData *accelerometerData,NSError *error)
      {
               //Treat data
      }];
}
else
{
      NSLog(@"Accelerometer is not available.");
}
if ([self.motionManager isGyrosAvailable])
{
      NSOperationQueue *queue = [[NSOperationQueue alloc]init];
```

```
      [self.motionManager startGyroUpdatesToQueue:queue withHandler:^(CMAGyroData
*gyroData,NSError *error)
      {
                //Treat data
      }];
}
else
{
       NSLog(@"Gyroscope is not available.");
}
```

3.4.2.2 Solicitud periódica de datos

Para realizar la petición de datos de forma periódica y manual, la aplicación debe llamar al método de arranque correspondiente sin pasar ningún parámetro y accede de forma periódica a los datos de movimiento almacenados en una propiedad específica para cada tipo de datos de movimiento. Esta opción es la recomendada por la documentación oficial de iOS al no introducir una sobrecarga adicional.

- **Acelerómetro:** se debe llamar al método *startAccelerometerUpdates* para arrancar el proceso. Periódicamente la aplicación accederá a los objetos *CMAccelerometerData* leyendo la propiedad *accelerometerData*. El método de parada es *stopAccelerometerUpdates*
- **Giroscopio:** el método de arranque se llama *startGyroUpdates* siendo *stopGyroUpdates* el de parada. Se podrá acceder a los objetos de tipo *CMGyroData* leyendo la propiedad *gyroData*.

```
[_motionManager startAccelerometerUpdates];
CMAcceleration acceleration = _motionManager.accelerometerData.acceleration;
[_motionManager startGyroUpdates];
CMGyroData gyroscope = _motionManager.gyroData.rotationRate
```

3.4.3 Localización

La clase *CLLocationManager* es la clase central que permite configurar y obtener eventos de localización desde las aplicaciones. Para configurar y utilizar este objeto para recibir eventos se han de seguir una serie de pasos:

1. Solicitar autorización para utilizar los servicios de localización y comprobar si el servicio está disponible
2. Crear una instancia de la clase *CLLocationManager* y almacenar una referencia a la misma en la aplicación.
3. Asignar un objeto propio a la propiedad *delegate*, el cual deberá cumplir el protocolo *CLLocationManagerDelegate*
4. Configurar otros parámetros relevantes del servicio
5. Invocar el método apropiado de arranque para empezar a recibir los eventos en el objeto delegado que se haya definido.

3.4.3.1 Info.plist

Para poder solicitar permisos al usuario, es necesario incluir en el archivo Info.plist las claves *NSLocationAlwaysUsageDescription* o *NSLocationWhenInUseUsageDescription* con el mensaje a mostrar al usuario en el dialogo de solicitud.

```
<key>NSLocationAlwaysUsageDescription</key>
<string>Mensaje para pedir permiso para siempre/string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>Mensaje para pedir permiso mientras esté arrancada la aplicación</string>
```

3.4.3.2 Solicitud de permisos para utilizar servicios de localización

Antes de utilizar los servicios de localización, la aplicación debe pedir permiso al usuario para emplear esos servicios y comprobar la disponibilidad de los servicios objetivo. Una secuencia típica sería la siguiente:

1. Llamar al método de la clase *authorizationStatus* para obtener el estado actual de autorización. En caso de que el valor sea *kCLAuthorizationStatusRestricted* o *kCLAuthorizationStatusDenied*, la aplicación no tiene permisos para utilizar los servicios de localización.
2. Crear un objeto *CLLocationManager* y asignar un observador a ella
3. Almacenar una referencia fuerte a este objeto
4. Si el estado de autorización es *kCLAuthorizationStatusNotDetermined*, se deberá llamar al método *requestWhenInUseAuthorization* o *requestAlwaysAuthorization* para solicitar el tipo correspondiente de autorización al usuario.
5. En el caso de querer utilizar el servicio de localización estándar, se deberá llamar al método *locationServicesEnabled*.

Para recibir notificaciones cuando cambia el estado de la autorización, habría que implementar el método *locationManager:didChangeAuthorizationStatus:* en el observador del manager de localización.

```
-(void)locationManager:(CLLocationManager *)manager
didChangeAuthorizationStatus:(CLAuthorizationStatus)status{
    NSLog(@"%d AUTH STATUS",status);
    if (status == kCLAuthorizationStatusAuthorizedWhenInUse) {
      [self.locationManager startUpdatingLocation];
    }
}
```

3.4.3.3 Recibir cambios en la localización

Mediante el uso del servicio de localización estándar se pueden especificar la precisión de la localización y recibir actualizaciones cuando la localización cambia.

En primer lugar se deberán indicar los valores de precisión en las propiedades *desiredAccuracy* y *distanceFilter* y a continuación invocar el método *startUpdatingLocation*. No se debe indicar un valor de precisión superior al que sea estrictamente necesario ya que el servicio de localización emplea este valor

de precisión para manejar correctamente la batería. Un valor más alto de precisión requiere del uso de hardware de mayor precisión como el GPS, que consume más potencia.

Los datos de localización serán enviados a la aplicación mediante el objeto observador que la aplicación haya asociado al gestor de localizaciones. Como puede ser necesario cierto tiempo para devolver la localización inicial, el gesto de localizaciones devuelve la última localización conocida de forma inmediata y va enviando localizaciones más actualizadas según van estando disponibles. Se recomienda por tanto comprobar la fecha que se incluye en los objetos de localización antes de realizar cualquier acción.

A continuación se muestra un ejemplo completo de uso del servicio de localización estándar.

```
#import "ViewController.h"
@import CoreLocation;

@interface ViewController () <CLLocationManagerDelegate>
@property (strong, nonatomic) CLLocationManager *locationManager;
@end

@implementation ViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.locationManager = [[CLLocationManager alloc] init];
    self.locationManager.delegate = self;

    if ([self.locationManager
respondsToSelector:@selector(requestWhenInUseAuthorization)]) {
        [self.locationManager requestWhenInUseAuthorization];
    }
    [self.locationManager startUpdatingLocation];
}

// Location Manager Delegate Methods
- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray
*)locations
{
    NSLog(@"%@", [locations lastObject]);
}
@end
```

4 PROPUESTA DE DSL

Tras analizar algunas de las diversas soluciones existentes con un planteamiento similar al deseado para este trabajo, se propone la creación de en un DSL interno generativo que creará el código final tras ejecutar una tarea que realice la generación.

Como ya se ha visto en el estudio del estado del arte, hay soluciones que se asemejan a la propuesta, como Tabris.js, LiveCode o Titanium SDK, que se encargan de generar el código de las aplicaciones móviles a partir de otros lenguajes (XML, JavaScript, HTML) mediante el uso de puentes o transformadores. A diferencia de estos, y similar a la aproximación de APPlause, en este trabajo se pretende incluir un lenguaje específico del dominio que permita que el código creado sea más legible y mantenible.

La cuestión de realizar la generación en una tarea independiente, al igual que lo realizan las soluciones evaluadas, se debe a la necesidad de no sobrecargar en exceso a los dispositivos que deben ejecutar las aplicaciones. Pero sobre todo se debe a algunas particularidades que tienen algunos de los lenguajes y máquinas virtuales que se ejecutan en estos dispositivos, como Android, que realiza una compilación especial para su máquina virtual Dalvik, la cual actualmente no permite la ejecución de algunos lenguajes externos, como Groovy.

En los siguientes apartados se indica una breve explicación de los DSLs y se explica en mayor profundidad las razones por las que se ha elegido el lenguaje Groovy como base para la construcción del DSL así como la elección de que sea bajo un enfoque generativo.

4.1 DOMAIN SPECIFIC LANGUAGE

4.1.1 ¿Qué es un DSL?

Un lenguaje específico de dominio (DSL – Domain-specific-language) es un lenguaje de programación orientado a solucionar un problema específico o dentro de un dominio concreto. Ejemplos típicos de DSL son SQL para el trabajo con base de datos o HTML para las páginas web. En contraposición se encuentran los lenguajes de propósito general, como pueden ser C o Java, que pueden emplearse para solucionar una amplia variedad de problemas.

Una definición dada por Van Deursen y compañía [1] de un DSL es:

> *Un lenguaje específico de dominio, Domain-Specific Language (DSL) es un lenguaje de programación o una especificación ejecutable de un lenguaje que ofrece, mediante las pertinentes anotaciones y abstracciones, poder de expresividad centrado en, y normalmente restringido a un problema de dominio particular.*

Las características principales de un DSL, que lo diferencian de un lenguaje de propósito general son dos:

- Está orientado a un problema o dominio concreto
- Contiene una sintaxis y una semántica que permiten modelar los conceptos al mismo nivel de abstracción que el dominio al que se refiere.

Por otro lado, es importante que un DSL sea intuitivo, legible y de fácil uso, ya que su utilización está orientada hacia usuarios no programadores, aunque no de forma exclusiva.

4.1.2 Estructura de un DSL

A continuación se describen los tres principios que un DSL bien diseñado debe cumplir para hacer el software más comunicativo para los usuarios del dominio:

- Proporciona un mapeo directo a los artefactos del problema del dominio. Si el problema del dominio contiene una entidad denominada *Cambio*, el script del DSL debe contener una abstracción con el mismo nombre que juegue el mismo papel.
- El DSL debe emplear el vocabulario común del problema del dominio, convirtiéndose en la parte central para proporcionar una mejor comunicación entre los desarrolladores y los usuarios de negocio. Cuando los usuarios del dominio emplean el software de modelado, el DSL es su interfaz.
- El script DSL debe ocultar y abstraer la implementación real, no debiendo contener complejidades accidentales que desvelen detalles de la implementación.

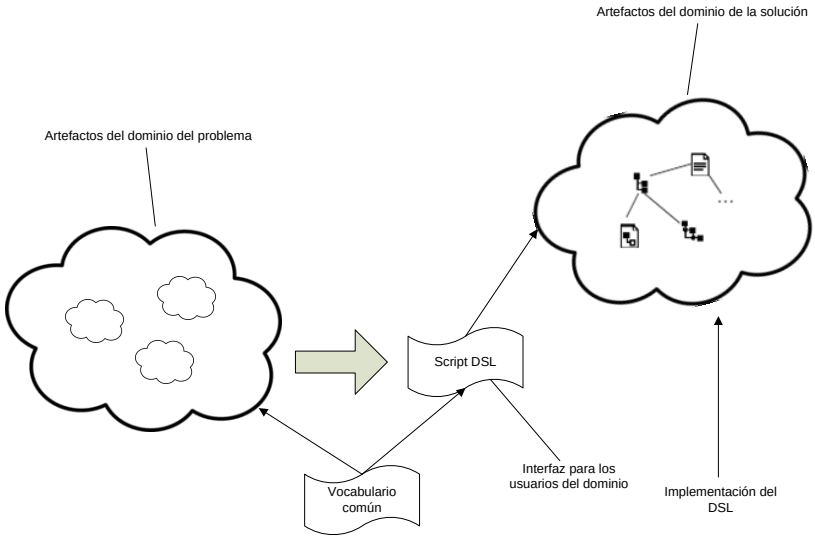

Ilustración 7 - Estructura de un DSL

En la Ilustración 7 se muestran estos tres principios mediante las relaciones entre el elemento etiquetado como Script DSL y el resto de nodos.

4.1.3 DSL interno

Fowler [2] propone una categorización de los DSL en dos grupos: internos y externos, en función de si han sido implementados sobre un lenguaje base o no.

Los DSL internos también se conocen como *embebidos o incrustados* (*embedded*) por estar implementados utilizando la infraestructura de un lenguaje de programación ya existente.

Por su lado, los DSL externos son denominados también *standalone* por estar desarrollados de principio a fin como un lenguaje independiente, sin utilizar la infraestructura de un lenguaje base. Disponen de una infraestructura independiente para el análisis léxico, técnicas de parseado, interpretación, compilación y generación de código.

En este trabajo se propone la creación de un DSL interno usando Groovy como lenguaje base, que será utilizado dentro de los lenguajes de los diferentes dispositivos móviles.

4.1.4 Enfoque generativo

Tanto el DSL como la tarea que acompaña al trabajo se encargan de generar el código necesario para pasar de una gramática general a una implementación específica para la plataforma móvil elegida como destino.

Gracias a este enfoque generativo del DSL se evita la repetición de código así como incrementar la expresividad del código y facilitar el mantenimiento del mismo. Esta separación permite además evolucionar de forma semindependiente el DSL y generador de las aplicaciones desarrolladas, pudiendo incluir o mejorar las funcionalidades sin obligar a los desarrolladores a actualizar sus aplicaciones, a menos que quieran hacer uso de dichas mejoras.

4.2 ARQUITECTURA DIRIGIDA POR MODELOS

Siguiendo las propuestas de la arquitectura dirigida por modelos, se ha planteado este trabajo tratando de formar parte de los modelos y proceso de transformación que define. El DSL junto con el generador presentado actuará como medio de transformación de un modelo independiente de la plataforma a un modelo específico de las plataformas de los diferentes dispositivos móviles.

Se puede considerar que el DSL es independiente ya que la gramática será común independiente del lenguaje de partida y la plataforma de destino, elegida únicamente en el momento de ejecución del generador.

4.2.1 ¿Qué es MDA?

La arquitectura dirigida por modelos (Model-Driven Architecture o MDA) es un acercamiento al diseño, de software propuesto por la Object Management Group (OMG). Se basa en las ideas de separación de la especificación de la operatividad de un sistema de los detalles de la forma en que el sistema utiliza las capacidades de su plataforma.

Las metas principales de MDA son la portabilidad, interoperabilidad y reutilización mediante la separación arquitectónica de conceptos. Estos objetivos pretenden conseguirse mediante la creación de herramientas que permitan:

- Realizar la especificación de un sistema de forma independiente a la plataforma de soporte
- Especificar diferentes plataformas
- Elegir una plataforma específica para el sistema
- Transformar la especificación del sistema en una plataforma particular

4.2.2 Principios de MDA

Cuatro son los principios que conforman la visión de MDA de OMG:

- La expresión de los modelos en una notación bien definida es un punto imprescindible para entender sistemas empresariales escalables.

- La construcción de sistemas puede organizarse mediante un conjunto de modelos mediante la imposición de na serie de transformaciones entre ellos, organizados en un marco arquitectónico de capas y de transformaciones
- Un respaldo formal para describir los modelos en un conjunto de meta-modelos que facilitan la integración y la transformación entre modelos, y es la base para la automatización a través de herramientas
- La aceptación y adopción generalizada de este enfoque basado en modelos requiere de estándares de la industria para proporcionar transparencia a los consumidores y fomentar la competencia entre los proveedores.

4.2.3 Modelos

Para dar soporte a estos principios, la OMG ha definido una serie de capas y transformaciones que proporcionan un marco conceptual y un vocabulario para MDA. Principalmente, OMG identifica tres tipos de modelos:

Ilustración 8 - Proceso MDA

- **Modelo Independiente de la Computación (CIM** o Computation Independent Model**):** sería el artefacto que reúne los requisitos del sistema
- **Modelo Independiente de la Plataforma (PIM** o Platform Independent Model): es el resultado de modelar el sistema de forma independiente a la plataforma de destino
- **Modelo Específico de la Plataforma (PSM** o Platform Specific Model): se obtiene un modelo específico para la plataforma, por ejemplo mediante el uso de un DSL construido previamente o lenguajes de propósito general

4.2.4 Beneficios de MDA

Entre los beneficios de la utilización de MDA podemos encontrar los siguientes:

- Reducción del coste de desarrollo de software
- Reducción del tiempo de desarrollo de software
- Integración rápida y sencilla de código antiguo o heredado además de tecnologías emergentes
- Mantenimiento automático de documentación

4.3 GROOVY

Groovy es un lenguaje dinámico, opcionalmente tipado, orientado a objetos para la plataforma Java que contiene muchas funcionalidades inspiradas de otros lenguajes como Python, Ruby y Smalltalk. Emplea una sintaxis semejante a Java, reduciendo la curva de aprendizaje necesaria para que estos desarrolladores adopten Groovy.

Groovy está fuertemente atado a la plataforma Java, tanto en términos de implementación (muchas partes de Groovy están escritas en Java, mientras otras muchas en el propio Groovy) y de interacción. Al ser una extensión de Java y no pretender sustituirlo por completo, se dispone desde Groovy toda la potencia de la plataforma Java, incluyendo todas las librerías existentes para esta plataforma.

Por otro lado, Groovy ofrece una serie de funcionalidades y características que lo hacen un lenguaje ideal para la creación de DSLs. A continuación se explican brevemente algunas de estas características.

4.3.1 Omisión de paréntesis

En Groovy los paréntesis pueden omitirse bajo ciertas circunstancias. Todas las sentencias o expresiones de alto nivel pueden beneficiarse de esta regla. También pueden omitirse el ; (punto y coma) que delimita el final de una instrucción en Java.

Un ejemplo de la diferencia entre Java y Groovy sería el siguiente, en el que se puede observar una mayor facilidad de lectura y más próximo al lenguaje natural en la versión Groovy.

Java

```
move(left);
```

Groovy

```
move left
```

4.3.2 Inyección de variables y constantes

Mediante el uso del *script binding* cada script tiene un mapa especial en el que se pueden guardar variables especiales para su uso posterior. Esto facilita el uso de las variables, la lectura del código y reduce la posibilidad de introducir errores accidentales.

```
def binding = new Binding([distance: 11400, time: 5 * 60]) //crear y llenar binding
def shell = new GroovyShell(binding) //Pasar binding al GroovyShell
shell.evaluate '''
    speed = distance / time // Utilizar las variables o constantes del binding
'''
```

4.3.3 Argumentos con nombre (named-arguments)

El soporte de Groovy para los parámetros con nombre ayuda a clarificar el significado de los parámetros que se pasan a los métodos en lugar de basarse puramente en el orden de los mismos.

Cuando se realiza una llamada a un método pasando una mezcla de argumentos con nombre y si nombre, Groovy coloca todos los argumentos con nombre en un mapa, que es pasado como primer argumento, y el resto de parámetros se pasan después de este mapa en el mismo orden en que aparecen en la llamada.

Por ejemplo, suponiendo que se realiza una llamada como la siguiente

```
method argOne, keyOne: valueOne, argTwo, keyTwo: valueTwo, argThree
```

Groovy interpretará la llamada en tiempo de ejecución como

```
method([keyOne: valueOne, keyTwo: valueTwo], argOne, argTwo, argThree)
```

Y finalmente realizará la llamada al método de la siguiente forma

```
def method(Map m, argOne, argTwo, argThree)
```

4.3.4 Encadenamiento de comandos

El encadenamiento de comandos junto con la mayor flexibilidad, traducida en la no necesidad de paréntesis ni caracteres delimitadores, permite que los DSLs creados con Groovy se asemejen más al lenguaje natural.

Por ejemplo, la siguiente sentencia escrita en Groovy

```
move right by 3.meters at 5.km/h
```

Sería la equivalente a la siguiente en el modo expandido

```
move(right).by(3.meters).at(5.km/h)
```

4.3.5 Cambio de contexto con closures

Al emplear POJOs (Plain Old Java Objects) o POGOs (Plain Old Groovy Objects) con muchas propiedades o cuando se asignan valores a muchas de esas propiedades, el código puede volverse repetitivo, teniendo que repetir el nombre del objeto en cada asignación. Los constructores con argumentos con nombre pueden paliar parcialmente esta situación, pero Groovy ofrece otra posibilidad.

Mediante el *closure delegation* Groovy permite realizar un cambio de contexto mediante el uso del método *with{}*, reduciendo la repetición de código.

```
def robot = new Robot()
robot.with { //marcar el bloque en el que el objeto robot será delegado
    move left //llamada al método robot.move
    move forward
}
```

4.3.6 Transformación basada en plantillas

Groovy proporciona varias posibilidades para la generación de texto de forma dinámica, pero además dispone de un framework de plantillas que facilita la generación de texto en aplicaciones que lo requieran.

Este marco de plantillas está compuesto por la clase abstracta *TemplateEngine* que los motores deben implementar y una interfaz *Template* que será implementada por las plantillas que se generan. Incluidas en Groovy hay varios motores de plantillas:

- **SimpleTemplateEngine:** para plantillas simples
- **StreamingTemplateEngine:** funcionalmente igual al anterior, pero permite tratar textos de más de 64k.
- **GStringTemplateEngine:** almacena la plantilla como closures que permiten escritura
- **XmlTemplateEngine:** pensada especialmente para los casos en que la plantilla y la salida son XML válidos
- **MarkupTemplateEngine:** es la opción más completa y optimizada.

Para el presente trabajo se hará uso de la implementación *SimpleTemplateEngine* para la generación del código en las distintas plataformas.

4.3.7 Tratamiento de ficheros y directorios

El manejo de ficheros y directorios es una funcionalidad muy utilizada normalmente en los DSLs que realizan generación de código. Groovy facilita el trabajo con ellos tanto en la creación, borrado, lectura y escritura.

```
// We must check if the origin is a file or a directory
if (origin.isDirectory()){
    // if the origin is a directory, we'll parse all the html files inside it
    origin.listFiles(new FilenameFilter(){
        public boolean accept(File f, String filename){
            return filename.endsWith(".html")
        }
    }).each { File f ->
        createdFiles += generator.generateGraphML(f)
    }
} else {
    // if it's an html file, we only parse that file
    if (origin.getName().endsWith(".html"))
        createdFiles += generator.generateGraphML(origin)
}
```

4.4 GRAMÁTICA DE LA PROPUESTA DE DSL

Con el objetivo de facilitar el uso, se ha intentado que el DSL se parezca lo más posible al lenguaje natural en inglés, haciendo uso de los *delegates* de Groovy para incluir funciones auxiliares para dar más sentido a las frases.

Para hacer más sencillo el uso del DSL para todo tipo de usuarios, aunque se presupone que los usuarios potenciales son desarrolladores, se permite el uso del DSL sin utilizar ningún identificador inicial específico.

Sin embargo, se ofrece también la opción de utilizar el identificador SensorDSL junto con un método estático que recibe un closure como parámetro. Dentro de esta función anónima es donde se irán utilizando el resto de métodos para el uso de los sensores. Esta opción se ofrece para permitir que algunos IDEs proporcionen autocompletado.

```
SensorDSL.with {
      //Aquí irá el resto del DSL
}
```

En los siguientes apartados se omite el uso de este método estático por abreviar.

4.4.1 Cámara

Para indicar el uso de la cámara, existe un método *camera* que permitirá indicar dentro del closure que recibe como parámetro el resto de configuraciones.

```
camera {}
```

Dentro de este closure, se podrá utilizar la gramática indicada en la siguiente tabla, pudiendo emplearse los elementos en el orden que se prefiera.

	Parámetros	**Descripción**
with_name	nombre de método	Se indicará el nombre del método que se generará para realizar la captura. Si no se indica se utilizará uno por defecto
take	picture/video	Indica el tipo de captura que se realizará: *picture* para fotografías o *video* para captura de vídeo.
store_in	ruta	Se indicará como texto el nombre de la variable que contiene la ruta completa donde se almacenará la fotografía o video tomado
set_properties	closure	En el caso de captura de vídeo, se podrán indicar ciertas propiedades usando este verbo
on	closure	Dentro del closure se podrán indicar los métodos a los que llamar en función del resultado de la captura

Tabla 2 - Gramática general de la cámara

Para indicar los métodos a los que llamar tras realizar la captura de imagen o video, se deberá usar la palabra *on* y dentro indicar los métodos para cada resultado posible. Si no se indican, no se realizará la llamada con los resultados

	Parámetros	**Descripción**
success	nombre método	Método al que llamar si la captura y almacenamiento ha sido correcta
cancel	nombre método	Método al que llamar si el usuario ha cancelado la captura
error	nombre método	Método al que llamar si se produce un error en el proceso

Tabla 3 - Gramática de callbacks para la cámara

En el caso de realizar una captura de video, se podrán indicar una serie de propiedades mediante el uso del elemento *set_properties* y la gramática interna que admite.

	Parámetros		Descripción
quality	high/low/valor propio		Se debe indicar la calidad con un valor entre 0 y 1, pudiendo utilizarse las palabras de la gramática *low (0)* y *high (1)*
limiting	closure		Se indicarán los límites del vídeo
	size	valor en kilobytes o mapa	Se puede indicar el tamaño máximo con un valor en kilobytes o utilizar un mapa con la clave to: para darle mayor significado
	duration	valor en segundos o mapa	Se puede indicar la duración máxima del video con un valor en segundos o utilizar un mapa con la clave *to:* para darle mayor significado

Tabla 4 - Gramática de propiedades de vídeo

4.4.2 Sensores de movimiento

Para el uso del acelerómetro y del giroscopio también existen métodos específicos, aunque la gramática interna será la misma para ambos casos.

```
gyroscope {}
accelerometer {}
```

	Parámetros	Descripción
track	always/foreground	Permite indicar si la captura de movimiento se ha de realizar siempre o sólo cuando la aplicación se encuentre en primer plano
on_change	nombre método	Indica el nombre del método al que llamar cuando se capture un cambio de movimiento

Tabla 5 - Gramática de sensores de movimiento

4.4.3 Localización

Para el uso de la localización existe otro método que puede invocarse desde el closure principal con el nombre *location*.

```
location {}
```

Con la gramática propia de la localización se podrá generar tanto un método para obtener la última localización conocida como para usar el servicio de localización para mantener actualizada de forma constante la localización del dispositivo.

	Parámetros		Descripción
get	last_location		Indica que se quiere obtener la última localización conocida.
	call	nombre de método	Indica el nombre del método al que llamar cuando se obtenga la última localización
track	automatically/manually		Indica que se quieren recibir cambios en el posicionamiento. Si se indica *automatically*, el observador lo activará el código generado, sino será el usuario el que deberá realizar las llamadas a los métodos proporcionados

Tabla 6 - Gramática general para la localización

Al elegir la opción de recibir actualizaciones en los cambios de posición, se pueden indicar una serie de configuraciones.

	Parámetros		**Descripción**
calling	nombre método		Nombre del método al que llamar con los cambios de posicionamiento
start_method	nombre método		Nombre del método que se quiere dar al que permite arrancar el seguimiento de la localización
stop_method	nombre método		Nombre del método que se quiere dar al que permite parar el seguimiento
with	closure		Permite indicar ciertas propiedades para el tracking
	each	intervalo en segundos	Cada cuantos segundos se solicitan actualizaciones
	mínimum	intervalo en segundos	intervalo mínimo que se necesita para procesar las peticiones, ya que pueden llegar más rápido que lo indicado en el intervalo
	accuracy	balanced / accurate / low_power / power_safe	Indica la precisión con la que se quieren obtener las actualizaciones

Tabla 7 - Gramática para el tracking de localización

4.5 EJEMPLO COMPLETO DE DSL

A continuación se muestra un ejemplo utilizando todas las opciones que ofrece el DSL desarrollado. En un ejemplo real lo más probable es que no se mezclen tantas opciones, pero se juntan aquí en un único ejemplo por abreviar y por demostrar que pueden emplearse de forma conjunta.

```
camera {
      with_name “capturePicture” take picture store_in “pictureUri” on {
            success “pictureTaken” cancel “pictureCancelled” error “pictureError”
      }
      with_name “captureVideo” take video store_in “videoUri” set_properties {
          quality high limiting {
             size 20000 duration 300
          }
      } on {
          success “videoTaken” cancel “videoCancelled” error “videoError”
      }
}
gyroscope {
      track always on_change “gyroscopeChanged”
}
accelerometer {
      track foreground on_change “accelerometerChanged”
}
location {
      get last_location call “lastKnowLocationCallback”
      track manually calling “locationChanged” start_method “startTracking” stop_method
“stopTracking” with {
            each 15 minimum 5 accuracy balanced
      }
}
```

4.6 EJECUCIÓN

Para ejecutar la tarea de generación se proporciona un jar ejecutable al que se deberán pasar una serie de parámetros para configurar la salida del generador.

```
java -jar DSLGenerator.jar <android|ios> <raíz> <carpetaOrigen> <carpetaDestino>
```

- El primer parámetro indicará la plataforma de destino: Android o iOS
- En el segundo parámetro se deberá indicar el directorio raíz donde se encuentra la carpeta con el código fuente que debe compilarse
- El tercer argumento es opcional, aunque deberá incluirse si se indica el cuarto. Se corresponde con el nombre de la carpeta que contiene los ficheros que deben pasar por el generador. Por defecto será *pre-src*.
- En el cuarto parámetro, también opcional, se indicará el nombre de la carpeta de destino en que crear los ficheros tras su paso por el generador. El valor por defecto es *src*.

En caso de no indicarse los parámetros necesarios o que exista algún problema con la ruta o directorios indicados, se mostrará un mensaje y se detendrá el proceso.

4.6.1 Descripción del proceso

Cuando comience el proceso, se recorrerán todos los subdirectorios que existan, buscando aquellos archivos que tengan la extensión *dsl.groovy*, que serán los que contengan el uso del DSL planteado en este trabajo.

Al encontrar uno de estos archivos, se compilará el script correspondiente. Durante esta fase se podrán obtener errores de compilación al haberse incluido una opción que aplica una comprobación de tipos durante la fase de compilación mediante transformaciones AST. Esta comprobación permitirá saber si se ha empleado una gramática incorrecta en el uso del DSL. Si se produce algún error se detendrá el proceso de generación.

```
SensorDSL sensorDSL = new SensorDSL()
binding = new Binding(sensorDSL:sensorDSL)
def compilerConfiguration = new CompilerConfiguration()
compilerConfiguration.scriptBaseClass = SensorDSLScript.class.name
compilerConfiguration.addCompilationCustomizers(
        new ASTTransformationCustomizer(TypeChecked)
)
shell = new GroovyShell(this.class.classLoader, binding, compilerConfiguration)
shell.evaluate(it)
```

A continuación se realizará la evaluación del script, obteniendo lo que se considera un modelo independiente de la plataforma. También se comprobará si existe un fichero con mismo nombre pero sin la extensión *.dsl.groovy*. Es este fichero el que será modificado para incluir las funcionalidades indicadas en el DSL. En caso de que no exista el fichero, si es requerido, se mostrará un mensaje de error y se detendrá el proceso de generación.

Partiendo de este modelo independiente de la plataforma, se realizará la generación del código nativo correspondiente a la plataforma que se haya indicado en los parámetros del comando de ejecución.

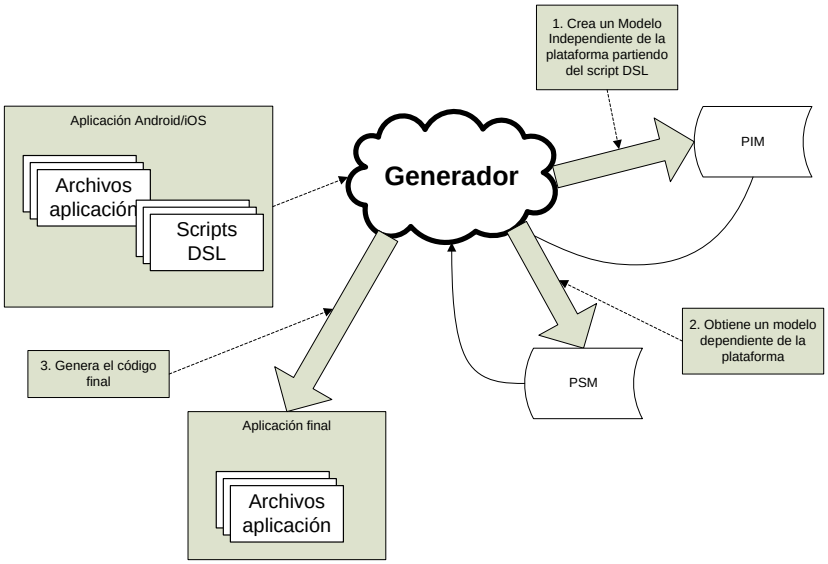

Ilustración 9 - Funcionamiento del generador

Durante este proceso de transformación se irán identificando también los datos a incluir relativos a petición de permisos y se irán almacenando en memoria. Tras rastrear todos los ficheros, se incluirán en los archivos correspondientes (Manifest.xml, info.plist…) los permisos o requisitos obligatorios que deben cumplir los dispositivos para poder ejecutar la aplicación.

Los nuevos ficheros generados o modificados, más aquellos que no necesiten de transformación, se almacenarán en la carpeta de destino. Esta carpeta es la que se ejecutará o empaquetará para producir el archivo ejecutable de los dispositivos móviles.

4.6.2 Limitaciones y notas

Al tratarse de un prototipo, el desarrollo realizado tiene una serie de limitaciones relacionados con el generador creado. Las siguientes tienen relación con los ficheros de código en los que se introducirá el nuevo código generado:

- No se pueden introducir comentarios en la misma línea de fin de método
- La definición del método y sus argumentos deben estar en la misma línea
- Debe existir un método *getContext()* que devuelva el contexto actual. (Java)
- Las aplicaciones Android deben tener especificada la dependencia con la librería android-support-v4-library (Android).
- Las sentencias de importación deben ir en líneas separadas (Java)

5 APLICACIÓN DE DSL

Para ver un ejemplo práctico del uso del DSL, se partirá de una aplicación base que permita manejar y trabajar con los sensores propuestos de forma gráfica. Esta parte gráfica y de comunicación entre vista y controladores no será generada por el DSL, sino que se creará de forma manual al no entrar dentro del alcance de esta propuesta.

5.1 APLICACIÓN EN ANDROID

5.1.1 Prototipo de aplicación

Para mostrar y comprobar el correcto funcionamiento del DSL, se ha diseñado una aplicación simple en la que se podrán utilizar los distintos sensores integrados en este prototipo.

Visualmente dispondrá de tres pestañas que permitirán acceder a tres partes diferenciadas de la aplicación:

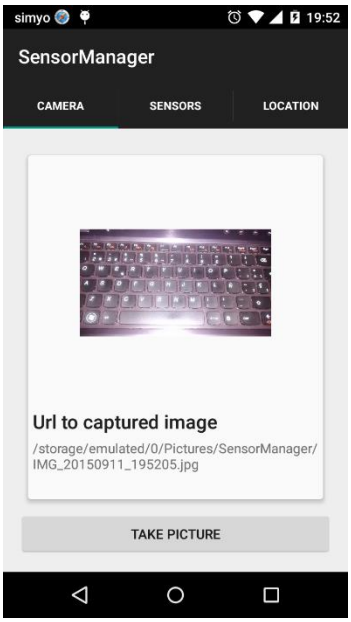

Ilustración 11 - Pestaña de cámara

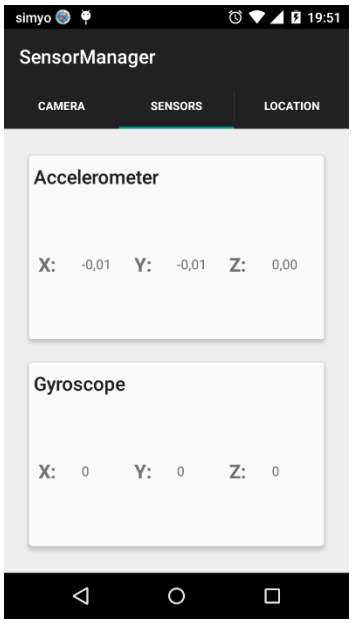

Ilustración 10 - Pestaña de sensores

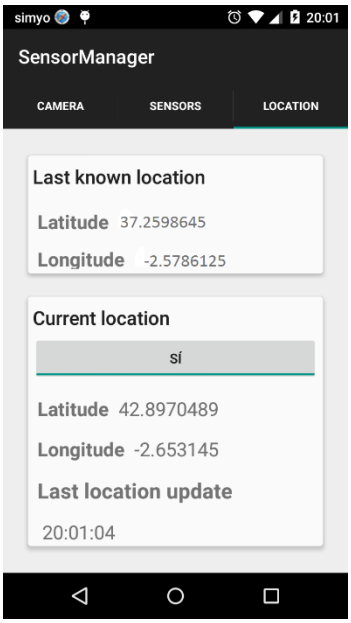

Ilustración 12 - Pestaña de localización

- **Cámara:** se podrá lanzar la aplicación de captura de imagen y también de vídeo. Si se realiza la captura, se mostrará una miniatura de la imagen y la ruta donde se ha almacenado. Si el usuario cancela, se mostrará un mensaje que indique que no se ha realizado ninguna captura.
- **Sensores:** se mostrarán los valores de los diferentes del acelerómetro (aceleración lineal y gravedad) y giroscopio, si el dispositivo los soporta. En caso de que no estén soportados, aparecerá un mensaje indicándolo.
- **Localización:** se incluirán las coordenadas de la localización del dispositivo junto con la fecha de la última actualización.

5.1.2 Aplicación del DSL en el prototipo

La navegación de la aplicación de ejemplo se basa en una única actividad que incluye tres fragmentos que se mostrarán en función de la pestaña que se seleccione.

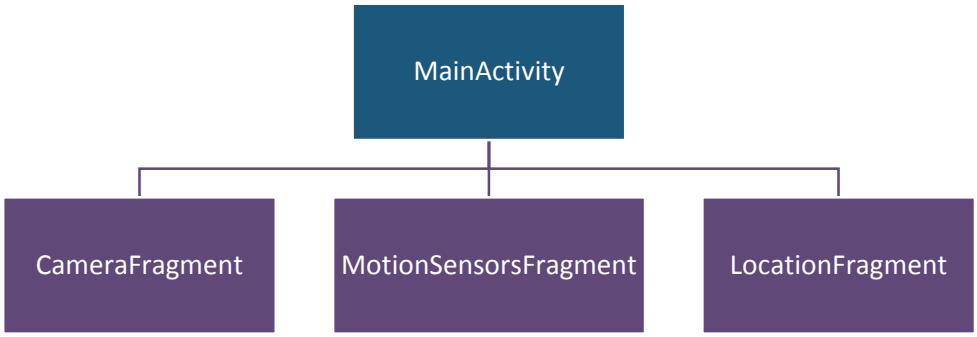

Ilustración 13 - Estructura UI aplicación Android

Para cada uno de estos fragmentos se ha creado un archivo con el mismo nombre pero con extensión *.dsl.groovy*, en la que se incluirá el DSL creado en este trabajo. El resto de ficheros pertenecientes a la aplicación serán copiados sin realizar ningún cambio a la carpeta destino.

5.1.2.1 CameraFragment

Para el fragmento en el que se usará la cámara, se ha creado el siguiente script en el que se usa el DSL que forma parte de este trabajo. El fichero se denomina *CameraFragment.dsl.groovy* y servirá para definir que se desea realizar captura de imágenes, almacenándola en una variable *fileUri* definida en el fichero de origen. Además el nombre del método que invoca la cámara será *startCamera* y habrá una serie de métodos que se invocarán en función de la respuesta del usuario ante la cámara.

```
camera {
	with_name "startCamera" take picture store_in "fileUri" on {
		success "cameraSuccessCallback" cancel "cameraCancelCallback" error
"cameraErrorCallback"
	}
}
```

Partiendo de este DSL, el generador modificará el archivo *CameraFragment* añadiendo los siguientes dos métodos que proporcionarán la funcionalidad necesaria para la captura de imágenes.

```
private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
public void startCamera(){
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);

    startActivityForResult(intent, CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
}
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
```

```
        if (resultCode == Activity.RESULT_OK) {
            cameraSuccessCallback(data);

        } else if (resultCode == Activity.RESULT_CANCELED) {
            cameraCancelCallback();

        } else {
            cameraErrorCallback();

        }
    }
}
```

5.1.2.2 MotionSensorsFragment

Se ha creado también un fichero *MotionSensorsFragment.dsl.groovy* en el que se usa el DSL propuesto para el uso del acelerómetro y del giroscopio. En este caso se escucharán los cambios únicamente mientras la aplicación esté en primer plano.

```
accelerometer {
    track foreground on_change "accelerometerCallback"
}
gyroscope {
    track foreground on_change "gyroscopeCallback"
}
```

En este caso, añadirá en la clase *SensorsFragment* las variables que se utilizan tanto en las inicializaciones que se añaden al método *onCreate* como una nueva clase *SensorDSLMotionListener* que será la que reciba las actualizaciones de los sensores y llame a los métodos oportunos definidos como callbacks.

```
private SensorManager mSensorManager;
private SensorDSLMotionListener sensorDSLMotionListener;
private Sensor mAccelerometerSensor;
private Sensor mGyroscopeSensor;

@Override
public void onCreate(Bundle savedInstanceState) {
    …
    //Retrieve sensorManager
    mSensorManager = (SensorManager) mContext.getSystemService(Context.SENSOR_SERVICE);
    //Retrieve sensors
    mGyroscopeSensor =     mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    mAccelerometerSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    sensorDSLMotionListener = new SensorDSLMotionListener();
```

```
    mSensorDSLLocationListener = new SensorDSLLocationListener(getContext());
    …
}

public class SensorDSLMotionListener implements SensorEventListener {
    @Override
    public void onSensorChanged(SensorEvent event) {
        switch (event.sensor.getType()){
            case Sensor.TYPE_ACCELEROMETER:
                accelerometerCallback(event.values);
                break;
            case Sensor.TYPE_GYROSCOPE:
                gyroscopeCallback(event.values);
                break;
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
}
```

5.1.2.3 LocationFragment

En el archivo *LocationFragment.dsl.groovy* se empleará el DSL para indicar que se quiere obtener la última localización conocida y debe ser enviada a un método *getLastLocation*. Además se piden actualizaciones de la localización, que será activado y desactivado de forma manual mediante los métodos *startLocationUpdates* y *stopLocationUpdates*. El método al que se llamará cada vez que se detecte un cambio de localización será el denominado *locationChanged*.

```
location {
    get last_location call "getLastLocation"
    track manually calling "locationChanged" start_method "startLocationUpdates"
stop_method "stopLocationUpdates"
}
```

Partiendo de este DSL, el generador creará una nueva clase interna *SensorDSLLocationListener*.

```
public class SensorDSLLocationListener implements GoogleApiClient.ConnectionCallbacks,
GoogleApiClient.OnConnectionFailedListener, LocationListener {
    private WeakReference<Context> mContext;
    private GoogleApiClient mGoogleApiClient;
    private boolean mRequestingLocationUpdates;
    private LocationRequest mLocationRequest;

    public SensorDSLLocationListener(Context context){
        this.mContext = new WeakReference(context);
        buildGoogleApiClient();
         mGoogleApiClient.connect();
        this.mRequestingLocationUpdates = false;
        this.mLocationRequest = null;
    }

    protected synchronized void buildGoogleApiClient() {
        Context context = mContext.get();
        mGoogleApiClient = new GoogleApiClient.Builder(context)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .addApi(LocationServices.API)
                .build();
    }
    @Override
    public void onConnected(Bundle bundle) {}
    @Override
    public void onConnectionSuspended(int i) {}
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {}

    @Override
    public void onLocationChanged(Location location) {locationChanged(location);}

    protected void startTracking() {
        if (mLocationRequest == null) {
            createLocationRequest();
        }
        if (mGoogleApiClient.isConnected() && !mRequestingLocationUpdates) {
            LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
            mRequestingLocationUpdates = true;
        }
    }
```

```
    protected void stopMethod() {
        if (mGoogleApiClient.isConnected() && mRequestingLocationUpdates) {
            LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient,
this);
        }
        mRequestingLocationUpdates = false;
    }
    protected void createLocationRequest() {
        mLocationRequest = new LocationRequest();
    }
}
```

Además se incluirá en los métodos *onResume*, *onPause* y *onCreate* el código necesario para arrancar, parar y crear el observador generado.

```
@Override
public void onResume() {
    super.onResume();
    if (mToggleButton.isChecked()) {
        mSensorDSLLocationListener.startLocationUpdates();
    }
}

@Override
public void onPause() {
    super.onPause();
    mSensorDSLLocationListener.stopLocationUpdates();
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mSensorDSLLocationListener = new SensorDSLLocationListener(getContext());
}
```

5.1.3 Configuración en Gradle

Las aplicaciones Android modernas pueden crearse utilizando Gradle como sistema de compilación. En la aplicación de demostración creada se ha utilizado Android Studio junto con Gradle.

Una de las opciones que permite Gradle para Android es la configuración de diferentes productos o aplicaciones que comparten código o recursos. Para este proyecto esta posibilidad ofrece un gran potencial al poder configurar dos versiones de producto diferentes, una con el código original y otro con el generado de forma automática a partir del DSL. Además, los recursos que no varían como imágenes,

estilos, etc. pueden mantenerse en el directorio principal para que sean compartidos en todas las compilaciones.

La definición de estos productos se definirán dentro del bloque *productFlavors*. Si se crea una carpeta dentro de *src* con el mismo nombre que se dé al producto, no será necesaria ninguna configuración extra. Además, podremos configurar librerías y dependencias que se carguen únicamente para alguno de estos productos. En este caso particular, se colocará el Jar dentro de una carpeta denominado *sensorDSLLibs* y se indicará dentro del bloque *dependencies* que se cargarán las librerías para la compilación del producto *sensorDSL*.

A continuación se incluye el fichero gradle correspondiente a la aplicación de ejemplo desarrollada para Android.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "com.eylen.sensormanager"
        minSdkVersion 14
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }


    productFlavors {
        complete{} //En esta carpeta se almacenará el código generado
        sensorDSL {} //Código inicial junto con DSLs
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.google.android.gms:play-services:7.5.0'
    compile 'com.android.support:cardview-v7:22.2.1'
    sensorDSLCompile fileTree(dir: 'sensorDSLLibs', include: ['*.jar']) // jars
exclusivos para el directorio con los scripts de generación
}
```

5.2 APLICACIÓN EN IOS

Para iOS no se ha desarrollado una aplicación de demostración, pero a continuación se incluye información de cómo se usaría el DSL en una aplicación iOS real.

5.2.1 Cámara

Partiendo del mismo DSL de Android, se obtendrá un código propio de las aplicaciones iOS. Se permitirá tomar fotografías invocando la función *startCamera*. En función del resultado se invocará una de las funciones indicadas en los callbacks.

```
camera {
      with_name "startCamera" take picture store_in "fileUri" on {
               success "cameraSuccessCallback" cancel "cameraCancelCallback" error
"cameraErrorCallback"
      }
}
```

Se creará un método que permita lanzar la captura de imágenes, limitándolo para evitar la captura de vídeos y eliminar los controles que permiten editar la imagen durante la captura.

```
- (BOOL) startCamera: (UIViewController*) controller
              usingDelegate: (id <UIImagePickerControllerDelegate,
                                  UINavigationControllerDelegate>) delegate {
    if (([UIImagePickerController isSourceTypeAvailable:
                UIImagePickerControllerSourceTypeCamera] == NO)
            || (delegate == nil)
            || (controller == nil))
       return NO;

    UIImagePickerController *cameraUI = [[UIImagePickerController alloc] init];
    cameraUI.sourceType = UIImagePickerControllerSourceTypeCamera;

    cameraUI.mediaTypes = [[NSArray alloc] initWithObjects: (NSString *) kUTTypeImage,
nil];

    // Hides the controls for moving & scaling pictures
    cameraUI.allowsEditing = NO;
    cameraUI.delegate = delegate;
    [controller presentModalViewController: cameraUI animated: YES];
    return YES;
}
```

Se incluirá también la implementación que observará el resultado de la toma de imágenes, invocando al método *cameraCancelCallback* si la captura de imágenes se cancela o *successCameraCallback* si la toma es correcta y se almacena de forma satisfactoria.

```
@implementation CameraViewController (CameraDelegateMethods)
 // For responding to the user tapping Cancel.
- (void) imagePickerControllerDidCancel: (UIImagePickerController *) picker {
    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
    [self cameraCancelCallback]
}
// For responding to the user accepting a newly-captured picture or movie
- (void) imagePickerController: (UIImagePickerController *) picker
            didFinishPickingMediaWithInfo: (NSDictionary *) info {
    NSString *mediaType = [info objectForKey: UIImagePickerControllerMediaType];
    UIImage *originalImage, *editedImage, *imageToSave;
    // Handle a still image capture
    if (CFStringCompare ((CFStringRef) mediaType, kUTTypeImage, 0)
            == kCFCompareEqualTo) {
        originalImage = (UIImage *) [info objectForKey:
                    UIImagePickerControllerOriginalImage];

        // Save the new image to the Camera Roll
        UIImageWriteToSavedPhotosAlbum (originalImage, nil, nil , nil);
    }
    [[picker parentViewController] dismissModalViewControllerAnimated: YES];
    [picker release];
    [self cameraSuccessCallback]
}
@end
```

5.2.2 Sensores de movimiento

Usando el mismo DSL empleado para la aplicación Android, se realizará la generación del código necesario correspondiente a iOS.

```
accelerometer {
    track foreground on_change "accelerometerCallback"
}
gyroscope {
    track foreground on_change "gyroscopeCallback"
}
```

Como se menciona en la documentación, se recomienda no instanciar el elemento *CMMotionManager* más de una vez en cada aplicación. Para permitir esto se añadirá en el *AppDelegate* un método que permita obtener la instancia del mismo, si no existe ya.

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

@interface MyAppDelegate : NSObject <UIApplicationDelegate> {
  CMMotionManager *motionManager;
}

@property (readonly) CMMotionManager *motionManager;
@end
- (CMMotionManager *)motionManager
{
  if (!motionManager) motionManager = [[CMMotionManager alloc] init];
  return motionManager;
}
```

Se creará en la clase donde incluir el resto del código un método que permita acceder al *motionManager* compartido en el *AppDelegate*.

```
-(CMMotionManager *)motionManager
 {
    CMMotionManager *motionManager = nil;
    id appDelegate = [UIApplication sharedApplication].delegate;
    if ([appDelegate respondsToSelector:@selector(motionManager)]) {
        motionManager = [appDelegate motionManager];
    }
    return motionManager;
}
```

En este caso se ha indicado que se comprobará de forma manual si hay actualizaciones, por lo que se generarán unos métodos para permitir leer los valores que deberán ser invocados desde la aplicación. Además se pedirá el comienzo de las actualizaciones al cargar la vista y se pararán cuando la vista se descargue.

```
(void)viewDidLoad
{
  [super viewDidLoad];
  if ([self.motionManager isAccelerometerAvailable])
  {
        [self.motionManager startDeviceMotionUpdates];
  }
}

- (void)viewDidDisappear:(BOOL)animated
```

```
{
  [super viewDidDisappear:animated];
  [self.motionManager stopAccelerometerUpdates];
  ...
}

-(CMAcceleration) accelerometerCallback
{
      return self.motionManager.accelerometerData.acceleration;
}
-(CMGyroData)gyroscopeCallback
{
      return self.motionManager.gyroData.rotationRate;
}
```

5.2.3 Localización

```
location {
    get last_location call "getLastLocation"
    track manually calling "locationChanged" start_method "startLocationUpdates"
stop_method "stopLocationUpdates"
}
```

Partiendo de este script DSL, se incluirán las sentencias y funciones necesarias para configurar y obtener los eventos de localización, incluyendo la solicitud de la autorización, creación de la instancia *CLLocationManager* y realizar las peticiones de obtención de la localización.

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

    self.locationManager = [[CLLocationManager alloc] init];
    self.locationManager.delegate = self;
    self.locationManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation;
    if ([self.locationManager
respondsToSelector:@selector(requestWhenInUseAuthorization)]) {
        [self.locationManager requestWhenInUseAuthorization];
    }
}

-(void)startLocationUpdate
{
      CLAuthorizationStatus status = [CLLocationManager
authorizationStatus];
```

```
      if (status==kCLAuthorizationStatusAuthorizedWhenInUse) {
              [self.locationManager startUpatingLocation];
     }
}

-(void)stopLocationUpdates
{
     CLAuthorizationStatus status = [CLLocationManager
authorizationStatus];
      if (status==kCLAuthorizationStatusAuthorizedWhenInUse) {
              [self.locationManager stopUpatingLocation];
     }
}

- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray
*)locations
{
    NSLog(@"%@", [locations lastObject]);
    [self locationChanged:[locations objectAtIndex:0]];
}

-(CLLocation)getLastLocation{
    return self.locationManager.location
}
```

6 CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

A lo largo de este trabajo se ha desarrollado un DSL que toma como base los principios de MDA, al considerarse el DSL como el medio de comunicación de un modelo independiente de la plataforma a un modelo ya dependiente de la plataforma. Es el generador que toma este DSL el encargado de realizar la generación del código para la plataforma que se haya elegido.

El trabajo realizado y descrito en esta memoria debe tomarse como una prueba de concepto en la que se muestra una posible implementación de un DSL que facilite el trabajo con los diferentes sensores en las diversas plataformas de dispositivos móviles, evitando tener que duplicar código para cada una de ellas. Además proporciona una sintaxis más sencilla al ocultar ciertas complejidades y permitir escribir las sentencias en un lenguaje más natural.

El hecho de no haber incluido parte que trate con la interfaz gráfica ha provocado que el desarrollo y quizás el uso del DSL sea algo más complejo, pero pretendía demostrarse la posibilidad de aplicar un DSL en entornos más difíciles y de forma parcial. Tal como se ha mostrado en este trabajo se puede emplear un lenguaje específico para ciertas partes o funcionalidades, manteniendo el resto del código intacto, a pesar de añadir ciertas restricciones.

También se pretendía demostrar con este trabajo que las soluciones multiplataforma que generan aplicaciones nativas y no híbridas tienen un gran potencial, como también han demostrado ya algunas de las soluciones existentes. Además, la apuesta por el uso de un lenguaje específico de dominio, aunque en principio pueda parecer una barrera para la adopción por la comunidad de desarrolladores al tener que aprender un nuevo lenguaje, es a la larga una ventaja al ser fácil adaptarse a un lenguaje que se asemeja al lenguaje natural.

6.2 APLICACIÓN DE CONOCIMIENTOS ADQUIRIDOS EN EL MÁSTER

El desarrollo del DSL y el generador asociado correspondientes a este proyecto me ha servido para relacionar los diferentes conocimientos adquiridos en las diferentes asignaturas cursadas en el máster, en especial aquellas pertenecientes al itinerario de Ingeniería del Software.

Gracias a la asignatura de Arquitecturas para sistemas software he podido aplicar los conocimientos adquiridos sobre MDA y algunos de los patrones de diseño de las aplicaciones estudiados a lo largo de la asignatura.

Con los conocimientos adquiridos en la asignatura de Generación Automática de Código he podido modelar y diseñar el DSL central del trabajo, así como el generador que realiza la generación posterior del código de las aplicaciones móviles. En cuanto a la asignatura de Desarrollo de Líneas de Producto Software me ha servido para tener más perspectiva de los DSLs y sus posibles implementaciones con otros lenguajes diferentes a Groovy. Además me ha servido para tener claro cómo poder organizar o modularizar el DSL y el generador para tener una visión más comercial y reutilizable.

Las asignaturas de Modelado y Simulación de Robots y Sistemas de Percepción Visual, se alejan un poco de los conocimientos empleados en este trabajo al ser asignaturas pertenecientes a la otra rama del máster. Sin embargo me han servido (junto con el resto de asignaturas mencionadas) para aplicar conocimientos más transversales como la realización de análisis de soluciones existentes, estado del arte, creación documentación técnica y organización.

6.3 LÍNEAS FUTURAS

En los siguientes apartados se indican algunas vías de mejora o continuación del presente trabajo.

6.3.1 Integración con IDEs

Para facilitar la adopción del DSL por parte de los desarrolladores sería conveniente crear plugins o elementos que permitan integrarlo de manera sencilla en los diferentes entornos de desarrollo. En aquellos que se usen para desarrollar en Java o lenguajes que extiendan de él, esta integración es más sencilla ya que identificarán los métodos y propiedades del DSL tal como se presenta en este trabajo.

También podría plantearse la posibilidad de realizar una integración visual del DSL dentro del entorno de desarrollo que permitiera, por ejemplo, enlazar los métodos a los que llamar de forma gráfica.

6.3.2 Soporte para más plataformas

Aunque se han elegido las plataformas de dispositivos móviles con mayor penetración en el mercado, una mejora clara sería la inclusión de nuevas plataformas soportadas por el DSL y el generador, como podrían ser Firefox OS o Windows 10 para móviles.

6.3.3 Integración de más sensores

El DSL presentado permite el uso de un grupo limitado de las posibilidades de los dispositivos móviles actuales. Una vía de mejora sería la ampliación del DSL y del generador de código para permitir el uso de más de estos sensores o de opciones más avanzadas. Por ejemplo se podrían incluir sensores de luminosidad, temperatura o facilitar el trabajo con las balizas o beacons.

6.3.4 Integración de más funcionalidades

Aunque el trabajo se ha centrado en el trabajo con algunos de los sensores que ponen a disposición los dispositivos móviles, podría ampliarse el DSL o crear otros independientes que permitieran, por ejemplo, realizar la creación de la interfaz gráfica junto con la navegación o el uso de otros servicios como puede ser la mensajería push.

6.3.5 Librería con elementos comunes

Al aumentar el contenedor del generador y del DSL sería conveniente crear una librería para cada plataforma, que proporcione ciertas funcionalidades comunes para todas las aplicaciones como pueden ser clases de ayuda, interfaces, observadores, etc.

7 LISTA DE REFERENCIAS Y BIBLIOGRAFÍA

DSL

[1] A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. ACM Sigplan Notices, 2000.

[2] Ghosh, Debasish. DSLs in Action. Manning Publications Co., 2011

Groovy

[3] Dierk König, A. Glover, P. King, G. Laforge, J. Skeet. Groovy in Action. Manning Publications Co., 2007

[4] Dierk Köning, Paul King, G. Laforge, H. D'Arcy, C. Champeau, E. Pragt, J. Skeet. Groovy In Action Second Edition. Manning Publications Co., 2015.

[5] Venkat Subramaniam. The Pragmatic Programmers: Programming Groovy. The Pragmatic Bookshelf, 2008

[6] Documentación oficial de Groovy. Última visita: 10/09/2015. http://www.groovy-lang.org/documentation.html

Rhodes

[7] Documentación oficial de Rhodes. Última visita: 20/07/2015. http://docs.rhomobile.com/en/2.2.0/rhodes/introduction

[8] Ejemplos de uso de Rhodes. Última visita: 20/08/2015. https://github.com/rhomobile/rhodes-system-api-samples

LiveCode

[9] Documentación oficial de LiveCode. Última visita: 20/08/2015. http://livecode.com/guides-documentation

[10] Ejemplos de LiveCode. Última visita: 20/08/2015. http://www.robertcailliau.eu/Alphabetical/L/LiveCode/Annotated%20Examples/

[11] Guía para desarrolladores de LiveCode. Última visita: 21/08/2015. https://livecode.com/resources/guides/developers-guide/

Tabris.js

[12] Documentación oficial de Tabrisjs. Última visita: 18/08/2015. https://tabrisjs.com/documentation/latest/

[13] Ejemplos de Tabrisjs. Última visita: 19/08/2015. http://eclipsesource.com/blogs/2015/02/20/tabris-js-examples-hello-world/

[14] Ejemplos de Tabrisjs. Última visita: 19/08/2015. https://tabrisjs.com/examples/

APPlause

[15] Repositorio oficial de APPlause. Última visita: 15/08/2015. https://github.com/applause/applause

[16] Vídeo demostración de APPlause. Última visita: 01/09/2015. https://vimeo.com/15018235

[17] Applitude: Elegant DSL to create iPhone apps in Eclipse. Última visita: 16/08/2015. https://changelog.com/applitude-applitude-is-an-objective-c-runtime-framework/

[18] Herman Lintvelt. DSLs and Mobile App Development. Última visita: 16/08/2015. https://changelog.com/applitude-applitude-is-an-objective-c-runtime-framework/

Titanium SDK

[19] Do4cumentación official de Titanium SDK. Última visita: 17/08/2015. http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_SDK_Quick_Start

[20] Ejemplos de Titanium SDK. Última visita: 17/08/2015. https://wiki.appcelerator.org/display/guides2/Titanium+Samples

Android

[21] Documentación de uso de la cámara. Última visita: 02/09/2015. http://developer.android.com/guide/topics/media/camera.html

[22] Documentación de uso del acelerómetro. Última visita: 02/09/2015. http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-accel

[23] Documentación de uso del giroscopio. Última visita: 02/09/2015. http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-gyro

[24] Documentación de uso de los servicios de localización. Última visita: 03/09/2015. https://developer.android.com/intl/ko/training/location/retrieve-current.html4

[25] Katarina Mitchell. Background Location Updates on Android. Última visita: 03/09/2015. https://uncorkedstudios.com/blog/background-location-updates-on-android

iOS

[26] Documentación de uso de la cámara. Última visita: 03/09/2015. https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/CameraAndPhotoLib_TopicsForIOS/Articles/TakingPicturesAndMovies.html

[27] Documentación de uso de sensores de movimiento. Última visita: 02/09/2015. https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CMMotionManager_Class/

[28] Documentación de uso de servicios de localización. Última visita: 04/09/2015. https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CLLocationManager_Class/

[29] Mike Lazer-Walker. Core Location in iOS 8. Última visita: 03/09/2015. http://nshipster.com/core-location-in-ios-8/

MDA

[30] MDA – The architecture of choice for a changing world. Última visita: 01/08/2015. http://www.omg.org/mda/

[31] MDA Guide Revision 2. Última visita: 01/08/2015. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf

Otros trabajos relacionados

[32] Olivier Le Goaer, Sacha Waltham. Yet another DSL for Cross-Platforms Mobile Development. Universidad de Pau, Francia. 2013

[33] Kristoffer Rosén. A Domain-Specific Language for Cross-Platform Smartphone application development. Departamento de Ingeniería del Software, Universidad de Lund. 2013

[34] Carlos Enrique Montenegro, Juan Manuel Cueva, Óscar Sanjuán, Paulo Alonso Gaona. Desarrollo de un lenguaje de dominio específico para sistemas de gestión de aprendizaje y su herramienta de implementación “KiwiDSM” mediante ingeniería dirigida por modelos. Universidad Distrital Francisco José de Caldas. 2010

8 GLOSARIO DE TÉRMINOS

A

API
La interfaz de programación de aplicaciones, abreviada como API (del inglés Application Programming Interface), es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción 7, 9, 10, 15, 20, 21, 22, 24

D

DSL
Domain Specific Language o Lenguaje Específico del Dominio. 1, 3, 5, 7, 9, 10, 11, 12, 33, 34, 35, 36, 39, 40, 42, 43, 45, 46, 47, 48, 50, 52, 53, 55, 57, 58, 59, 60, 61

G

GPL
General Purpouse Language o Lenguaje de Propósito General 3
Groovy
lenguaje dinámico, opcionalmente tipado, orientado a objetos para la plataforma Java que contiene muchas funcionalidades inspiradas de otros lenguajes como Python, Ruby y Smalltalk 1, 3, 33, 35, 36, 37, 38, 39, 57, 59

H

HTML
siglas de HyperText Markup Language (.....9, 11, 12, 33

I

IDE
Un ambiente de desarrollo integrado, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software 15

J

JavaScript
es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. 9, 10, 11, 12, 33

M

MDA
Model Driven Architecture o Arquitectura Dirigida por Modelos.....35, 36, 57, 61

O

Objective-C
lenguaje de programación orientado a objetos creado como un superconjunto de C para que implementase un modelo de objetos parecido al de Smalltalk. Lenguaje utilizado para aplicaciones iOS 14, 15

P

plugin
Aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software . 7, 11, 51, 58

R

Ruby
un lenguaje de programación interpretado, reflexivo y orientado a objetos 9, 12, 36

S

SDK
Un kit de desarrollo de software o SDK (siglas en inglés de software development kit) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones para un sistema concreto..... 11, 13, 15

W

WebView
Se trata de una vista que muestra páginas web 9, 10, 11, 12

X

XML
eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.....11, 12, 33, 39

XMLSchema
un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML..... 11

Xtext
Entorno de código abierto para el desarrollo de lenguajes de programación y lenguajes específicos de dominio. Es un proyecto de Eclipse. 11