

*Máster Universitario de Investigación  
en Ingeniería de Software y Sistemas  
Informáticos*

Itinerario Ingeniería de Software (Código 31105128)

**GEN MÓVIL:  
Generador Automático de  
Aplicaciones Móviles**

Alumno: Miguel Ángel Muñoz González  
Director: Ismael Abad Cardiel

*Curso 2012/2013  
Convocatoria Junio 2013*

*Máster Universitario de Investigación  
en Ingeniería de Software y Sistemas  
Informáticos*

Itinerario Ingeniería de Software (Código 31105128)

**GEN MÓVIL:  
Generador Automático de  
Aplicaciones Móviles**

Generador de código automático basado en HTML5 como DSL que tiene como objetivo plataformas orientadas a dispositivos móviles: iOS y Android

Alumno: Miguel Ángel Muñoz González  
Director: Ismael Abad Cardiel





IMPRESO TFDm05\_AUTOR  
AUTORIZACIÓN DE PUBLICACIÓN  
CON FINES ACADÉMICOS



**Impreso TFDm05\_Autor. Autorización de publicación  
y difusión del TFDm para fines académicos**

## **Autorización**

Autorizo/amos 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/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16  
28040, Madrid

Tel: 91 398 89 10  
Fax: 91 398 89 09

[www.issi.uned.es](http://www.issi.uned.es)

## **RESUMEN**

Este trabajo analiza la viabilidad de un generador de código automático para dispositivos móviles multiplataforma, centrándose en iOS y Android como sistemas objetivo. Se propone HTML5 como DSL para especificar la aplicación a generar.

El estudio se enfoca en la capa de usuario, es decir, interfaces, controles y elementos visibles que permiten la interacción con el usuario de la aplicación.

El trabajo presenta un breve análisis sobre la forma de operar de cada plataforma en cuanto a interfaces de usuario. Seguidamente se identifican las partes y elementos comunes a ambas plataformas para mostrar la relación entre ellos. Tras una breve puntualización sobre consideraciones de diseño, se plantea HTML5 como lenguaje específico de diseño, con ejemplos de cada posible situación de generación de código.

Posteriormente se detalla el código a generar para cada uno de los elementos en relación con el DSL propuesto. Esto se basa en un desarrollo de un prototipo que asegura la viabilidad de la solución.

## **LISTA DE PALABRAS CLAVE**

Generación, código, móvil, ios, android, plataforma, html, dsl, java, objective-c, interfaz, control

## CONTENIDOS

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>1</b>
<b>2</b>	<b>ÁMBITO DE APLICACIÓN .....</b>	<b>2</b>
<b>3</b>	<b>PLATAFORMAS ANALIZADAS .....</b>	<b>3</b>
3.1	IOS .....	3
3.1.1	<i>Gestión de ventanas y vistas</i> .....	3
3.1.2	<i>Animaciones</i> .....	4
3.1.3	<i>Geometría</i> .....	5
3.1.4	<i>Controladores de vistas</i> .....	5
3.1.5	<i>Tipos de controladores de vista</i> .....	6
3.1.6	<i>Controles, vistas y elementos predefinidos</i> .....	8
3.1.6.1	Barras .....	8
3.1.6.2	Vistas de Contenido .....	9
3.1.6.3	Alertas y otras vistas o diálogos modales.....	11
3.1.6.4	Controles .....	12
3.2	ANDROID.....	13
3.2.1	<i>Vistas y Grupos de Vistas</i> .....	13
3.2.2	<i>Activities</i> .....	14
3.2.3	<i>Navegación</i> .....	15
3.2.4	<i>Geometría</i> .....	15
3.2.5	<i>Layouts</i> .....	16
3.2.6	<i>Animaciones</i> .....	17
3.2.7	<i>Controles</i> .....	18
<b>4</b>	<b>RELACIÓN ENTRE PLATAFORMAS .....</b>	<b>20</b>
4.1	VISTAS .....	20
4.2	ACTIVITIES (ANDROID) Y VIEW CONTROLLERS (IOS).....	20
4.3	VISTAS ESPECIALES.....	22
4.4	NAVEGACIÓN .....	23
4.5	CONTROLES .....	24
<b>5</b>	<b>CONSIDERACIONES DE DISEÑO GENERALES.....</b>	<b>27</b>
<b>6</b>	<b>HTML5 COMO DSL .....</b>	<b>29</b>
6.1	CONTENEDORES.....	30
6.2	NAVEGACIÓN .....	31
6.3	VISTAS .....	33
6.3.1	<i>Vista Formulario o simple</i> .....	33
6.3.2	<i>Vista de tabla</i> .....	33
6.3.3	<i>Vista de lista</i> .....	34
6.3.4	<i>Vista de imagen</i> .....	35
6.3.5	<i>Vista de mapa</i> .....	35
6.4	CONTROLES .....	35
6.4.1	<i>Etiqueta</i> .....	36
6.4.2	<i>Campo de texto</i> .....	36
6.4.3	<i>Botón</i> .....	36
6.4.4	<i>Selector</i> .....	37
6.4.5	<i>Interruptor</i> .....	37
6.4.6	<i>Barras deslizantes o sliders</i> .....	38
6.4.7	<i>Lenguaje</i> .....	38

<b>7</b>	<b>GENERACIÓN DE CÓDIGO .....</b>	<b>39</b>
7.1	HTML5 A IOS .....	39
7.1.1	Contenedores.....	40
7.1.2	Navegación.....	41
7.1.3	Vistas .....	44
7.1.3.1	Vista de formulario.....	44
7.1.3.2	Vistas de tabla .....	44
7.1.3.3	Vista de lista .....	46
7.1.3.4	Vista de imagen.....	47
7.1.3.5	Vista de mapa.....	48
7.1.4	Controles.....	49
7.1.4.1	Etiqueta .....	49
7.1.4.2	Campo de texto.....	49
7.1.4.3	Botón.....	50
7.1.4.4	Selector .....	50
7.1.4.5	Interruptor .....	52
7.1.4.6	Barras deslizantes o sliders .....	52
7.2	HTML5 A ANDROID.....	53
7.2.1	Contenedores.....	53
7.2.2	Navegación.....	55
7.2.3	Vistas .....	57
7.2.3.1	Vista formulario.....	57
7.2.3.2	Vista de tabla.....	58
7.2.3.3	Vista de lista .....	59
7.2.3.4	Vista de imagen.....	61
7.2.3.5	Vista de mapa.....	61
7.2.4	Controles.....	63
7.2.4.1	Etiqueta .....	63
7.2.4.2	Campo de Texto .....	63
7.2.4.3	Botón.....	64
7.2.4.4	Selector .....	65
7.2.4.5	Interruptor .....	66
7.2.4.6	Barras deslizantes o sliders .....	67
7.3	TABLAS COMPARATIVAS ENTRE LAS PLATAFORMAS .....	67
7.3.1	Contenedores.....	67
7.3.2	Navegación.....	68
7.3.3	Vistas .....	70
7.3.4	Controles.....	72
<b>8</b>	<b>PROTOTIPO.....</b>	<b>73</b>
<b>9</b>	<b>CONCLUSIONES Y TRABAJOS FUTUROS.....</b>	<b>78</b>
<b>10</b>	<b>BIBLIOGRAFÍA.....</b>	<b>79</b>
<b>11</b>	<b>SIGLAS .....</b>	<b>80</b>

## LISTA DE FIGURAS

FIGURA 1 DESGLOSE DE VENTANAS Y VISTAS .....	3
FIGURA 2 COMBINACIÓN DE DISTINTAS VISTAS Y ELEMENTOS .....	4
FIGURA 3 CONTROLADORES DE VISTA .....	5
FIGURA 4 STORYBOARD: NAVEGACIÓN ENTRE DIFERENTES VISTAS .....	6
FIGURA 5 CONTROLADORES DE CONTENIDO Y DE CONTENEDOR (NAVEGACIÓN) .....	7
FIGURA 6 CONTROLADORES DE CONTENIDO Y DE CONTENEDOR (TABULAR) .....	7
FIGURA 7 DIFERENTES BARRAS DE ESTADO.....	8
FIGURA 8 BARRA DE NAVEGACIÓN .....	8
FIGURA 9 BARRAS DE HERRAMIENTAS.....	9
FIGURA 10 BARRA PARA NAVEGACIÓN TABULAR.....	9
FIGURA 11 ESTRUCTURA JERÁRQUICA DE VISTAS Y GRUPOS DE VISTAS.....	14
FIGURA 12 DIFERENTES TIPOS DE LAYOUT: LINEAR, RELATIVE Y WEB .....	16
FIGURA 13 VISTAS DE LISTA Y REJILLA .....	16
FIGURA 14 SELECTOR EN ANDROID (EN ESTADO NORMAL Y DESPLEGADO) E IOS.....	29
FIGURA 15 CONTENEDOR EN IOS .....	40
FIGURA 16 NAVEGACIÓN LINEAL EN IOS.....	42
FIGURA 17 NAVEGACIÓN TABULAR EN IOS.....	42
FIGURA 18 EJEMPLO DE TABLA (UICOLLECTIONVIEW) EN IOS.....	45
FIGURA 19 EJEMPLO DE LISTA EN IOS (UITABLEVIEW) .....	46
FIGURA 20 EJEMPLO DE IMAGEN EN IOS (UIIMAGEVIEW) .....	48
FIGURA 21 EJEMPLO DE MAPA EN IOS (MKMAPVIEW).....	48
FIGURA 22 EJEMPLO DE ETIQUETA EN IOS (UILabel).....	49
FIGURA 23 EJEMPLO DE CAMPO DE TEXTO EN IOS (UITEXTFIELD) .....	49
FIGURA 24 EJEMPLO DE BOTÓN EN IOS (BUTTON) .....	50
FIGURA 25 EJEMPLO DE SELECTOR EN IOS (UIPICKERVIEW) .....	50
FIGURA 26 EJEMPLO DE INTERRUPTOR EN IOS (UISWITCH).....	52
FIGURA 27 EJEMPLO DE BARRA DESLIZANTE EN IOS (UISLIDER).....	52
FIGURA 28 EJEMPLO DE CONTENEDOR EN ANDROID.....	54
FIGURA 29 EJEMPLO DE NAVEGACIÓN LINEAL EN ANDROID.....	55
FIGURA 30 EJEMPLO DE CONTROL QUE PERMITE NAVEGACIÓN TABULAR EN ANDROID .....	56
FIGURA 31 EJEMPLO DE TABLE EN ANDROID (GRIDVIEW) .....	58
FIGURA 32 EJEMPLO DE LISTA EN ANDROID (LISTVIEW) .....	60
FIGURA 33 EJEMPLO DE IMAGEN EN ANDROID (IMAGEVIEW) .....	61
FIGURA 34 EJEMPLO DE MAPA EN ANDROID (MAPVIEW) .....	62
FIGURA 35 EJEMPLO DE ETIQUETA EN ANDROID (TEXTVIEW).....	63
FIGURA 36 EJEMPLO DE ETIQUETA EN ANDROID (TEXTVIEW).....	64
FIGURA 37 EJEMPLO DE BOTÓN EN ANDROID (BUTTON).....	64
FIGURA 38 EJEMPLO DE SELECTOR EN ANDROID, EN ESTADO NORMAL Y DEPLEGADO (SPINNER) .....	65
FIGURA 39 EJEMPLO DE INTERRUPTORES EN ANDROID (SWITCH).....	66
FIGURA 40 EJEMPLO DE CHECKBOXES EN ANDROID (CHECKBOX) .....	66
FIGURA 41 EJEMPLO DE BARRA DESLIZANTE EN ANDROID (SEEKBAR) .....	67
FIGURA 42 BOTÓN Y ETIQUETA GENERADOS EN IOS Y ANDROID .....	74
FIGURA 43 ELEMENTOS DE TIPO LISTA E IMAGEN GENERADOS EN IOS Y ANDROID .....	75



## LISTA DE TABLAS

TABLE 1 RELACIÓN ENTRE CLASES PARA MODELAR VISTAS IOS Y ANDROID .....	20
TABLE 2 RELACIÓN ENTRE CLASES PARA MODELAR CONTROLADORES DE VISTA Y ACTIVITIES .....	21
TABLE 3 CÓDIGO GENERADO PARA UN VIEWCONTROLLER Y UNA ACTIVITY .....	21
TABLE 4 RELACIÓN ENTRE CLASES PARA MODELAR VISTAS ESPECIALES IOS Y ANDROID.....	22
TABLE 5 CÓDIGO NECESARIO PARA UNA IMPLEMENTAR UNA LISTA EN IOS Y ANDROID .....	23
TABLE 6 RELACIÓN ENTRE NAVEGACIÓN DE IOS Y ANDROID.....	24
TABLE 7 CÓDIGO GENERADO PARA NAVEGAR LINEALMENTE EN IOS Y ANDROID .....	24
TABLE 8 RELACIÓN ENTRE CONTROLES DE IOS Y ANDROID .....	26
TABLE 9 CÓDIGO GENERADO PARA MODELAR UNA BARRA DESLIZANTE .....	26
TABLE 10 CÓDIGO GENERADO PARA MODELAR UN CONTENEDOR EN IOS Y ANDROID.....	68
TABLE 11 CÓDIGO GENERADO PARA MODELAR NAVEGACIÓN LINEAL EN IOS Y ANDROID .....	69
TABLE 12 CÓDIGO GENERADO PARA MODELAR NAVEGACIÓN TABULAR EN IOS Y ANDROID .....	69
TABLE 13 CÓDIGO GENERADO PARA MODELAR VISTAS DE TIPO TABLA EN IOS Y ANDROID .....	70
TABLE 14 CÓDIGO GENERADO PARA MODELAR VISTAS DE TIPO LISTA EN IOS Y ANDROID .....	71
TABLE 15 CÓDIGO GENERADO PARA MODELAR ETIQUETAS EN IOS Y ANDROID .....	72
TABLE 16 CÓDIGO GENERADO PARA MODELAR BOTONES EN IOS Y ANDROID .....	72

# 1 Introducción

Este trabajo constituye una aproximación a la generación de código en múltiples plataformas de dispositivos móviles. Una vez especificada una aplicación resulta costoso implementar la misma en cada una de las plataformas disponibles. Por ello, el último objetivo de este trabajo es la simplificación de la tarea de implementación en cada una de las plataformas de desarrollo (solo para móviles). Para conseguir este objetivo, este trabajo trata de proponer y analizar la viabilidad de un generador de código automático.

El generador de código debe partir de un lenguaje específico de dominio, independiente de plataforma, y ser capaz de proporcionar la implementación de la aplicación especificada en cada plataforma con intervención mínima del desarrollador. El lenguaje de dominio deberá ser abstracto, independiente y suficientemente amplio como para cubrir de forma genérica las posibilidades existentes en todas las plataformas y no sufrir ninguna limitación. Esta es una de las razones por las que se ha elegido HTML5 como DSL.

Esta propuesta analiza la posibilidad de trabajar con dos de las plataformas más importantes en la fecha de la elaboración de este trabajo: iOS y Android, actualmente las dos plataformas más usadas en dispositivos móviles. La elección de estas plataformas resulta muy ilustrativa debido a que son muchas las diferencias entre ambas y en todos los aspectos: desde los componentes de la interfaz, la forma de manejar las aplicaciones, gestión interna de los elementos y sobre todo el lenguaje usado en cada una: Objective-C y Java, respectivamente.

El trabajo se centra en la capa de usuario, es decir, en la generación de interfaces. Sin embargo su objetivo principal es demostrar que la aproximación es válida para continuar con cualquier otro aspecto del desarrollo, por ejemplo el almacenamiento en base de datos, la gestión de eventos o el uso de elementos hardware específico tales como los sensores de luz, orientación o posicionamiento.

El resultado sugiere un posible desarrollo del generador, cubriendo los aspectos más importantes que muestren la extensibilidad y ampliación del trabajo en todos los niveles y que incluso admita la generación de código en otras plataformas.

## 2 **Ámbito de aplicación**

La generación de código puede estructurarse en tres capas: Datos, Usuario y Proceso. La capa de datos se encarga de la gestión de los mismos, bien en sistemas locales o remotos y con diversos tipos de paradigmas: relacional, orientado a objetos, etc. La capa de proceso conlleva la gestión de los mecanismos internos de cada plataforma en cuanto a manejo de procesos, hilos de ejecución, permisos, prioridades, intercambio de mensajes, etc. Finalmente la capa de usuario es la más externa y visible dado que implica la gestión de interfaces y elementos o controles que permiten interactuar al usuario final. El ámbito de este trabajo está localizado en la capa de usuario y el propósito es extenderlo en futuras revisiones al resto de capas y plataformas, así como añadir mayor detalle sobre áreas más concretas. Por ejemplo, en trabajos posteriores se abordaría la lógica de los eventos asociados a los recursos gráficos: la generación del código Java en Android o C++ en iOS partiendo de Javascript en HTML5

En cuanto a la capa de usuario, el trabajo estudia la posibilidad de generar código relacionado con controles, vistas y en general elementos gráficos de interfaz de usuario. Este estudio se orienta a analizar la viabilidad de una solución que además sea lo suficientemente genérica como para que permita ser ampliada posteriormente con mayor detalle y extensible a un número mayor de controles y elementos más específicos.

Aunque el objetivo principal del trabajo sea centrarse en elementos y conceptos genéricos, se muestran también algunos casos específicos que se han considerado más relevantes para el estudio. El trabajo sirve como aproximación inicial al problema y la restricción del dominio a la capa de usuario es solo el principio de un desarrollo posterior.

## 3 Plataformas analizadas

### 3.1 iOS

El sistema operativo iOS se basa en el modelo vista controlador MVC, además de otros populares patrones utilizados para transferir información, gestionar eventos o implementar callbacks (Delegation, Target-Action, Block objects). Este trabajo se centrará en la parte relacionada con la gestión de vistas y los interface de usuario.

#### 3.1.1 Gestión de ventanas y vistas

Para este propósito, se necesitará el concepto UIWindow. En iOS, las UIWindow se usan para contener la parte visible de nuestra aplicación y forman parte activa en la gestión de eventos de pulsación y como parte de los controladores de vistas (por ejemplo en los cambios de orientación). Es obligatorio definir al menos una por aplicación y en general este será el número habitual de instancias que se manejarán. Suelen estar asociadas a pantallas o dispositivos físicos, por ejemplo un display externo. Nótese que estas ventanas son contenedores en blanco de UIViews. Por defecto no contienen menús, botones o elementos similares.

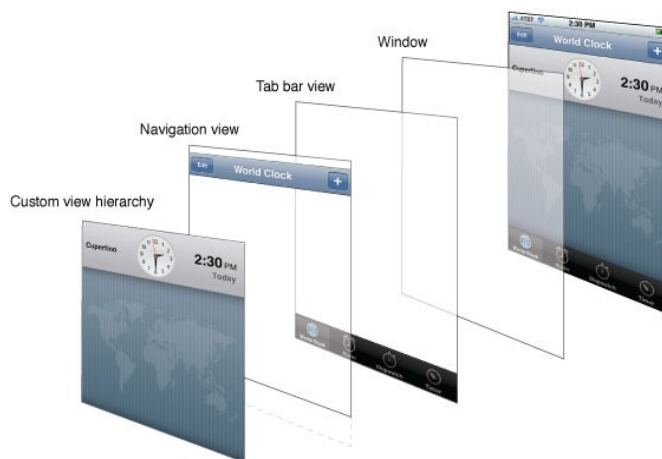
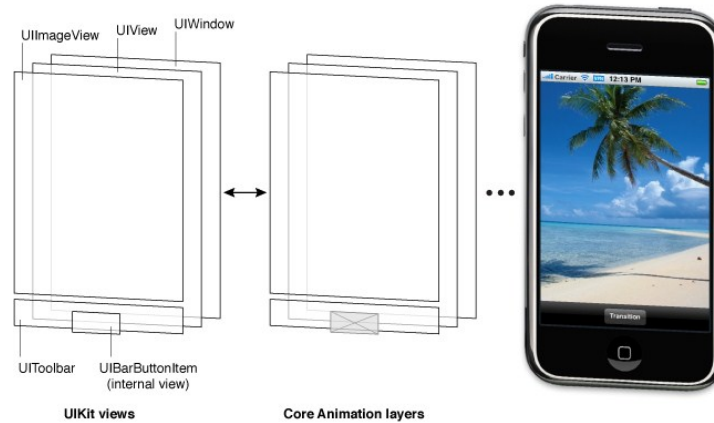


Figura 1 Desglose de ventanas y vistas

Mediante las vistas `UIView` se define un área rectangular en la ventana sobre la que podemos posicionar controles y elementos nativos, tales como imágenes, botones, etc. De hecho los controles se consideran vistas especiales. Además maneja los eventos de pintado e interacción con dicha región. Las vistas pueden formar jerarquías y contener, además de otros elementos, otras subvistas.



**Figura 2** Combinación de distintas vistas y elementos

### 3.1.2 Animaciones

Para manejar las animaciones asociadas a las vistas, se utiliza el objeto `CALayer`. En general, cada vista lleva asociada una instancia de este tipo de objeto. Para el uso normal de las vistas no es necesario manejarlo explícitamente, pero en situaciones que requieran mayor control sobre el renderizado o las animaciones se puede usar directamente.

Las animaciones disponibles nos permiten aplicarlas a cambios de tamaño, posición, rotación, transiciones entre vistas, etc.

Existe un mayor control sobre animaciones con el cual se podrían definir a medida, usando Core Animation Layers.

### 3.1.3 Geometría

El sistema de coordenadas de iOS comienza en la esquina inferior izquierda y usa valores en punto flotante para mayor precisión (no usa píxeles reales del dispositivo sino puntos lógicos). Un punto no se corresponde necesariamente con un pixel físico. Estos datos son siempre dados con referencia a un sistema de coordenadas que puede ser el de la vista o del contenedor en el que se está trabajando. Nótese que todos los dispositivos Apple tienen programáticamente la misma resolución (para todos los iPhone: 320x480 puntos lógicos y para todos los iPad 768x1024) que es trasladada a la resolución física real en píxeles por el sistema operativo. De esta forma se le evita al programador tener que controlar diferentes resoluciones según el dispositivo o la versión.

### 3.1.4 Controladores de vistas

Los controladores de vista (View Controllers) forman parte fundamental del modelo MVC y entre sus diversas funciones, se encuentran gestionar cómo los datos aparecen en pantalla, controlar conjuntos de vistas, gestionar el contenido y coordinar su visualización. Además proporcionan un comportamiento común a todas las aplicaciones mediante los controladores estándar predefinidos.

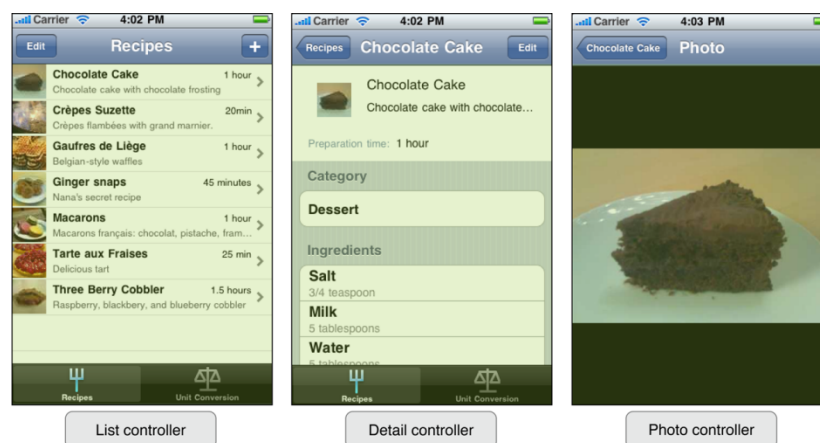


Figura 3 Controladores de vista

Los controladores también se comunican con otros controladores para establecer relaciones que suponen en la práctica transiciones entre diferentes vistas.

Nótese que para entender y desarrollar mejor las relaciones entre todos los controladores, se pueden definir storyboards que facilitan la comprensión y mejoran el desarrollo e implementación de dichas relaciones.

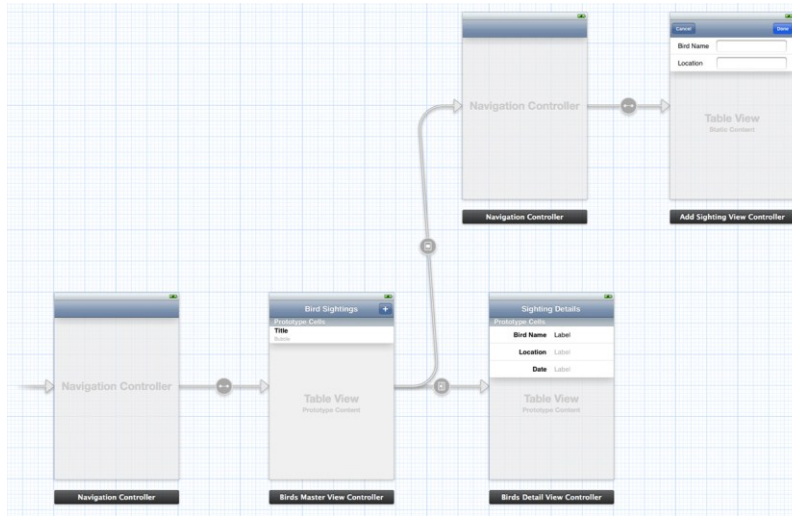


Figura 4 Storyboard: navegación entre diferentes vistas

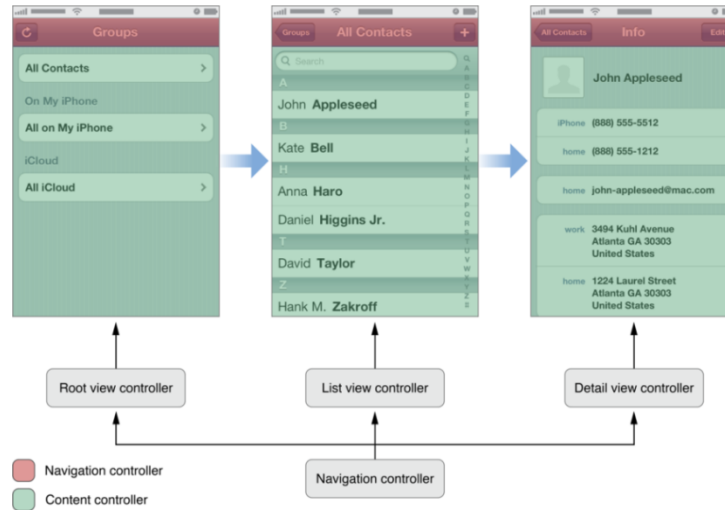
### 3.1.5 Tipos de controladores de vista

Los controladores de vista se dividen en dos tipos, de contenido (Content) y de contenedor (Container).

Los 'content view controllers' muestran o recogen datos de usuario, presentan diversas opciones sobre las que se puede navegar o realizan tareas específicas. Consisten en una o varias vistas, organizadas jerárquicamente, que conocen y muestran el contenido de los datos que se van a mostrar en pantalla.

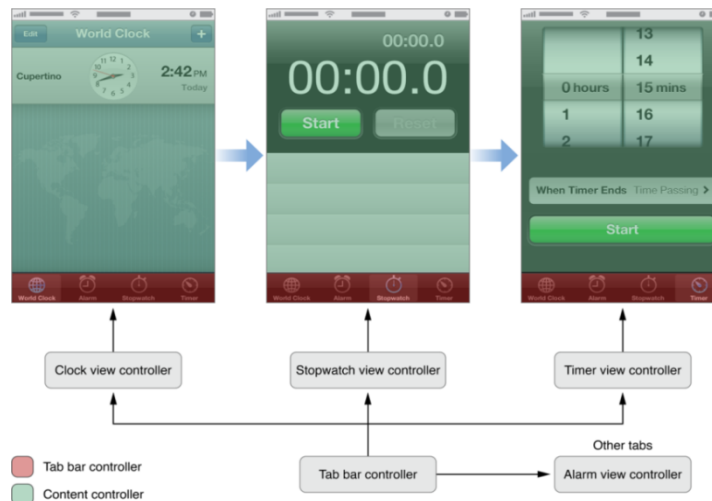
Por otra parte, los 'container view controllers' trabajan con varios hijos que son a su vez otros controladores de vista (generalmente de tipo 'content'). Cada controlador de tipo container establece un interfaz de usuario en el que los controladores de vista hijos operan. Existen varios predefinidos aunque siempre podemos definir los nuestros propios.

Por ejemplo, el “Navigation Controller” UINavigationController contiene una pila de controladores de contenido que representa el camino que ha tomado el usuario mientras navega por distintos controladores de vistas. La figura muestra un ejemplo:



**Figura 5 Controladores de contenido y de contenedor (navegación)**

Otro ejemplo de controlador de vista de tipo contenedor es el UITabBarController, que resulta típico en las aplicaciones de iOS. Permite navegar entre distintas partes de la aplicación, pero a diferencia del anterior, no apila las distintas vistas por las que el usuario navega. Se puede utilizar para dividir nuestra aplicación en distintos modos o pantallas de presentación.



**Figura 6 Controladores de contenido y de contenedor (tabular)**



Para navegar entre distintos view controllers podemos aplicar transiciones que iOS denomina 'segues'. Dichas transiciones pueden ser de tipo:

- Push segue: Introduce el controlador de vista siguiente en la pila del controlador de navegación.
- Modal segue: Presenta el controlador de vista indicado sin permitir más transiciones salvo la vuelta al anterior.
- Popover segue: Muestra el controlador de vista en un popover (menú contextual posicionado en el lugar que recibe la atención del usuario y que generalmente ocupa una pequeña parte de la pantalla).
- Custom segue: Transición a medida definida por el usuario.

### 3.1.6 Controles, vistas y elementos predefinidos

iOS incluye un amplio conjunto de controles, vistas y elementos estándar predefinidos que definen un comportamiento común en todas las aplicaciones, así como actualizaciones automáticas de los mismos. Las categorías disponibles incluyen Barras, Vistas de Contenido, Alertas y Vistas Modales y finalmente Controles.

#### 3.1.6.1 Barras

**Status bar:** Se trata de la barra habitual de la parte superior que muestra información básica del dispositivo (operadora, hora, batería, etc.)

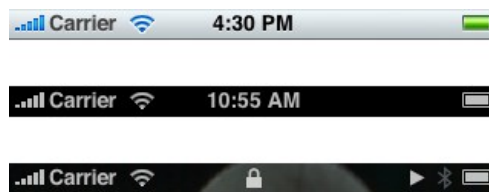


Figura 7 Diferentes barras de estado

**Navigation bar:** Permite navegar entre distintos contenidos de forma jerárquica.



Figura 8 Barra de navegación

**Tool bar:** Permite ejecutar diferentes acciones sobre objetos en la pantalla



Figura 9 Barras de herramientas

**Tab bar:** Permite cambiar entre distintas tareas o vistas.

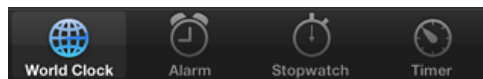


Figura 10 Barra para navegación tabular

### 3.1.6.2 Vistas de Contenido

Sin ser exactamente controles, iOS cuenta con diversos elementos que pueden clasificarse como vistas y controladores de vista que constituyen componentes predefinidos. Éstos son habitualmente usados en las aplicaciones iOS.

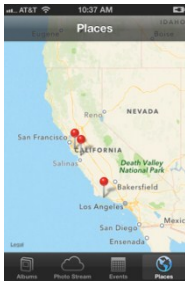


**Activity y Activity View Controller:** Presenta una vista transitoria que lista servicios del sistema (y algunos creados a medida) que se pueden ejecutar sobre un determinado contenido (por ejemplo, abrir un fichero adjunto por diversos servicios de visualización: video, e-books, etc.)



**Collection View:** Gestiona una colección ordenada de elementos que se muestran en una configuración modificable.

**Image View:** Muestra una imagen, o una animación basada en series de imágenes.

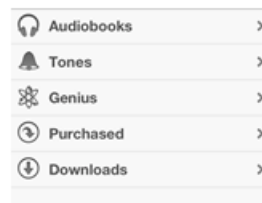
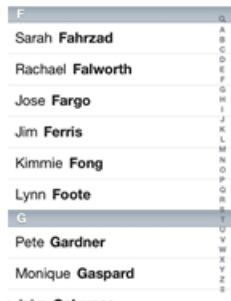


**Map View:** Muestra datos geográficos y soporta la mayoría de la funcionalidad de la aplicación de mapas de iOS.



**Scroll View:** Se utiliza para mostrar contenido que es más grande que la pantalla del dispositivo.

**Table View:** Presenta datos en múltiples filas y una sola columna



**Text View:** Se usa para mostrar y recoger texto en múltiples líneas.



**Web View:** Muestra una región que contiene HTML.


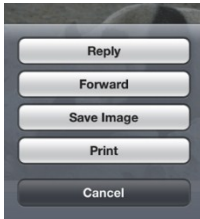



Otros controles menos comunes:

Page View Controller	Popover	Split View
Muestra múltiples vistas que transicionan mediante animaciones emulando un libro	(Solo Ipad) Se trata de una vista transitoria que aparece cuando se pulsa un control o zona específica.	Es una vista a pantalla completa que consiste en dos paneles colocados lado a lado.
		

### 3.1.6.3 Alertas y otras vistas o diálogos modales

Este tipo de vistas son temporales y aparecen para requerir la atención del usuario en momentos determinados o cuando es necesario que el usuario elija una acción para continuar. Mientras se muestran estas vistas el usuario no puede interactuar con el resto de la aplicación.

Alerta	Action Sheet	Modal View
Una alerta muestra información importante que aparece en mitad de la pantalla y flota sobre el resto de vistas. Siempre contiene al menos un botón.	Un Action Sheet muestra una lista de alternativas relacionadas con una tarea que el usuario ha iniciado.	Una vista modal contiene funcionalidad auto contenida en el contexto de la tarea actual y bloquea el resto de la aplicación hasta que se finaliza la tarea.
		

### 3.1.6.4 Controles

Activity Indicator	Date Picker	Progress View	Refresh Control
Slider	Stepper	Switch	Text Field
Rounded Rectangle Button	Search Bar	Segmented Control	
Picker	Page Control	Label	Network Activity Indicator

## 3.2 Android

En general en Android las aplicaciones muestran una estructura que combina diferentes pantallas (que son denominadas Activities) cada una orientada a una determinada función, claramente identificada y que determina el diseño y la funcionalidad de dicha Activity. Cada aplicación suele contar con diversas Activities que proporcionan diversos caminos de navegación a otras pantallas con más datos e información detallada o que realizan diferente funcionalidad.

Para realizar el cambio entre diferentes pantallas se utilizan los llamados Intents. Un Intent es una encapsulación de un mensaje que es llamado para arrancar distintas Activities, bien de la propia aplicación o de aplicaciones ajenas.

También existe el concepto de Fragments: éstos permiten definir fragmentos de interfaz considerados como Activities que pueden empotrarse en interfaces de usuario mayores.

Android usa principalmente XML para definir interfaces de usuario, de forma que su edición resulta muy sencilla y cómoda. Con ello permite además separar claramente la definición del interfaz del resto de código. Por otra parte, los interfaces también se pueden definir programáticamente desde la aplicación, lo cual aporta toda la flexibilidad posible.

Los siguientes apartados detallan los puntos clave relacionados con esta plataforma.

### 3.2.1 Vistas y Grupos de Vistas

En Android todos los interfaces de usuario se construyen usando objetos llamados Views y ViewGroups. Las Views permiten dibujar en la pantalla elementos con los que el usuario puede interactuar y con ViewGroup se pueden agrupar u organizar otras View o ViewGroup jerárquicamente.

Una vista representa el bloque básico de construcción de interfaces. Ocupa un área rectangular en la pantalla y es responsable de dibujar y gestionar los eventos que ocurren en ella. La clase View es la base de controles y widgets. Existen varias subclases de vistas predefinidas que corresponden a elementos comunes tales como botones, campos de texto, layouts, etc.

Los grupos de vistas o ViewGroups son un tipo especial de vista que puede contener otras vistas y que actúan como contenedores y layouts. El gráfico muestra la relación entre ambos conceptos.

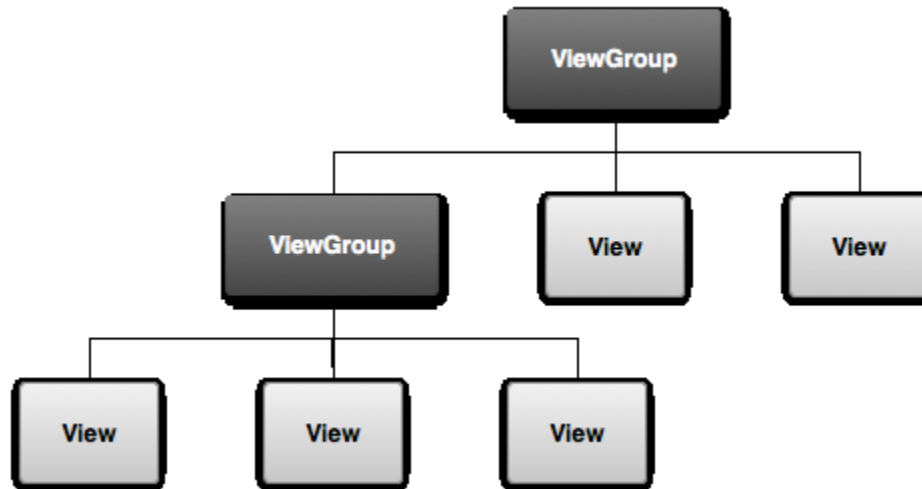


Figura 11 Estructura jerárquica de vistas y grupos de vistas

### 3.2.2 Activities

Las Activities son componentes de la aplicación que proporcionan una pantalla en la cual el usuario puede interactuar para realizar ciertas acciones, por ejemplo, marcar un número, enviar un correo o ver un mapa. Cada Activity usa la ventana completa o puede tomar parte de ella, flotando sobre el resto de ventanas visibles.

Una aplicación está formada generalmente por varias Activities, débilmente acopladas entre ellas y con un propósito individual claro y bien definido. Una Activity puede comenzar otra Activity para realizar otra tarea. En ese caso la Activity pasa al 'back stack' y deja de ejecutarse para dejar paso a la nueva que contará con el foco del usuario. Cuando el usuario termina de utilizar la Activity actual puede volver hacia atrás recuperándolas del 'back stack'.

Las Activities pueden llevar asociados 'Intent filters' que son utilizados por otras aplicaciones para activar dichas vistas. Siempre se ha de definir al menos un 'main intent filter' para indicar cuál es la Activity principal en la aplicación. En este proyecto nos centraremos en los Intent asociados a las Activities dentro de la misma aplicación.

Nótese que en los casos en los que se manejan grandes ventanas, por ejemplo en tablets, existen 'fragments' que son pequeñas partes en las que podemos dividir una Activity, con objeto de gestionarla mejor.

### 3.2.3 Navegación

Al igual que en otros sistemas, en Android es posible navegar entre diferentes vistas utilizando diversos mecanismos.

Generalmente se realizarán transiciones de una vista a la siguiente abriendo la Activity deseada mediante Intents. Esta navegación de tipo lineal está implícitamente implementada en la plataforma y no es necesario desarrollar complejos fragmentos de código para llevarla a cabo, tal y como se verá más adelante. En dicha navegación lineal, una Activity se apila sobre las anteriores a medida que el usuario navega. Se puede retroceder a petición del usuario simplemente pulsando el botón que está especialmente diseñado para ello en cada dispositivo Android.

Por otra parte, también es posible navegar tabularmente. De esta forma, una Activity que implemente un control tabular ofrece diferentes contenidos que se pueden alternar por el usuario mediante distintas pestañas o tabs. Para implementarla basta con crear un control de tipo tabular que se asocia de forma muy sencilla a Intents, los cuales abren cada una de las Activities que se desean presentar. Se debe observar que en este caso la pulsación del botón retroceso no deshace las transiciones realizadas entre distintas pestañas, sino que vuelve a la Activity previa a la inicial.

### 3.2.4 Geometría

En Android cada dispositivo cuenta con un número de píxeles determinado por lo que se recomienda consultar este número antes de hacer cálculos o posicionar elementos. Para ello basta con realizar una petición a la anchura y altura del display, mediante una llamada a la API. Por otra parte conviene mencionar que el sistema de coordenadas comienza en la esquina superior izquierda.



### 3.2.5 Layouts

Los layouts definen la estructura visual de un interfaz de usuario. Pueden anidarse para conseguir layouts más complejos o bien definirse por el programador. Algunos de los tipos de layout más utilizados son:

- Linear Layout: Organiza los contenidos en líneas horizontales o verticales.
- Relative Layout: Permite especificar la localización de los objetos hijos de forma relativa entre ellos o con relación al padre.
- Web View: Permite mostrar páginas web.



Figura 12 Diferentes tipos de layout: linear, relative y web

Para los casos en los que el layout no es fijo, por ejemplo una tabla o una rejilla de objetos con scroll, Android permite crear un nuevo layout mediante herencia de 'AdapterView' y rellenarlo en tiempo de ejecución. A su vez, mediante un 'Adapter' se establece la conexión entre los datos a mostrar y el interfaz gráfico. Dicho adaptador convierte cada dato en una vista que se puede añadir al layout de 'AdapterView'. Ejemplos habituales son ListView y GridView. Estos casos están disponibles y forman parte del SDK de Android para cualquier desarrollo.



Figura 13 Vistas de lista y rejilla

### 3.2.6 Animaciones

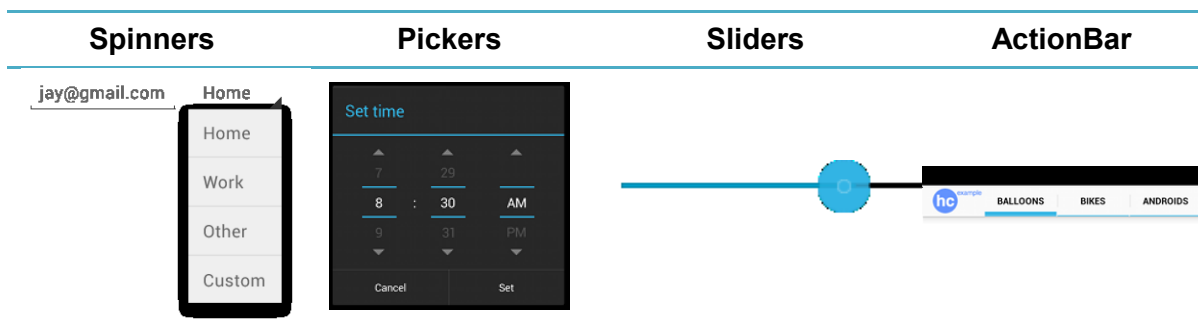
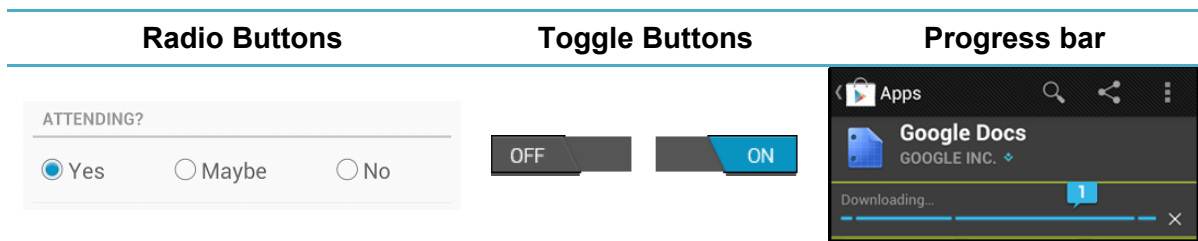
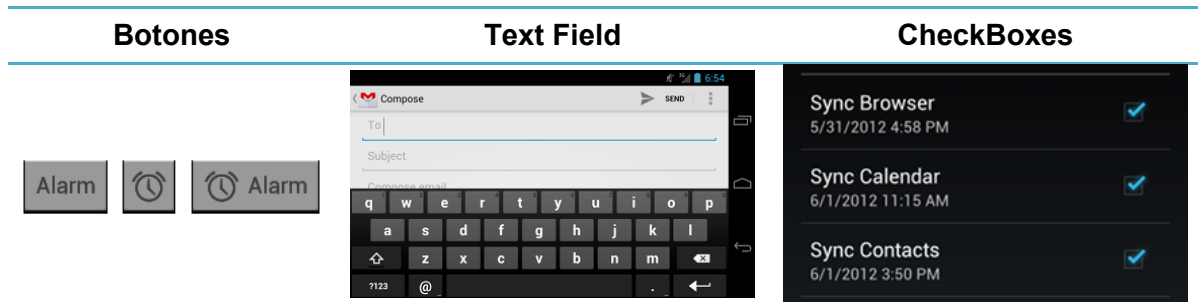
En Android es posible utilizar diferentes animaciones para mejorar visualmente el aspecto y modelo de la aplicación. Se recomienda no abusar de ellas debido a la carga que puede suponer en el dispositivo y la confusión que podría generar en el usuario. Entre las distintas posibilidades existen:

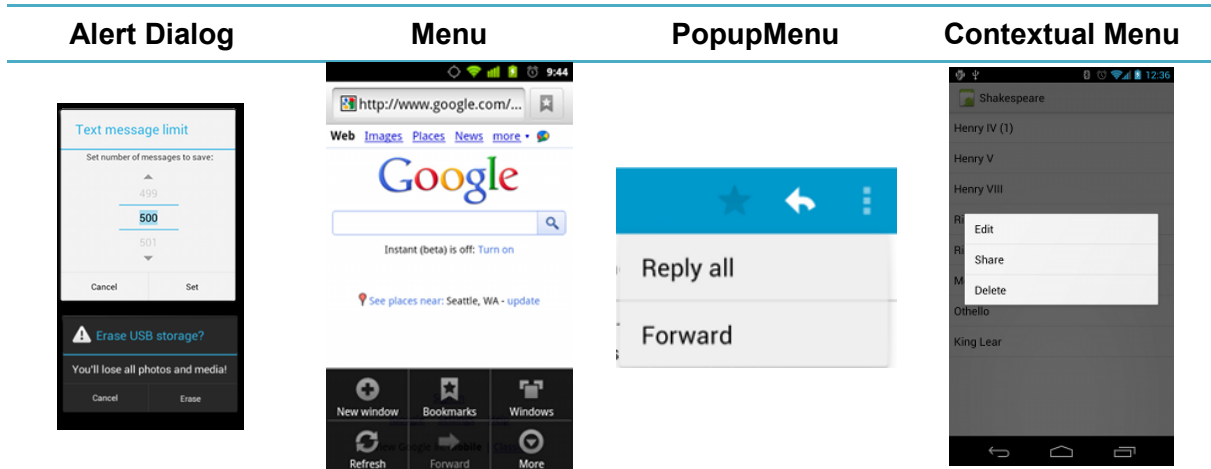
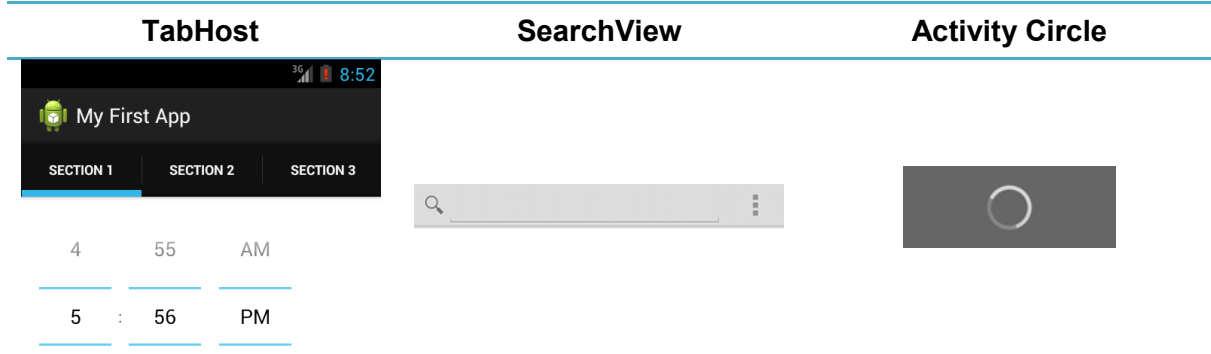
- Realizar un fundido para mostrar una transición gradual de una vista a la siguiente, de tal forma que se solapan en pantalla.
- Utilizar un ViewPager para mostrar diapositivas: Para vistas que muestran contenido variado y cambiante, por ejemplo una selección de diapositivas o las páginas de un libro, se puede utilizar este objeto que muestra cada página deslizándose sobre la anterior.
- Voltear vistas: Para alternar entre dos vistas, se puede optar por una animación que voltea la pantalla y muestra la siguiente haciendo suponer que la siguiente vista se encuentra detrás de la actual.
- Zoom: Es posible realizar zoom de vistas o elementos particulares, por ejemplo para ampliar suavemente las imágenes de una galería o vista de rejilla.
- Cambios en layout: Cada vez que se ejecuta un cambio en la configuración de los elementos de pantalla se puede utilizar una animación a medida para suavizar la apariencia de la operación. Por ejemplo, cuando se añaden o se borran elementos de una lista bajo petición del usuario.

Nótese que por defecto las aplicaciones implementan ciertas animaciones en las operaciones habituales. Por ejemplo al abrir una nueva Activity la transición incluye una animación por defecto sin necesidad de que el desarrollador añada ningún código especial.

### 3.2.7 Controles

Los controles en Android siguen el estilo y funcionalidad habitual de cualquier otro sistema operativo. Por tanto, podemos encontrar los habituales botones, cuadros de texto, checkboxes, toggle buttons, etc.





El control ActionBar merece mención especial dado que Google está solicitando a los desarrolladores que se utilice como parte de los interfaces, al mismo tiempo que se deprecia el antiguo objeto menú. De hecho se incluye por defecto en uno de los temas que se sugieren en los asistentes. ActionBar supone un reemplazo al menú tradicional, que será deprecado a partir de la versión 11 del API de Android.

## 4 Relación entre plataformas

Para establecer las bases en la generación del código automático, se han de refinar y precisar las diferencias y puntos comunes entre las plataformas. Ello servirá para definir el camino que tomará cada acción de generación de código.

### 4.1 Vistas

Salvando las diferencias en los nombres, en ambas plataformas encontramos objetos que definen el contenido de una vista y que suponen un punto común en ambas partes, dado que el grado de similitud es muy elevado. En ambos casos las vistas almacenan el contenido gráfico que se muestra en el interfaz de usuario.

iOS	Android
UIView	View

Table 1 Relación entre clases para modelar vistas iOS y Android

### 4.2 Activities (Android) y View Controllers (iOS)

En Android las Activities han de ser, por definición, auto contenidas y ofrecer una funcionalidad clara y bien definida al usuario. Como ejemplos encontramos Activities para enviar un correo o ver un mapa.

Por el contrario, en iOS se denomina Activities a un concepto totalmente diferente. Sin embargo las Activities de Android pueden encajar o modelarse en iOS mediante los View Controllers. Una vista y su controlador definen una interfaz asociada a una función específica que se muestra al usuario, con el mismo objetivo con el que se define una Activity en Android.

iOS	Android
UIViewController	Activity

Table 2 Relación entre clases para modelar controladores de vista y Activities

Como ejemplo de código se pueden observar las diferencias de generar controladores de vista y Activities en iOS y Android:

iOS	Android
<pre>#import &lt;UIKit/UIKit.h&gt;  @interface VistaTabular1ViewController : UIViewController { } @end  [...]  #import "VistaTabular1ViewController.h" @implementation VistaTabular1ViewController  - (void)viewDidLoad {     [super viewDidLoad];      [self setTitle:@"Vista Tabular 1"]; }  - (void)didReceiveMemoryWarning {     // Releases the view if it doesn't have a <u>superview</u>.     [super didReceiveMemoryWarning];      // Release any cached data, images, etc. that aren't     in use. }  - (void)viewDidUnload {     [super viewDidUnload];     // Release any retained <u>subviews</u> of the main view.     // e.g. self.myOutlet = nil; }  - (void)dealloc {     [super dealloc]; }  @end</pre>	<pre>package com.uned.androidexample;  import android.app.Activity; import android.os.Bundle; import android.view.Menu; import android.widget.ListView;  public class Tabular1Activity extends Activity {      @Override     protected void onCreate(Bundle savedInstanceState) {         super.onCreate(savedInstanceState);         setContentView(R.layout.activity_tabular1);     }      @Override     public boolean onCreateOptionsMenu(Menu menu) {         getMenuInflater().inflate(R.menu.tabular1, menu);         return true;     }      [...]      &lt;RelativeLayout     xmlns:android="http://schemas.android.com/apk/res/android"     xmlns:tools="http://schemas.android.com/tools"     android:layout_width="match_parent"     android:layout_height="match_parent"     android:paddingBottom="@dimen/activity_vertical_margin"     android:paddingLeft="@dimen/activity_horizontal_margin"     android:paddingRight="@dimen/activity_horizontal_margin"     android:paddingTop="@dimen/activity_vertical_margin"     tools:context=".Tabular1Activity" &gt;      &lt;/RelativeLayout&gt;</pre>

Table 3 Código generado para un ViewController y una Activity

Nótese que en ambos casos se implementan más métodos en cada una de las clases, además del fichero XML que se necesita en Android para generar el layout. Las diferencias completas se pueden ver en los ejemplos de prototipo descritos más adelante.

### 4.3 Vistas especiales

En ambas plataformas se definen ciertas vistas comunes para funciones muy habituales, como visualización de tablas, mapas, etc. Todas las vistas analizadas en este apartado tienen su correspondencia en ambas plataformas, aunque difieran en ciertos detalles de implementación. Para la mayoría de ellas, incluso el nombre es el mismo en ambos casos. La siguiente tabla muestra dicha equivalencia.

Nótese como en algunos casos no existe una equivalencia directa y por tanto se ha tratado de localizar el componente más aproximado en cada plataforma.

iOS	Android
UITableViewController	ListView
UICollectionViewController	GridView
UIImageView	ImageView
MkMapView	MapView
UIScrollView	ScrollView
UITextView	TextView
UIWebView	WebView

Table 4 Relación entre clases para modelar vistas especiales iOS y Android

La siguiente tabla muestra la generación de código para una lista (Table en iOS y List en Android).

iOS	Android
<pre>[...]  UITableView *myTableView = [[UITableView alloc] initWithFrame:CGRectMake(10,10,280,200)]; myTableView.delegate = self; myTableView.dataSource = self; [self.view addSubview:myTableView];  [...]</pre>	<pre>[...]  ListView listView = (ListView) findViewById(R.id.listView1); listView.setAdapter(new ListView1ViewAdapter(this));  [...]</pre>

---

```

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section { return 2; }

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:cellIdentifier];
    if (cell == nil)
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:cellIdentifier];
    if (indexPath.row == 0)
        cell.textLabel.text=@"Elemento de Lista 1";
    if (indexPath.row == 1)
        cell.textLabel.text=@"Elemento de Lista 2";
    return cell;
}

```

```

public int getCount() { return 2; }

public View getView(int position, View convertView, ViewGroup
parent) {
    TextView textView;
    if (convertView == null) {
        textView = new TextView(mContext);
        if (position == 1)
            textView.setText("Elemento de Lista 1");
        if (position == 2)
            textView.setText("Elemento de Lista 2");
    } else {
        textView = (TextView) convertView;
    }
    return textView;
}

```

---

Table 5 Código necesario para una implementar una lista en iOS y Android

## 4.4 Navegación

Para la navegación entre diferentes vistas en iOS contamos con los ‘Container View Controllers’. Los más comunes son ‘Navigation Controller’ y ‘Tab View Controller’.

En Android la navegación se hace a través de diferentes ‘Activities’: una vista puede llamar a la activación de otra ‘Activity’ directamente, incluso si es de otra aplicación. La ‘Activity’ receptora ha de implementar un ‘Intent filter’ para recibir ese tipo de mensajes, activarse y pasar a tener el foco de usuario.

La equivalencia de ‘Navigation Controller’ de iOS (navegación entre vistas, paso a paso, con vuelta a atrás mediante pulsadores en la esquina superior izquierda de la pantalla) se consigue en Android de forma nativa por el sistema operativo. Cuando una vista en Android llama a otra ‘Activity’, la primera pasa al ‘back stack’ donde se guarda en una estructura tipo ‘Last In, First Out’. Mediante el botón retroceso de los dispositivos Android (botón físico disponible en todos los teléfono y tablets) se puede ‘deshacer’ el camino recorrido de vista en vista. No es necesario que el programador implemente un controlador especial y ni siquiera es necesario mostrar una pestaña específica como se requiere en iOS.

Por otra parte la navegación entre distintas pestañas se consigue en iOS mediante el ‘Tab View Controller’ (UITabBarController). Dicho controlador permite saltar a cada una de las vistas predefinidas para cada pestaña. En Android existen controles que facilitan esta



tarea: TabWidget y TabHost. Basta con incluirlo en la vista padre que estamos usando y automáticamente podremos añadir elementos a cada una de las vistas hijas.

iOS	Android
NavigationController	De forma nativa y automática en Android
TabViewController	Mediante el control TabWidget

Table 6 Relación entre navegación de iOS y Android

Como ejemplo se puede observar el código generado para obtener navegación lineal en las dos plataformas:

iOS	Android
<pre>- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {      UINavigationController* navController = [[UINavigationController alloc] init];      VistaPrincipalViewController* vPVC = [[VistaPrincipalViewController alloc] init];      [navController pushViewController:vPVC animated:NO];     [self.window addSubview: navController.view];      [self.window makeKeyAndVisible];     return YES; } [...]</pre>	<pre>Intent intent = new Intent(this, SecundariaActivity .class); startActivity(intent);</pre>
<pre>VistaSecundariaViewController *vSVC = [[VistaSecundariaViewController alloc] init]; [[self navigationController] pushViewController:vSVC animated:YES];</pre>	

Table 7 Código generado para navegar linealmente en iOS y Android

## 4.5 Controles

La mayoría de los controles tienen un equivalente en ambas plataformas cuya función es prácticamente idéntica. Aun así, hay detalles de menor nivel que pueden complicar la generación automática de código para ciertas características muy específicas. Una operación sencilla en una plataforma puede ser muy compleja en otra, por ejemplo implementar un ‘Segmented Control’ de iOS en Android es relativamente complicado.

Existen algunas alternativas ya realizadas, pero en el caso de este trabajo sustituiremos la implementación de dicho control por el más similar en la plataforma destino.

<b>iOS</b>	<b>Android</b>
UIStatusBar (solo para pequeñas modificaciones)	StatusBar (solo para pequeñas modificaciones)
UIToolBar	No disponible. Usar Menu (deprecado) o bien ActionBar.
UIActivityViewController	No disponible para programación. Se configura en ajustes del dispositivo y el sistema operativo lo muestra de forma automática
UIPageViewController	No disponible. Hay implementaciones OpenSource
UIPopover (solo Ipad)	Contextual Menu
UISplitViewController	Uso conjunto de Vistas, Fragments y Layouts
UIAlertView	AlertDialog
UIActionSheet	Contextual Menu / Popup Menu
Modal Views (exceptuando alertas)	Se ha de conseguir manualmente, incluyendo el efecto de transparencia del fondo.
UIActivityIndicatorView	Activity Circle o Activity Bar
UIProgressView	ProgressBar
UIRefreshView	No existe como tal. Se ha de implementar con un icono + botón.
UISlider	Seek Bar o Sliders
UIStepper	No existe. Se puede tomar ZoomButtonsController y cambiar el funcionamiento manteniendo el aspecto.
UISwitch	Toggle Buttons
UITextField	TextField
UIButton	Button
UISearchBar	SearchView
UISegmentedControl	No disponible.
UIPickerView	Spinner (DatePicker/TimePicker para fechas)
UIPageControl	No disponible. Se puede conseguir con implementaciones alternativas.

iOS	Android
UILabel	Text
Network Activity Indicator	No disponible. Se sugieren alternativas como ProgressBar, Activity Bar o Activity Circle.

Table 8 Relación entre controles de iOS y Android

Como ejemplo de código de cada plataforma, se muestra el caso de generación de una barra deslizable. Se puede observar como en Android sólo es necesario modificar el XML (para casos sencillos):

iOS	Android
<pre>UISlider *slider = [[UISlider alloc] initWithFrame:CGRectMake(20,250, 280, 40)]; slider.minimumValue = 0.0; slider.maximumValue = 100.0; slider.continuous = YES; slider.value = 25.0; [self.view addSubview:slider];</pre>	<pre>&lt;SeekBar     android:id="@+id/slider1"     android:layout_width="match_parent"     android:layout_height="wrap_content"     android:layout_alignParentLeft="true"     android:layout_marginTop="55dp"     android:max="100"     android:progress="0" /&gt;</pre>

Table 9 Código generado para modelar una barra deslizable

## 5 Consideraciones de diseño generales

Este trabajo analiza la viabilidad de una solución completa de generación de código multiplataforma. El estudio de la solución completa de todas las capas a nivel de programación incluye innumerables detalles concretos y específicos. Por ello, para poder abordar esta solución es necesario restringirla a un determinado dominio, en este caso la capa de usuario. Se trata de analizar la viabilidad de la propuesta y su futura escalabilidad al resto de capas y plataformas.

Por tanto, es necesario descomponer el sistema de generación de código en múltiples áreas y en cada una de ellas se han de realizar ciertas consideraciones que ayuden a dividir y organizar los pasos a dar.

Centrándonos en el contexto de generación de interfaces de usuario, es necesario tener en cuenta estas consideraciones, que podrán ser tratadas posterior e individualmente.

El generador asume que usará controles, componentes y elementos nativos de cada plataforma y por tanto no se utilizará para desarrollar nuevos componentes gráficos o controles.

Los componentes nativos serán usados en cada plataforma de forma natural sin que sean alterados ni modificados de alguna forma por el generador. El propósito es usar únicamente éstos componentes que son proporcionados de forma nativa, siguiendo las indicaciones dadas en la documentación de referencia.

El código generado se podrá compilar directamente y será utilizable para construir la aplicación final. Esto no impide que dicho código pueda ser utilizado para empaquetarse como un componente más, pero de hacerlo será de forma manual y con intervención del desarrollador.

Las plataformas móviles suelen utilizar múltiples animaciones para mostrar transiciones entre vistas o pantallas, movimiento de botones, cierre de ventanas, etc. Todas estas animaciones serán utilizadas por defecto si la plataforma provee dicha funcionalidad de forma automática, aprovechando las animaciones que se ofrecen de base.

Aunque en ambas plataformas existe la posibilidad de utilizar OpenGL, esto supone un paradigma completamente distinto, con unas APIs diferentes y que supuestamente serán comunes en ambas plataformas. No obstante no es el objetivo de este trabajo dado que supone una aproximación distinta de la actual.

Para poder llegar al objetivo de investigación del trabajo se cubrirán inicialmente una serie de controles, elementos y componentes a generar, tratando de buscar la generalidad sin entrar en los detalles específicos de cada plataforma. Se trata de cubrir los más esenciales y estudiar la viabilidad sin perder generalidad.

## 6 HTML5 como DSL

El generador de código multiplataforma necesita inevitablemente de un medio a través del cual se especifiquen las intenciones de la aplicación a generar. Este medio puede tomar diversas formas, por ejemplo puede ser una herramienta gráfica que permita la creación de bocetos que sean transformables a código específico de cada plataforma.

En este caso se ha optado por utilizar un Domain Specific Language (DSL) por ser más claro y preciso teniendo en cuenta el propósito de investigación de este trabajo. La utilización de un DSL permite definir de forma exacta qué es exactamente lo que se desea, gracias a la utilización de primitivas y composición de objetos y elementos bien definidos funcionalmente.

La utilización de una herramienta gráfica sería más visual e intuitiva inicialmente pero dada la distinta naturaleza gráfica de las plataformas en las que se genera código, el resultado final distaría notoriamente del boceto origen. Por ejemplo, en cuanto a navegación lineal, iOS necesita de un botón superior para volver atrás cuando se avanza de vista en vista, mientras que Android incluye esta funcionalidad de forma nativa y no es necesario añadir ningún control en pantalla (existe un botón físico para retroceder).

La siguiente figura muestra la diferencia entre usar el mismo componente en las diferentes plataformas. Aunque se pueden realizar añadidos al código para obtener una visualización lo más similar posible, la utilización de forma natural de los componentes nativos puede mostrar visualizaciones completamente diferentes:



Figura 14 Selector en Android (en estado normal y desplegado) e iOS

Por ello se prefiere la utilización de un DSL y se ha optado por HTML5 como lenguaje específico. Aunque se podría haber desarrollado un nuevo lenguaje para este caso,

HTML5 es suficientemente flexible para implementar las necesidades actuales de este trabajo y las posibilidades futuras. Los motivos que sugieren la utilización de HTML5 son:

- Se trata de un lenguaje estandarizado.
- Es altamente flexible, permitiendo modelar prácticamente cualquier idea.
- Es ampliable, incluso se podría completar con etiquetas propias.
- Aunque en esta primera versión no se obtenga todo el partido de HTML5, en futuras revisiones de este trabajo se podrá aprovechar todo su potencial (por ejemplo por el uso de multimedia, geolocalización o sensores).
- Es muy popular y está ampliamente extendido.
- Evolucionará con la industria.

No obstante, el lenguaje es tan amplio que deberá limitarse exclusivamente a los aspectos definidos y contemplados en este documento. Cualquier otro uso no reflejado podría dar lugar a resultados inesperados o la imposibilidad de generar código.

En los siguientes apartados se revisan los puntos cubiertos por este trabajo en cuanto a utilización de HTML5. Para simplificar y sin perder generalidad se han cubierto inicialmente los conceptos más relevantes que permitirán generar código y que corresponden a contenedores, navegación, vistas y controles. Para entender el proceso de generación es preciso definir los conceptos que se usarán en el DSL. Adicionalmente se analiza la utilización de un lenguaje de programación para poder desarrollar el código necesario en la gestión de eventos o para implementar la funcionalidad que se ocultaría tras el desarrollo de un interfaz gráfico, por ejemplo algoritmos u organización de datos.

## **6.1 Contenedores**

Denominaremos contenedores a aquellas entidades que servirán para albergar contenidos presentables al usuario, sean tablas, formularios, gráficos, mapas, etc. El objetivo es definir una entidad que sirva para mantener todos aquellos elementos que forman parte de contenido visible y que permita interactuar con usuario. La idea es que el contenedor albergue diversos elementos simultáneamente, ofreciendo una determinada funcionalidad intuitiva y fácilmente usable.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Contenedor A</title>
  </head>
  <body>
    ...
  </body>
</html>
```

Para modelar esta entidad se tomará un fichero HTML en su totalidad como contenedor. Es decir, un fichero de texto compuesto de los tags <html>, <head> y <body> será denominado contenedor y servirá para albergar un grupo de elementos de tal forma que constituyan una funcionalidad que permita interactuar al usuario.

Realmente será el contenido de la etiqueta <body> lo que determinará realmente los elementos a mostrar.

En una aplicación típica, será normal encontrar varios contenedores que el usuario visualizará a medida que navegue o realice determinadas acciones.

En evoluciones futuras del trabajo se puede proponer una estructura jerárquica de contenedores, aunque por ahora se asume que un contenedor no podrá incluir otros para componer vistas, aunque sí para realizar navegación.

## 6.2 Navegación

Dado que pueden existir diversos contenedores por aplicación, se debe definir también la forma de poder navegar entre ellos. Se asume que existen dos posibilidades de navegación en el código a generar:

- Navegación lineal: Se produce cuando el usuario avanza de un contenedor a otro por acciones iniciadas por él mismo, por ejemplo pulsar un botón, un enlace o abrir información sobre un contacto en una lista.

Los contenedores visitados previamente se apilan en una estructura de tipo pila de tal forma que el usuario es capaz de retroceder a los anteriores en orden inverso.



- Navegación tabular: Se produce cuando existe un control de tipo pestaña (Tab). Este control permite al usuario pasar de un contenedor a otro pulsando en los botones definidos en el control Tab. La navegación no sigue ningún patrón ni orden salvo el que aplique el propio usuario durante el uso. Los contenedores visitados no se guardan ni se apilan como en el caso anterior y por tanto no existe historia sobre lo visitado previamente.

Para implementar la navegación lineal se usará la etiqueta <a> de html. Cada vez que se desee ofrecer al usuario la posibilidad de navegar a otro contenedor se deberá utilizar un hipervínculo que podría además llevar parámetros en la url para facilitar el paso de información. La utilización de hipervínculos se puede llevar a cabo en etiquetas, botones, elementos de listas, etc. La pulsación de un elemento asociado a un hipervínculo conllevará la navegación al contenedor indicado, apilando en memoria el actual para su posterior recuperación.

```
<a href="ContenedorB.html?Contact=Isabel.html">
  Detalles del contacto: Isabel
</a>
```

Para implementar la navegación tabular es necesario definir un contenedor que incluya un tag de nueva definición denominado <tabNavigation>. Este tag incluirá referencias <a> a tantos contenedores como se desee mostrar. El código generará un componente Tab y en cada pestaña ofrecerá al usuario la posibilidad de navegar a cada uno de los contenedores indicados.

```
<body>
  <tabNavigation>
    <a href="ListaFotos.html">Mis fotos</a>
    <a href="MapaFotos.html">Mapa de fotos</a>
    <a href="TiendasFotografia.html">Tiendas</a>
  </tabNavigation>
</body>
```

## 6.3 Vistas

Denominaremos vistas a los recursos que se utilizan para colocar elementos (dentro de un contenedor) y que en algunos casos incluirán funcionalidad determinada, generalmente más compleja que la de un control. Este trabajo asume que existirán vistas de tipo formulario, tabla, lista, foto y mapa: por ejemplo, el mecanismo para colocar objetos en forma de grid o tabla se llevará a cabo mediante la etiqueta <table> mientras que si se desea colocar objetos en forma de lista se usará <ul>. En muchos casos un contenedor estará asociado a una única vista, pero en muchos otros se combinarán varias vistas para ofrecer múltiples funcionalidades.

Nótese que se admite composición recursiva (hasta un determinado punto) para flexibilizar la composición. Por ejemplo una vista de formulario puede contener una vista de lista en la parte superior del contenedor y continuar con una vista de grid en la parte inferior.

### 6.3.1 Vista Formulario o simple

La vista simple o de formulario es la que se utiliza por defecto y no es necesario configurar. En este modo, el desarrollador incluiría controles o elementos que se mostrarían secuencialmente (o de acuerdo a algún layout configurable que se podría incluir en posteriores revisiones de este trabajo).

```
<body>
  <label>Click en la sgte. línea para ver tus fotos</label>
  <a href="ListaFotos.html">Mis fotos</a>
</body>
```

### 6.3.2 Vista de tabla

En la vista de tabla los elementos se organizan en un grid de elementos que forman filas y columnas. Es más adecuado para mostrar imágenes o iconos. Para crear esta vista se usará la etiqueta <table> y las asociadas <tr>, <td> para definir filas y columnas.

```

<body>
  <table>
    <tr>
      <td>
        <label>1,1</label>
      </td>
      <td>
        <label>1,2</label>
      </td>
    </tr>
    <tr>
      <td>
        <label>2,1</label>
      </td>
      <td>
        <label>2,2</label>
      </td>
    </tr>
  </table>
</body>

```

Nótese que estas etiquetas definen una distribución deseada de los objetos en filas y columnas, pero el dispositivo puede modificar esta distribución en función del tamaño de los objetos, de la pantalla, de la orientación del dispositivo, etc. Por ello quizás se podría considerar la utilización únicamente de etiquetas <table> y <td> para definir los elementos, omitiendo la especificación de filas mediante <tr>.

### 6.3.3 Vista de lista

Para modelar una lista se utilizarán las etiquetas <ul> y <li> para cada elemento de la misma. La lista se transformará en una lista desplazable de elementos con scroll en cada dispositivo. En cada elemento de la lista se podrán insertar textos o imágenes según se desee.

```

<body>
  <ul>
    <li>Alemania</li>
    <li>España</li>
  </ul>
</body>

```

### 6.3.4 Vista de imagen

Este elemento permite insertar imágenes en otros elementos, contenedores o vistas. Serviría para incluir imágenes en una tabla, una lista o para mostrarlas a pantalla completa. Posteriormente se podrían agregar parámetros para ubicar la imagen, ampliarla, rotarla, etc.

Para hacerla efectiva se debe usar la etiqueta <img>

```
<body>  
  Alemania</img>  
</body>
```

### 6.3.5 Vista de mapa

Finalmente, para conseguir mostrar un mapa se requiere la etiqueta <map> que es definida explícitamente para este propósito y que no pertenece al estándar. Aunque HTML es capaz de mostrar mapas gracias a las diversas APIs existentes en la red, en este caso es necesario simplificar este proceso y por ello se ha buscado un sistema más sencillo:

```
<body>  
  <map org="40.1324335, 12.3243">...</map>  
</body>
```

Entre la etiqueta de apertura y de cierre se podrían añadir puntos en el mapa para señalar determinadas zonas, puntos de interés, dibujar rutas, etc. Todo ello serían argumentos que se podrían añadir a esta especificación, basándose en los APIs ofrecidos por Google Maps o cualquier otro proveedor de recursos de geolocalización.

## 6.4 Controles

Por generalidad se ha seleccionado un conjunto de controles que permiten demostrar la generación de código en las distintas plataformas. En este trabajo se consideran etiquetas, campos de texto, botones, selectores, interruptores y barras de deslizamiento o sliders. Para todos ellos se podrían considerar atributos de visualización, tales como color

principal, color de fondo, tamaño, y posición. Este último requeriría un análisis posterior para homogeneizar y abstraer los diferentes layouts de cada plataforma. Los atributos se introducirían como parte de las etiquetas de HTML:

```
<label fgColor:blue bgColor:black posX:50% posY:20%>Nombre</label>
```

### 6.4.1 Etiqueta

Las etiquetas constituyen las líneas de texto individuales que sirven para mostrar información estática y no alterable por el usuario. Se utilizan por ejemplo para indicar información acerca de los controles a los que van anexados. Para definir las se usará la etiqueta <label>:

```
<body>
  <label>Nombre</label>
</body>
```

### 6.4.2 Campo de texto

El campo de texto es otro de los elementos indispensables en cualquier formulario. Mediante el campo de texto se permite la introducción de información textual. Para utilizarlo basta con incluir una referencia a <input>:

```
<body>
  <input name="name" type="text">
</body>
```

### 6.4.3 Botón

Gracias a los botones se puede enviar información, navegar entre diferentes contenedores o ejecutar un determinado código. Se usará la etiqueta <button> para este propósito:

```
<body>
  <button onclick="alert('Hola')">Pulsa aquí</button>
  <button onclick="top.location='NextPage.html'">Siguiete página
</button>
</body>
```

Nótese que para la ejecución de código se utiliza javascript como lenguaje por defecto en HTML5. En este generador será necesario interpretarlo para poder tener un mínimo conjunto de instrucciones operativas que permitan ejecutar cálculos y operaciones básicas, como las mostradas en el ejemplo.

#### 6.4.4 Selector

Como selector se entiende aquel control que permite elegir un elemento de una lista de posibles alternativas. Se conocen como pickers o combo box en su denominación más tradicional. En general en entornos móviles cuando se selecciona uno de estos controles para cambiar la alternativa, aparece una ventana modal ocupando parte de la pantalla para facilitar el desplazamiento y la elección de la alternativa deseada.

```
<body>
  <select name="carSelector">
    <option value="Mercedes">Mercedes</option>
    <option value="Audi">Audi</option>
  </select>
</body>
```

Más adelante se podrían desarrollar otros selectores para fechas, horas u otros tipos de datos genéricos.

#### 6.4.5 Interruptor

Los interruptores son controles con dos estados que permiten activar o desactivar determinadas características. Funcionalmente son equivalentes a los checkboxes tradicionales.

```
<body>
  <input type="checkbox" name="checkbox1">Marque la casilla para
  activar la función</input>
</body>
```

## 6.4.6 Barras deslizantes o sliders

Estas barras se utilizan para modificar un valor de un rango determinado moviendo un puntero a lo largo de una barra, generalmente creciente de izquierda a derecha.

```
<body>  
  <input type="range" name="brightness" min="0" max="100"/>  
</body>
```

## 6.4.7 Lenguaje

Todo lo visto hasta ahora está orientado a la generación de un interfaz gráfico partiendo de HTML5 con objetivo de obtener código específico de plataforma Android o iOS. Una interfaz gráfica necesita estar complementada con mecanismos que permitan indicar qué hacer cuando un usuario presiona un botón, cambia un valor o navega a la siguiente ventana. Además, determinadas aplicaciones desarrollarán algoritmos, cálculos u operaciones que darán el sentido final a la aplicación. Para llevar esto a cabo es necesario utilizar un lenguaje de programación y la recomendación más apropiada es Javascript, teniendo en cuenta la base de HTML5. Javascript y HTML5 están muy relacionados y suelen ir parejos en el desarrollo de aplicaciones web. De hecho evolucionan también conjuntamente y ambos están estandarizados.

La transformación de Javascript en el lenguaje propio de cada plataforma es otro ejercicio que requiere un estudio aparte. En este caso se trataría de transformar Javascript a Objective-C para iOS o a Java para Android. Dicha investigación podría llevarse a cabo más adelante y por el momento solo se asumirán sentencias básicas para permitir por ejemplo el salto de una vista a la siguiente o para mostrar una alerta.

## 7 Generación de código

El propósito final del generador es proporcionar código que se puede compilar y ejecutar en cada una de las plataformas soportadas partiendo de la especificación en HTML5. Para ello, se analizará la relación entre las sentencias vistas previamente en HTML y su impacto en el código a generar de cada una de las plataformas.

### 7.1 HTML5 a iOS

La realización de aplicaciones en iOS se lleva a cabo en el entorno xCode y se utiliza el framework Cocoa. Generalmente las aplicaciones se realizan con ayuda de un programa adicional denominado “interface builder” que vuelca el resultado a ficheros .xib. La utilización de este constructor gráfico resulta útil para visualizar instantáneamente el aspecto que finalmente tendrá la aplicación, además de ayudar enormemente en el diseño y sobre todo en la rapidez con la que se crean nuevos interfaces.

Sin embargo, para este trabajo se prescindirá de este constructor dado que todos los elementos usados han de crearse programáticamente. Esta forma es mucho más compleja y menos intuitiva, pero necesaria para la aproximación de generación automática tomada en este trabajo.

De forma general se crearán las siguientes clases necesarias en cada despliegue de aplicación iOS:

- Una clase UIWindow que irá asociada a una clase UIScreen que están accesibles pero no se declaran explícitamente, ya que la plantilla del asistente utilizada como base las genera automáticamente y permanecen ocultas como parte del interfaz.
- Una clase de tipo UIApplicationDelegate que se encarga de gestionar los principales eventos asociados a una aplicación, por ejemplo cuando ha terminado de inicializarse, cuando la memoria es baja o cuando la aplicación está a punto de terminar. En este caso se utilizará esta clase para inicializar el controlador de vista principal o raíz (rootViewController) y hacer la ventana visible.



El código perteneciente a la clase delegada y que inicializa el controlador de vista principal se localiza en el método “didFinishLaunchingWithOptions”:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    RootViewController *rvc = [[RootViewController alloc] init];
    self.window.rootViewController = rvc;

    [self.window makeKeyAndVisible];

    [rvc release];

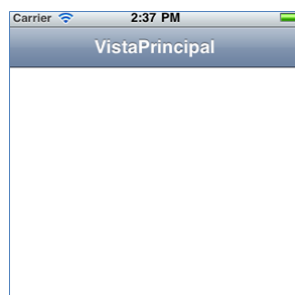
    return YES;
}
```

A partir de este punto el resto de clases generadas dependerán de lo especificado por el usuario. Las relaciones entre la especificación del usuario y el código generado se detallan en los siguientes apartados. Cada uno de los capítulos siguientes determinará qué código es necesario generar por cada elemento definido en HTML5.

### 7.1.1 Contenedores

Cada contenedor definido por el usuario está pensado para contener una pantalla de información que realiza una serie de tareas determinadas, tales como mostrar o recabar información. Se permite saltar de un contenedor a otro, para por ejemplo mostrar información más detallada.

En iOS esto encaja con el modelo propuesto por UIViewController. Un controlador de vista permite crear elementos gráficos, anidar vistas o incluso otros controladores de vista, por ejemplo para gestionar un componente de tabla deslizable.



**Figura 15 Contenedor en iOS**

Por tanto, la equivalencia entre un contenedor definido en HTML5, es decir, un fichero con extensión html será directamente una clase UIViewController.

```
#import <UIKit/UIKit.h>
@interface VistaPrincipalViewController : UIViewController {
}
End
[...]

#import "VistaPrincipalViewController.h"
@implementation VistaPrincipalViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    [self setTitle:@"VistaPrincipal"];
}

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc. that aren't in use.
}

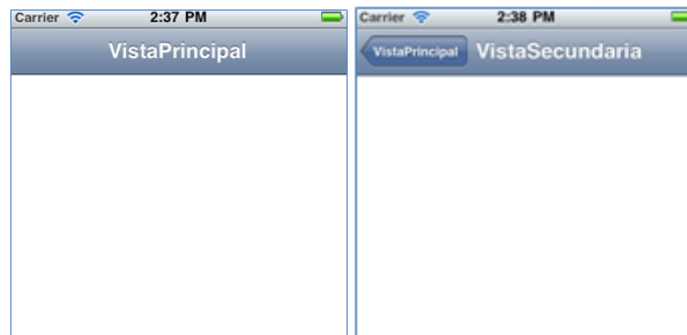
- (void)viewDidUnload {
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)dealloc {
    [super dealloc];
}
@end
```

### 7.1.2 Navegación

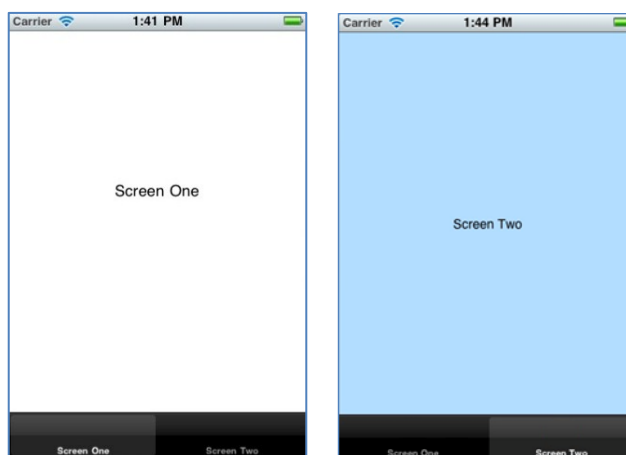
La afirmación anterior requiere matizarse, dependiendo de la navegación existente en el contenedor.

Si el documento o contenedor HTML contiene saltos a otros contenedores y por tanto se trata de navegación lineal, entonces la clase a generar deberá ser UINavigationController, también hija de UIViewController. Con esto se permite que se puedan apilar los controladores de vista que el usuario recorre.



**Figura 16 Navegación lineal en iOS**

Si el contenedor HTML contiene un elemento de tipo `<tabNavigation>`, entonces se requiere navegación tabular y por tanto la clase a generar será de tipo `UITabBarController` (heredará de ella).



**Figura 17 Navegación tabular en iOS**

No obstante, existe una forma alternativa que simplifica la generación de código y no requiere herencia. Se trata de instanciar las clases `UINavigationController` o `UITabBarController` en la inicialización y organizar programáticamente sin necesidad de herencia el layout del controlador en cuestión. Esta será la forma elegida para este trabajo.

Por ejemplo, para ofrecer navegación lineal en un controlador de vista se llamará al controlador con el recubrimiento de un UINavigationController tal y como muestra el ejemplo:

```
UINavigationController* navController = [[UINavigationController alloc] init];
VistaPrincipalViewController* vPVC = [[VistaPrincipalViewController alloc]
init];
[navController pushViewController:vPVC animated:NO];
```

Cuando se desee llamar al siguiente controlador de vista, se tendrá que realizar de la siguiente forma:

```
VistaSecundariaViewController *vSVC = [[VistaSecundariaViewController alloc]
init];
[[self navigationController] pushViewController:vSVC animated:YES];
```

Para retroceder al último controlador apilado ejecutaríamos este comando:

```
[self.navigationController popViewControllerAnimated:YES];
```

En el caso de navegación tabular será necesario inicializar los controladores de vista recubriéndolos con la clase UITabBarController (nótese que también se utiliza auxiliarmente la clase UINavigationController):

```
UITabBarController *tabBars = [[UITabBarController alloc] init];
NSMutableArray *localViewControllersArray = [[NSMutableArray alloc]
initWithCapacity:2];

VistaTabular1ViewController *vT1VC = [[VistaTabular1ViewController alloc] init];
UINavigationController *myController1 = [[UINavigationController alloc]
initWithRootViewController:vT1VC];
myController1.title = @"Vista 1";

VistaTabular2ViewController *vT2VC = [[VistaTabular2ViewController alloc] init];
UINavigationController *myController2 = [[UINavigationController alloc]
initWithRootViewController:vT2VC];
myController2.title = @"Vista 2";

[localViewControllersArray addObject:myController1];
[localViewControllersArray addObject:myController2];

tabBars.viewControllers = localViewControllersArray;

[self.view addSubview:tabBars.view];
```

El paso de un controlador a otro se realiza automáticamente cuando el usuario pulsa en cada una de las pestañas, por tanto no es necesario añadir más código para gestionarlo.

### 7.1.3 Vistas

En este apartado se tratan las vistas, que se pueden considerar pseudo controles que implementan una determinada funcionalidad. Generalmente es más compleja que la que puede implementar un simple control., por ejemplo una tabla o una lista. En iOS este tipo de elementos son considerados también como vistas, al tener y ofrecer una funcionalidad más compleja.

#### 7.1.3.1 Vista de formulario

Las vistas de formulario son las más sencillas de implementar dado que son las que por defecto asume el sistema. Es decir, cuando se añade un elemento a esta vista significa que el elemento se añade al controlador de vista actual, con los atributos necesarios. Se implementaría con el controlador de vista que se esté utilizando en cada momento por lo que no es necesario añadir o generar código especial para tratarlo. Basta con añadir el elemento deseado en el método `viewDidLoad`:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UISlider *slider = [[UISlider alloc] initWithFrame:CGRectMake(0,0, 280, 40)];
    [self.view addSubview:slider];
}
```

#### 7.1.3.2 Vistas de tabla

Las tablas en iOS se crean con el componente `UICollectionView` que forma una rejilla o grid en la que se pueden encajar elementos. Los contenidos se dividen en secciones y cada sección contiene un número determinado de elementos que se configuran para cada vista.



**Figura 18 Ejemplo de tabla (UICollectionView) en iOS**

Una posibilidad es crear una clase que herede de `UICollectionViewController` que represente el objeto colección que se va a instanciar, en dicho objeto añadiremos los métodos que gestionan la colección. Sin embargo esto no es imprescindible (aunque sí recomendable si existen muchos elementos en una misma vista). Para simplificar en este trabajo se usará el objeto `UICollectionView` que se instanciará directamente en el controlador de vista en el que se desee implementar el elemento.

```
UICollectionView *collectionView = [[UICollectionView alloc]
    initWithFrame:CGRectMake(0,0,280,200)];
collectionView.delegate = self;
collectionView.dataSource = self;
[self.view addSubview: collectionView];
```

Es necesario implementar los siguientes métodos (que en este caso estarán en el propio controlador de vista que contiene el elemento), en donde se especifican el número de secciones y los elementos de cada sección. Cada colección se estructura en secciones y cada sección puede contener un número determinado de elementos. Nótese que el código incluye más elementos y parámetros que habrán de ser tratados más adelante, por ejemplo respecto al tamaño, color y colocación de los controles.

```
-(NSInteger)numberOfSectionsInCollectionView:(UICollectionView *)collectionView
{
    return 1;
}

-(NSInteger)collectionView:(UICollectionView *)view
    numberOfItemsInSection:(NSInteger)section {
    return 4;
}

-(UICollectionViewCell *)collectionView:(UICollectionView *)cv
    cellForItemAtIndexPath:(NSIndexPath *)indexPath; {
```

```

Cell *cell = [cv dequeueReusableCellWithIdentifier:@"MY_CELL"
              forIndexPath:indexPath];
cell.label.textAlignment = NSTextAlignmentCenter;
cell.label.textColor = [UIColor blackColor];
cell.label.font = [UIFont boldSystemFontOfSize:35.0];
cell.label.backgroundColor = [UIColor whiteColor];

if (indexPath.item == 1)
    cell.label.text = [@"1,1"];
if (indexPath.item == 2)
    cell.label.text = [@"1,2"];
if (indexPath.item == 3)
    cell.label.text = [@"2,1"];
if (indexPath.item == 4)
    cell.label.text = [@"2,2"];
return cell;
}

```

### 7.1.3.3 Vista de lista

Las vistas de lista en iOS se corresponden con tablas de desplazamiento que están modeladas por el objeto `UITableViewController` o también mediante el objeto `UITableView`.

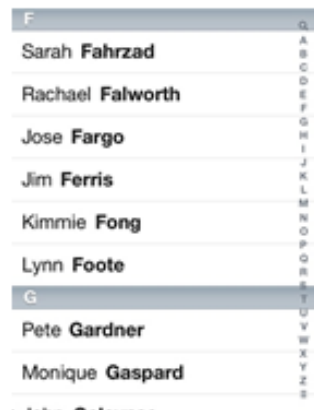


Figura 19 Ejemplo de lista en iOS (UITableView)

El funcionamiento es muy similar al caso anterior. Se puede crear una clase nueva que herede de `UITableViewController` o bien seguir la aproximación anterior:

```

UITableView *myTableView = [[UITableView alloc]
initWithFrame:CGRectMake(10,10,280,200)];
myTableView.delegate = self;
myTableView.dataSource = self;
[self.view addSubview:myTableView];

```

Se implementan métodos similares a los ya vistos para definir el número de elementos y secciones. Cada celda se incluye en una sección que forma parte del controlador, el cual puede incluir varias secciones. La diferencia radica en el nombrado de los métodos, que se sustituyen por `numberOfSectionsInTableView`, `numberOfRowsInSection`, y `cellForRowAtIndexPath`:

```
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    return 2;
}
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:cellIdentifier];
    if (cell == nil)
        cell = [[UITableViewCell alloc]
            initWithStyle:UITableViewCellStyleSubtitle
            reuseIdentifier:cellIdentifier];
    if (indexPath.row == 0)
        cell.textLabel.text=@"Elemento de Lista 1";
    if (indexPath.row == 1)
        cell.textLabel.text=@"Elemento de Lista 2";
    return cell;
}
```

#### 7.1.3.4 Vista de imagen

Para mostrar fotos en las vistas de iOS solo es necesario instanciar un componente de tipo `UIImageView` y configurarlo de forma muy sencilla.

Las siguientes líneas muestran el código a generar para visualizar una imagen:

```
UIImageView* myImage = [[UIImageView alloc]
    initWithFrame:CGRectMake(0,0,280,200)];
myImage.image = [UIImage imageNamed:@"germanFlag.gif"];
myImage.contentMode = UIViewContentModeCenter;
[codeView addSubview:myImage];
```



Aunque iOS clasifica las imágenes como vistas, su uso práctico hace pensar que pudieran ser consideradas simples controles.



Figura 20 Ejemplo de imagen en iOS (UIImageView)

#### 7.1.3.5 Vista de mapa

El uso del componente de mapa llamado MkMapView, perteneciente al Map Kit de iOS, se puede usar mediante una simple instancia de la forma habitual.



Figura 21 Ejemplo de mapa en iOS (MKMapView)

El siguiente código muestra el código generado necesario para instanciarlo:

```
MKMapView mapView = [[MKMapView alloc] initWithFrame: CGRectMake  
(0,0,280,300)];  
[codeView addSubview: mapView];
```

## 7.1.4 Controles

Para cada uno de los controles enumerados previamente en el apartado de HTML5, se expondrá el código equivalente que genera dicho control para iOS.

### 7.1.4.1 Etiqueta

Cada etiqueta declarada en HTML5 generará el siguiente código, que podrá tener parámetros de acuerdo a las indicaciones dadas por el usuario.

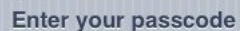


Figura 22 Ejemplo de etiqueta en iOS (UILabel)

Nótese que por simplicidad se está asumiendo que se define un rectángulo determinado que dará cabida al elemento de forma automática. Bien el generador o bien el uso de un layout determinado podrían calcular automáticamente estos valores. Además se deben tener en cuenta el resto de atributos de la etiqueta que también son parametrizables (color, fuente, alineación, etc.)

```
UILabel *labelCode = [ [UILabel alloc ] initWithFrame:CGRectMake(10, 20, 320, 20)];
labelCode.textAlignment = NSTextAlignmentCenter;
labelCode.textColor = [UIColor whiteColor];
labelCode.backgroundColor = [UIColor lightGrayColor];
labelCode.font = [UIFont fontWithName:@"Arial Rounded MT Bold" size:(16.0)];
labelCode.text = [NSString stringWithString: @"Nombre"];
[codeView addSubview:labelCode];
```

### 7.1.4.2 Campo de texto

De la misma manera que en el caso anterior, el siguiente código generaría un campo de texto en el controlador de vista correspondiente (el contenedor en HTML5).



Figura 23 Ejemplo de campo de texto en iOS (UITextField)

Nótese nuevamente la existencia de un rectángulo que da cabida al elemento y la parametrización de múltiples atributos (estilo del borde, color del texto, alineación, etc.)

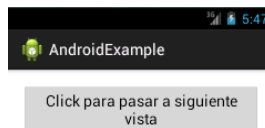
```

UITextField *txtFieldCode = [[UITextField alloc] initWithFrame:CGRectMake(10,
200, 300, 30)];
txtFieldCode.borderStyle = UITextBorderStyleRoundedRect;
txtFieldCode.textAlignment = UITextAlignmentCenter;
txtFieldCode.textColor = [UIColor blackColor];
txtFieldCode.backgroundColor = [UIColor whiteColor];
[codeView addSubview:txtFieldCode];

```

### 7.1.4.3 Botón

En este caso el código para generar un botón muestra como novedad la inclusión de un método que será ejecutado cuando exista algún evento sobre el mismo.



**Figura 24 Ejemplo de botón en iOS (Button)**

El caso más típico de activación será cuando el usuario presiona el botón.

```

UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[button addTarget:self action:@selector(button1Clicked:)
forControlEvents:UIControlEventTouchUpInside];
[button setTitle:@" Pulsa aquí " forState:UIControlStateNormal];
button.frame = CGRectMake(30,40,260,40);
[self.view addSubview:button];

```

```

- (void) button1Clicked: (id)sender
{
    NSLog(@"button1 Clicked");
}

```

### 7.1.4.4 Selector

Un selector es conocido en iOS como picker. Este caso es más complejo dado que se trata de una vista: es necesario crear el objeto UIPickerView, que conlleva una configuración diferente al resto de componentes.



**Figura 25 Ejemplo de selector en iOS (UIPickerView)**

Primero se instancia el elemento correctamente:

```
UIPickerView *myPickerView = [[UIPickerView alloc] initWithFrame:CGRectMake(0,
200, 320, 200)];
myPickerView.delegate = self;
myPickerView.showsSelectionIndicator = YES;
[codeView addSubview:myPickerView];
```

Y seguidamente se ha de configurar mediante los métodos delegados, que proveerán la información necesaria para terminar de configurar el aspecto:

```
//Este método sera llamado cada vez que se seleccione un element
- (void)pickerView:(UIPickerView *)pickerView didSelectRow: (NSInteger)row
inComponent:(NSInteger)component {
//Vacio
}

// Se le indica al component cuantas filas va a mostrar
- (NSInteger)pickerView:(UIPickerView *)pickerView
numberOfRowsInComponent:(NSInteger)component {
    NSInteger numRows = 2;

    return numRows;
}

//Se le indica al componente cuantos elementos contendrá, es decir, cuantas
columnas (generalmente una)
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}

// Se le indica al componente el slogan de cada elemento
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
forComponent:(NSInteger)component {
    NSString *title;
    if (row == 1)
        title = [@"Mercedes"];
    if (row ==2)
        title = [@"Audi"];

    return title;
}

// Se le indica al component la anchura de cada elemento
- (CGFloat)pickerView:(UIPickerView *)pickerView
widthForComponent:(NSInteger)component {
    int sectionWidth = 300;

    return sectionWidth;
}
```

### 7.1.4.5 Interruptor

La creación de un interruptor es relativamente sencilla.



Figura 26 Ejemplo de interruptor en iOS (UISwitch)

El código requerido necesita de muy pocas líneas (aunque se puede acompañar de un método que se ejecute cuando cambia el valor, como se puede apreciar en el ejemplo):

```
UISwitch* switchCode = [[UISwitch alloc] initWithFrame:CGRectMake(140, 60, 300, 200)];  
[switchCode addTarget:self action:@selector(switchControlinAction:)  
  forControlEvents:UIControlEventValueChanged];  
[codeView addSubview:switchCode];
```

### 7.1.4.6 Barras deslizantes o sliders

La barra deslizante se ha de configurar con un valor mínimo y máximo que define un rango, así como un valor inicial.



Figura 27 Ejemplo de barra deslizante en iOS (UISlider)

El siguiente código muestra el código generado que lo realiza:

```
CGRect frame = CGRectMake(0.0, 0.0, 200.0, 10.0);  
UISlider *slider = [[UISlider alloc] initWithFrame:frame];  
[slider addTarget:self action:@selector(sliderAction:)  
  forControlEvents:UIControlEventValueChanged];  
[slider setBackgroundColor:[UIColor clearColor]];  
slider.minimumValue = 0.0;  
slider.maximumValue = 50.0;  
slider.continuous = YES;  
[codeView addSubview:slider];
```

## 7.2 HTML5 a Android

En general una aplicación Android está formada por varios tipos de contenido, entre los que se encuentran:

- Ficheros fuente en Java.
- Ficheros generados para tests.
- Librerías.
- Recursos tipo iconos, imágenes.
- Ficheros XML para especificar el diseño de las pantallas (los llamados `activities.xml`).
- Ficheros XML para definir los menús.
- Ficheros de propiedades, manifiesto, etc.

En general la estructura es clara, sencilla y fácil de manejar debido a su aspecto intuitivo.

Para realizar la generación de código se partirá de un proyecto base generado automáticamente por el entorno Eclipse, aunque todos los ficheros pueden ser generados manualmente por línea de comandos. Para realizar la compilación manual se utiliza Ant.

Sobre dicha base, el generador propuesto en este trabajo añadirá los contenidos especificados en el lenguaje HTML5 utilizado como DSL.

### 7.2.1 Contenedores

Cada contenedor en HTML5 se mapeará a una Activity en Android. Cada Activity está compuesta de:

- Un fichero `XXActivity.java` que implementa el código asociado a la Activity. Por ejemplo, incluye el código para ejecutar las acciones de la pulsación de botones.
- Un fichero `activity_xx.xml` que define en XML el aspecto gráfico de la pantalla para esa Activity. Define botones, etiquetas, etc. incluyendo propiedades y métodos de los mismos que luego son ejecutados en la parte java.
- Ficheros de recursos tipo imágenes, iconos, mensajes (para internacionalización), etc.

- El fichero de manifiesto de la aplicación AndroidManifest.xml, que ha de ser modificado para indicar la nueva Activity añadida.

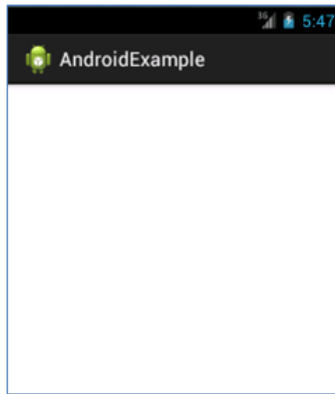


Figura 28 Ejemplo de contenedor en Android

El caso de generación de un nuevo contenedor se definirá una clase que hereda de Activity con el siguiente código:

```
import android.os.Bundle;
import android.app.Activity;

public class FirstActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_first);
    }
}
```

Por otra parte es necesario crear un archivo XML para definir el layout:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".FirstActivity" >
//Los elementos y controles se definirán en esta parte
</RelativeLayout>
```

Por defecto se usará un layout de tipo relativo, aunque existen muchos otros como ya se mencionó anteriormente.

## 7.2.2 Navegación

Por defecto se soportarán dos tipos de navegación ya conocidos: lineal y tabular.

La navegación lineal no requiere implementación especial en Android dado que por defecto todas las Activities por las que se navega quedan apiladas automáticamente y se pueden recuperar mediante el botón disponible en todos los dispositivos.



Figura 29 Ejemplo de navegación lineal en Android

Para llevar a cabo navegación lineal, basta con llamar a la siguiente Activity mediante el Intent correspondiente. Por ejemplo, en el caso de un botón que navegue a otra vista se generará el siguiente código:

```
public void startAct(View view) {  
    Intent intent = new Intent(this, SecondActivity.class);  
    startActivity(intent);  
}
```

La vuelta atrás no requiere de acción ni código especial: el usuario pulsa el botón y la última Activity vuelve al plano principal.

En cuanto a navegación tabular, es necesario crear una Activity que actúe como contenedora y que albergue un control de tipo TabWidget.



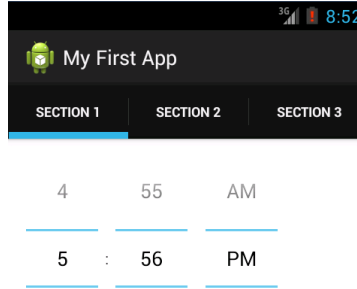


Figura 30 Ejemplo de control que permite navegación tabular en Android

Éste estará contenido en un TabHost. El siguiente código muestra el fichero XML que configura la Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

Por otra parte el fichero java a generar muestra una clase que hereda de TabActivity y que en la creación implementa el número de tabs requeridos:

```
public class ThirdActivity extends TabActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_third);

        TabHost tabHost = getTabHost();
```

```

        tabHost.addTab(tabHost.newTabSpec("tab_1") .setIndicator("Mis Fotos")
            .setContent(new Intent(this, Tab1Activity.class)));
        tabHost.addTab(tabHost.newTabSpec("tab_2") .setIndicator("Mapa Fotos")
            .setContent(new Intent(this, Tab2Activity.class)));
        tabHost.addTab(tabHost.newTabSpec("tab_3") .setIndicator("Tiendas")
            .setContent(new Intent(this, Tab3Activity.class)));
    }
}

```

La clase ha de crear cada una de las pestañas que van a estar disponibles y para cada una de ellas asocia una Activity que inicia a través de un Intent.

### 7.2.3 Vistas

En Android existen diferentes vistas predefinidas que podemos usar para cubrir las necesidades de mostrar los elementos con una determinada configuración, por ejemplo de tabla o lista. Se detallan en los siguientes apartados.

#### 7.2.3.1 Vista formulario

La denominada vista formulario no tiene traducción directa en Android dado que no necesita nada especial para implementarse. Este modo de vista añade y muestra controles o elementos en el interfaz de usuario y esto mismo se puede hacer en el contenedor o Activity actual que se esté generando en Android. Es decir, este modo sencillamente muestra todo aquello que se le ha indicado sin ninguna distribución especial. Y de la misma forma que se define en HTML5 se traduce a Android: Los elementos a añadir se incluirían en la definición de la Activity correspondiente.

```

<Button
    android:id="@+id/button1"
    [...]
    android:text="Click para pasar a siguiente vista" />

<TextView
    android:id="@+id/textView1"
    [...]
    android:text="Click para pasar a siguiente vista"/>

```

### 7.2.3.2 Vista de tabla

Para mostrar los elementos en modo tabla, Android ofrece GridView. Consiste en un ViewGroup para colocar elementos en un array de dos dimensiones. Los elementos a visualizar se insertan utilizando un ListAdapter.

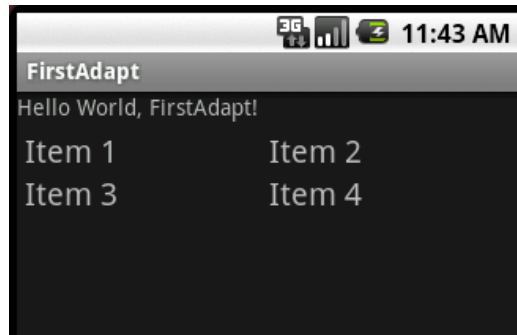


Figura 31 Ejemplo de table en Android (GridView)

En ese caso el código generado requerirá nuevamente actualizar el fichero XML de layout correspondiente a la Activity que se esté generando:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Y el código Java de la Activity incluiría las siguientes líneas. Es necesario asociar un adaptador MyViewAdapter que se encargará de 'rellenar' cada uno de los elementos que se incluyen en el grid.

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

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new MyViewAdapter(this));
};
```

El adaptador debe controlar qué elementos se incluyen en cada parte de la tabla: estos elementos pueden ser imágenes, texto u otros controles o vistas.

```
public class MyViewAdapter extends BaseAdapter {
    private Context mContext;

    public MyViewAdapter (Context c) { mContext = c; }

    public int getCount() { return 4; }

    public Object getItem(int position) { return null; }

    public long getItemId(int position) { return 0; }

    public View getView(int position, View convertView, ViewGroup parent) {
        TextView textView;
        if (convertView == null) {
            textView = new TextView(mContext);
            textView.setLayoutParams(new GridView.LayoutParams(85, 85));
            textView.setText("Elemento:"+position);
        } else {
            textView = (TextView) convertView;
        }

        return textView;
    }
}
```

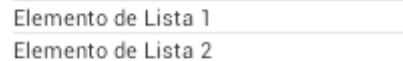
El código define para cada posición del array la vista que se va a mostrar. En este caso se ha elegido un tipo TextView aunque se podría haber optado por cualquier otro elemento o control.

### 7.2.3.3 Vista de lista

Esta vista se crea de manera muy similar a la anterior vista de tabla. La diferencia es que los elementos se muestran en una lista en lugar de una rejilla. El código generado en los archivos XML y Java es prácticamente idéntico salvo en el tipo de objetos creados. El contenedor XML deberá incluir el siguiente componente de vista:

```
<ListView
    android:id="@+id/ListView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1" >
</ListView>
```

Se puede apreciar cómo el control utiliza referencias a otros elementos (textView1) para situarse de forma relativa en la pantalla.



Elemento de Lista 1  
Elemento de Lista 2

Figura 32 Ejemplo de lista en Android (ListView)

En el código Java, en la creación de la Activity, es necesario relacionar el elemento ListView con el adaptador correspondiente. El adaptador se encargará nuevamente de presentar en cada posición de la lista la información requerida:

```
public class TableActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_table);  
  
        ListView listView = (ListView) findViewById(R.id.listView1);  
        listView.setAdapter(new MyViewAdapter(this));  
  
    }  
}
```

El adaptador cumple la misma función que en el apartado anterior. En cada posición devuelve la vista requerida, en este caso de tipo TextView:

```
public class MyViewAdapter extends BaseAdapter {  
    private Context mContext;  
  
    public MyViewAdapter (Context c) { mContext = c; }  
  
    public int getCount() { return 3; }  
  
    public Object getItem(int position) { return null; }  
  
    public long getItemId(int position) { return 0; }  
  
    public View getView(int position, View convertView, ViewGroup parent) {  
        TextView textView;  
        if (convertView == null) {
```

```

        textView = new TextView(mContext);
        if (position == 1)
        {
            textView.setText("Alemania");
        }
        if (position == 2)
        {
            textView.setText("España");
        }

    } else {
        textView = (TextView) convertView;
    }
    return textView;
}
}

```

#### 7.2.3.4 Vista de imagen

En Android crear una imagen es muy sencillo y no requiere de código Java inicialmente.



Figura 33 Ejemplo de imagen en Android (ImageView)

Se puede mostrar una imagen añadiéndola como recurso e incluyéndola en el código de layout XML tal y como muestra el ejemplo:

```

<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:src="@drawable/germanFlag" />

```

#### 7.2.3.5 Vista de mapa

Esta vista requiere de un componente de tipo vista que implementa los servicios de cartografía proporcionados por Google Maps. Dejando a un lado los aspectos de solicitud de licencia para usar el API, se requiere actualizar el XML de layout con el siguiente componente:

```

<?xml version="1.0" encoding="utf-8"?>
<com.google.android.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="Your Maps API Key goes here"
/>

```

Por otra parte se requiere crear una clase que herede de MapActivity y que se utilizará para gestionar todos los aspectos relacionados con el mapa.

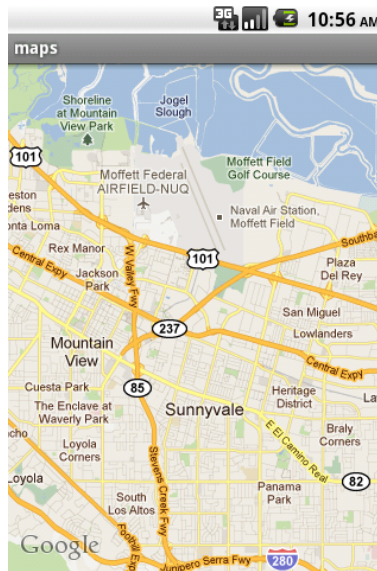


Figura 34 Ejemplo de mapa en Android (MapView)

Tal y como sugieren en la documentación, se deben implementar los siguientes métodos:

```

public class HelloGoogleMaps extends MapActivity
@Override
protected boolean isRouteDisplayed() {
    return false;
}
@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    MapView mapView = (MapView) findViewById(R.id.mapview);
    mapView.setBuiltInZoomControls(true);
}

```

Adicionalmente se podrían implementar otros elementos para añadir elementos al mapa, personalizar la apariencia o ejecutar cualquiera de las funciones que ofrece la API de Google Maps.

## 7.2.4 Controles

### 7.2.4.1 Etiqueta

La creación de este elemento sólo requiere de su inclusión en el fichero XML del layout.

Click para pasar a siguiente  
vista

Figura 35 Ejemplo de etiqueta en Android (TextView)

La propia cadena de texto se incluye como parte de los atributos del elemento. Nuevamente se incluyen referencias de posicionamiento con respecto a otros elementos (button3 y button4):

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button4"
    android:layout_alignBottom="@+id/button4"
    android:layout_toRightOf="@+id/button3"
    android:text="Nombre"
    android:textAppearance="?android:attr/textAppearanceLarge" />
```

### 7.2.4.2 Campo de Texto

Es necesario añadir el elemento en el XML para mostrarlo en la Activity:

```
<EditText
    android:id="@+id/editText_Name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/button3"
    android:layout_alignLeft="@+id/textView1"
    android:ems="10" >
    <requestFocus />
</EditText>
```



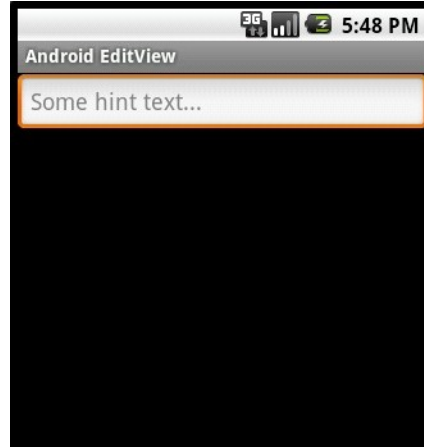


Figura 36 Ejemplo de etiqueta en Android (TextView)

Si se deseara acceder al contenido del campo bastaría con localizarlo en el código Java y acceder a sus métodos:

```
mEdit = (EditText)findViewById(R.id.editText_Name);  
String message = mEdit.getText().toString();
```

### 7.2.4.3 Botón

En el caso del botón se incluiría el siguiente fragmento en el fichero XML del layout que debería ir acompañado del código necesario en Java para procesar la acción del botón:

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/button3"  
    android:layout_alignLeft="@+id/edit_message"  
    android:layout_marginBottom="35dp"  
    android:onClick="startActivity_NextPage"  
    android:text="Siguiete Página" />
```

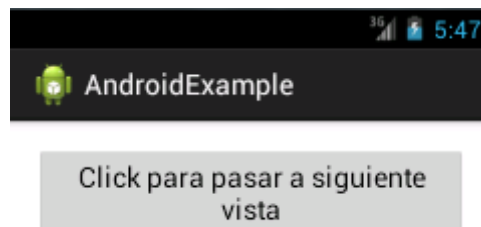


Figura 37 Ejemplo de botón en Android (Button)

El siguiente código podría localizarse en la clase Activity asociada al interfaz que se esté generando en cada momento:

```
public void startActivity_NextPage (View view) {  
    Intent intent = new Intent(this, nextPageActivity.class);  
    startActivity(intent);  
}
```

#### 7.2.4.4 Selector

El selector requiere ser iniciado con valores determinados, además de la configuración de layout habitual en XML.

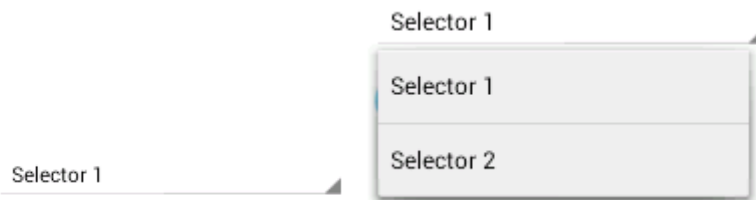


Figura 38 Ejemplo de selector en Android, en estado normal y desplegado (Spinner)

Los valores a cargar pueden introducirse por código o, de forma más correcta, almacenarse como recursos en un fichero XML paralelo que será cargado dinámicamente por el código Java, tal y como muestra el siguiente ejemplo:

```
<Spinner  
    android:id="@+id/carSelector_spinner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

El siguiente fragmento muestra otro fichero XML de recursos que contiene las cadenas de texto:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="carSelector_array">  
        <item>Mercedes</item>  
        <item>Audi</item>  
    </string-array>  
</resources>
```

Y finalmente el código de la Activity en Java que realiza la carga de los recursos y configura el Spinner para que incluya los valores deseados:

```
{
Spinner spinner = (Spinner) findViewById(R.id.spinner);

ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.carSelector_array, android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);

}
```

#### 7.2.4.5 Interruptor

El interruptor se implementa en Android mediante un tipo Switch si el API utilizado es de nivel 14 o superior.



Figura 39 Ejemplo de interruptores en Android (Switch)

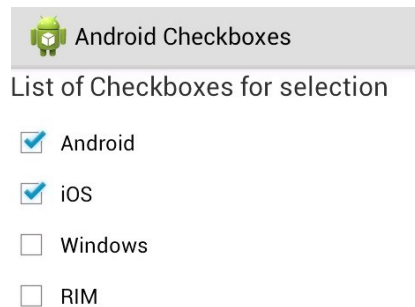


Figura 40 Ejemplo de checkboxes en Android (Checkbox)

En los casos en los que no se implemente ese nivel, siempre se puede recurrir al tipo CheckBox, que presenta menos problemas de compatibilidad al ser más antiguo.

```
<CheckBox
    android:id="@+id/checkbox1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="17dp"
    android:text="Marque la casilla para activar la función" />
```

#### 7.2.4.6 Barras deslizantes o sliders

Las barras deslizantes se consiguen gracias a SeekBar.



Figura 41 Ejemplo de barra deslizante en Android (SeekBar)

Basta con incluir el elemento en XML para disponer de él en la aplicación. El acceso a sus propiedades y métodos se hará de la forma habitual.

```
<SeekBar
    android:id="@+id/seekBar1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="55dp"
    android:max="50"
    android:progress="43" />
```

Se debe apreciar que el componente define un valor máximo pero no uno mínimo, por lo que se deberán hacer transformaciones en los valores que se definan en HTML5, ya que no podrán ser copiados directamente. Por ejemplo un rango de 50-100 en HTML5 se traduciría a 0-50 en Android.

### 7.3 Tablas comparativas entre las plataformas

Las siguientes tablas resumen el código generado anteriormente para que sea fácil observar las diferencias entre ambas plataformas y sirva a modo de recopilatorio de la información previa.

#### 7.3.1 Contenedores

Se muestran los ficheros .h y .m en el caso de iOS y el fichero .java y .xml para el caso de Android.

iOS	Android
<pre>#import &lt;UIKit/UIKit.h&gt; @interface VistaPrincipalViewController : UIViewController { } @end  [...]  #import "VistaPrincipalViewController.h" @implementation VistaPrincipalViewController  - (void)viewDidLoad {     [super viewDidLoad];     [self setTitle:@"VistaPrincipal"]; }  - (void)didReceiveMemoryWarning {     [super didReceiveMemoryWarning]; }  - (void)viewDidUnload {     [super viewDidUnload]; }  - (void)dealloc {     [super dealloc]; }  @end</pre>	<pre>import android.os.Bundle; import android.app.Activity;  public class FirstActivity extends Activity {      @Override     protected void onCreate(Bundle savedInstanceState) {         super.onCreate(savedInstanceState);         setContentView(R.layout.activity_first);     }      [...]  &lt;RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent" android:layout_height="match_parent" android:paddingBottom="@dimen/activity_vertical_margin" android:paddingLeft="@dimen/activity_horizontal_margin" android:paddingRight="@dimen/activity_horizontal_margin" android:paddingTop="@dimen/activity_vertical_margin" tools:context=".FirstActivity" &gt; &lt;/RelativeLayout&gt;</pre>

Table 10 Código generado para modelar un contenedor en iOS y Android

### 7.3.2 Navegación

Se muestran dos tablas, para navegación lineal y tabular. En iOS se incluye el código para configurar la navegación lineal, así como para añadir más vistas a la pila y la vuelta hacia atrás. En Android, además del código Java, se incluye el código XML necesario para la navegación tabular.

iOS	Android
<pre>UINavigationController* navController = [[UINavigationController alloc] init]; VistaPrincipalViewController* vPVC = [[VistaPrincipalViewController alloc] init]; [navController pushViewController:vPVC animated:NO];  [...]  VistaSecundariaViewController *vSVC = [[VistaSecundariaViewController alloc] init]; [[self navigationController] pushViewController:vSVC animated:YES];</pre>	<pre>public void startAct(View view) {     Intent intent = new Intent(this,         SecondActivity.class);     startActivity(intent); }</pre>

```
[...]
[self.navigationController
popViewControllerAnimated:YES];
```

Table 11 Código generado para modelar navegación lineal en iOS y Android

iOS	Android
<pre>UITabBarController *tabBars = [[UITabBarController alloc] init]; NSMutableArray *localViewControllersArray = [[NSMutableArray alloc] initWithCapacity:2];  VistaTabular1ViewController *vT1VC = [[VistaTabular1ViewController alloc] init]; UINavigationController *myController1 = [[UINavigationController alloc] initWithRootViewController:vT1VC]; myController1.title = @"Vista 1";  VistaTabular2ViewController *vT2VC = [[VistaTabular2ViewController alloc] init]; UINavigationController *myController2 = [[UINavigationController alloc] initWithRootViewController:vT2VC]; myController2.title = @"Vista 2";  [localViewControllersArray addObject:myController1]; [localViewControllersArray addObject:myController2];  tabBars.viewControllers = localViewControllersArray;  [self.view addSubview:tabBars.view];</pre>	<pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;TabHost xmlns:android="http://schemas.android.com/apk/res/android" android:id="@android:id/tabhost" android:layout_width="fill_parent" android:layout_height="fill_parent"&gt; &lt;LinearLayout android:orientation="vertical" android:layout_width="fill_parent" android:layout_height="fill_parent" android:padding="5dp"&gt; &lt;TabWidget android:id="@android:id/tabs" android:layout_width="fill_parent" android:layout_height="wrap_content" /&gt; &lt;FrameLayout android:id="@android:id/tabcontent" android:layout_width="fill_parent" android:layout_height="fill_parent" android:padding="5dp" /&gt; &lt;/LinearLayout&gt; &lt;/TabHost&gt; [...]</pre> <pre>public class ThirdActivity extends TabActivity {  @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_third);  TabHost tabHost = getTabHost(); tabHost.addTab(tabHost.newTabSpec("tab_1") .setIndicator("Mis Fotos") .setContent(new Intent(this, Tab1Activity.class))); tabHost.addTab(tabHost.newTabSpec("tab_2") .setIndicator("Mapa Fotos") .setContent(new Intent(this, Tab2Activity.class))); tabHost.addTab(tabHost.newTabSpec("tab_3") .setIndicator("Tiendas") .setContent(new Intent(this, Tab3Activity.class))); } }</pre>

Table 12 Código generado para modelar navegación tabular en iOS y Android

### 7.3.3 Vistas

Por simplicidad se muestran los ejemplos para las dos vistas más relevantes, las de tabla y lista.

iOS	Android
<pre>UICollectionView *collectionView = [[UICollectionView alloc] initWithFrame:CGRectMake(0,0,280,200)]; collectionView.delegate = self; collectionView.dataSource = self; [self.view addSubview: collectionView];  [...]  - (NSInteger)numberOfSectionsInCollectionView:(UICollectionView *)collectionView { return 1; }  - (NSInteger)collectionView:(UICollectionView *)view numberOfItemsInSection:(NSInteger)section { return 4; }  - (UICollectionViewCell *)collectionView:(UICollectionView *)cv cellForItemAtIndexPath:(NSIndexPath *)indexPath; { Cell *cell = [cv dequeueReusableCellWithReuseIdentifier:@"MY_CELL" forIndexPath:indexPath]; cell.label.textAlignment = NSTextAlignmentCenter; cell.label.textColor = [UIColor blackColor]; cell.label.font = [UIFont boldSystemFontOfSize:35.0]; cell.label.backgroundColor = [UIColor whiteColor];  if (indexPath.item == 1) cell.label.text = [@"1,1"]; if (indexPath.item == 2) cell.label.text = [@"1,2"]; if (indexPath.item == 3) cell.label.text = [@"2,1"]; if (indexPath.item == 4) cell.label.text = [@"2,2"]; return cell; }</pre>	<pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;GridView xmlns:android="http://schemas.android.com/apk/res/android" android:id="@+id/gridview" android:layout_width="fill_parent" android:layout_height="fill_parent" android:columnWidth="90dp" android:numColumns="auto_fit" android:verticalSpacing="10dp" android:horizontalSpacing="10dp" android:stretchMode="columnWidth" android:gravity="center" /&gt;  [...]  public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main);  GridView gridView = (GridView) findViewById(R.id.gridview); gridView.setAdapter(new MyViewAdapter(this)); }  [...]  public class MyViewAdapter extends BaseAdapter { private Context mContext; public MyViewAdapter (Context c) { mContext = c; } public int getCount() { return 4; } public Object getItem(int position) { return null; } public long getItemId(int position) { return 0; } public View getView(int position, View convertView, ViewGroup parent) { TextView textView; if (convertView == null) { textView = new TextView(mContext); textView.setLayoutParams(new GridView.LayoutParams(85, 85)); textView.setText("Elemento:" + position); } else { textView = (TextView) convertView; }  return textView; } }</pre>

Table 13 Código generado para modelar vistas de tipo tabla en iOS y Android

---

## iOS

```
UITableView *myTableView = [[UITableView alloc]
initWithFrame:CGRectMake(10,10,280,200)];
myTableView.delegate = self;
myTableView.dataSource = self;
[self.view addSubview:myTableView];

[...]

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    return 2;
}

- (UITableViewCell *)tableView:(UITableView
*)tableView
cellForRowAtIndexPath:(NSIndexPath
*)indexPath {
    static NSString *cellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:cellIdentifier];
    if (cell == nil)
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:cellIdentifier];
    if (indexPath.row == 0)
        cell.textLabel.text=@"Elemento de Lista 1";
    if (indexPath.row == 1)
        cell.textLabel.text=@"Elemento de Lista 2";
    return cell;
}
```

---

## Android

```
<ListView
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1" >
</ListView>

[...]

public class TableActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_table);

        ListView listView = (ListView)
            findViewById(R.id.listView1);
        listView.setAdapter(new MyViewAdapter(this));
    }
}

[...]

public class MyViewAdapter extends BaseAdapter {
    private Context mContext;

    public MyViewAdapter (Context c) { mContext = c; }

    public int getCount() { return 3; }

    public Object getItem(int position) { return null; }

    public long getItemId(int position) { return 0; }

    public View getView(int position, View convertView,
        ViewGroup parent) {
        TextView textView;
        if (convertView == null) {
            textView = new TextView(mContext);
            if (position == 1)
                textView.setText("Alemania");
            if (position == 2)
                textView.setText("España");
        } else {
            textView = (TextView) convertView;
        }
        return textView;
    }
}
```

---

Table 14 Código generado para modelar vistas de tipo lista en iOS y Android



## 7.3.4 Controles

Nuevamente por simplicidad se muestran dos de los controles más comunes, etiqueta y botón.

iOS	Android
<pre>UILabel *labelCode = [ UILabel alloc ] initWithFrame:CGRectMake(10, 20, 320, 20)]; labelCode.textAlignment = UITextAlignmentCenter; labelCode.textColor = [UIColor whiteColor]; labelCode.backgroundColor = [UIColor lightGrayColor]; labelCode.font = [UIFont fontWithName:@"Arial Rounded MT Bold" size:(16.0)]; labelCode.text = [NSString stringWithString: @"Nombre"]; [codeView addSubview:labelCode];</pre>	<pre>&lt;TextView     android:id="@+id/textView1"     android:layout_width="wrap_content"     android:layout_height="wrap_content"     android:layout_alignBaseline="@+id/button4"     android:layout_alignBottom="@+id/button4"     android:layout_toRightOf="@+id/button3"     android:text="Nombre"      android:textAppearance="?android:attr/textAppearanceLarge" /&gt;</pre>

Table 15 Código generado para modelar etiquetas en iOS y Android

iOS	Android
<pre>UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect]; [button addTarget:self     action:@selector(aMethod:)     forControlEvents:UIControlEventTouchUpInside]; [button setTitle:@"Pulsa aquí"  forState:UIControlStateNormal]; button.frame = CGRectMake(80, 210, 160, 40); [codeView addSubview:button];  [...]</pre> <pre>- (void) button1Clicked: (id)sender {     NSLog(@"button1 Clicked"); }</pre>	<pre>&lt;Button     android:id="@+id/button1"     android:layout_width="wrap_content"     android:layout_height="wrap_content"     android:layout_above="@+id/button3"     android:layout_alignLeft="@+id/edit_message"     android:layout_marginBottom="35dp"     android:onClick="startActivity_NextPage"     android:text="Siguiente Página" /&gt;</pre> <pre>[...]</pre> <pre>public void startActivity_NextPage (View view) {     Intent intent = new Intent(this,         nextPageActivity.class);     startActivity(intent); }</pre>

Table 16 Código generado para modelar botones en iOS y Android

## 8 Prototipo

Se desea realizar un prototipo consistente en una aplicación móvil básica, implementada para cada plataforma, que muestre, visualice y asegure que el código indicado en este trabajo y que sería proporcionado por un generador es válido y correcto, reforzando las afirmaciones realizadas en este documento. El prototipo consiste en desarrollar el código que supuestamente se generaría para tomarlo como base y referencia en la construcción posterior del generador real.

Para ello se trataría de realizar la misma aplicación en ambas plataformas, consistente en una vista principal que mediante un botón o enlace diese paso a una segunda vista, la cual tendría un componente de tipo tabular para permitir navegación alternativamente entre cada una de las pestañas. Cada una de las vistas podría contener determinados botones u otros controles simples de los ya vistos en este trabajo.

Mediante esta prueba se determinaría que la construcción del generador es posible y reafirmaría la viabilidad de la solución para los casos de ejemplo mostrados, además de servir como base para obtener las plantillas de código a generar. De hecho la elaboración de este trabajo se ha realizado con prototipos similares, especialmente la obtención del código a generar explicitado en el apartado 7

El código generado correspondería al siguiente código HTML5:

Vista Principal (VistaPrincipal.html):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Vista Principal</title>
  </head>
  <body>
    <button onclick="location.href='VistaSecundaria.html';">
      Click para pasar a siguiente vista
    </button>
    <a href="VistaSecundaria.html">
      Click para pasar a siguiente vista
    </a>
  </body>
</html>
```

Las pantallas de ambos prototipos se muestran a continuación. Muestran las diferencias evidentes de la distinta naturaleza gráfica de cada plataforma:

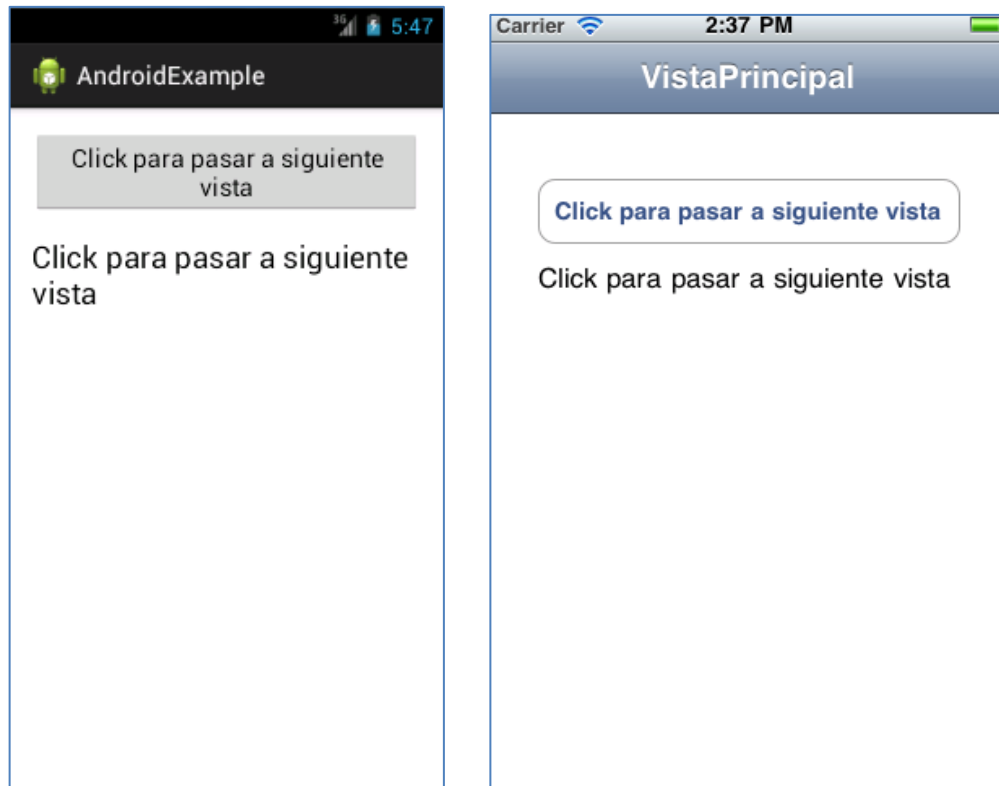


Figura 42 Botón y etiqueta generados en iOS y Android

Vista Secundaria (VistaSecundaria.html):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Vista Secundaria</title>
  </head>
  <body>
    <tabNavigation>
      <a href="VistaTabular1.html">Vista 1</a>
      <a href="VistaTabular2.html">Vista 2</a>
    </tabNavigation>
  </body>
</html>
```

En el caso de la vista secundaria se muestra implícitamente la vista tabular que contiene por defecto. En cuanto a la navegación lineal, la diferencia más patente radica en la

creación automática del botón de retroceso en la parte superior para iOS. En Android esto no es necesario dado que el botón existe en todos los dispositivos.

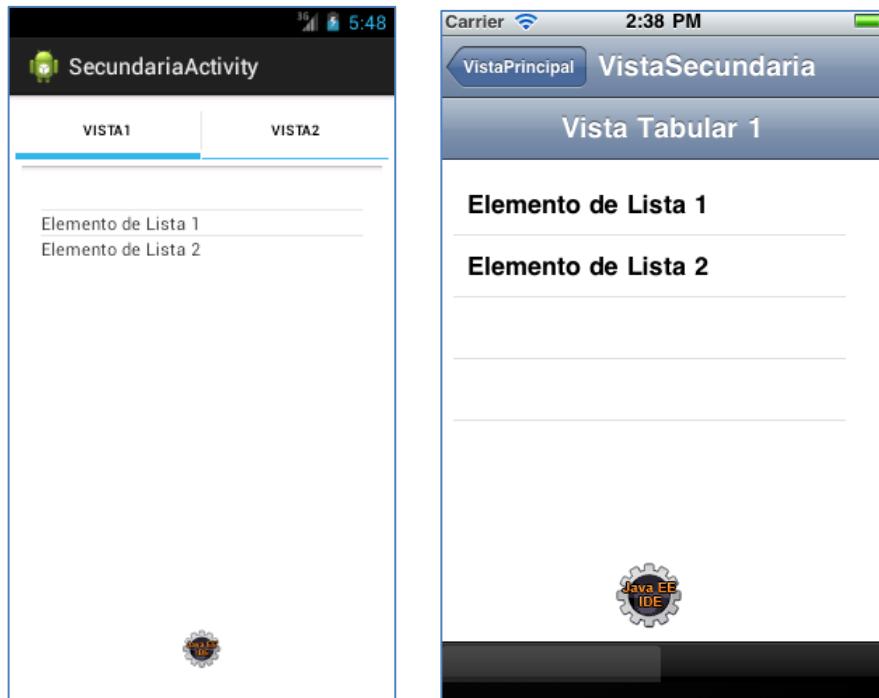


Figura 43 Elementos de tipo lista e imagen generados en iOS y Android

Vista tabular numero 1 (VistaTabular1.html):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Vista Tabular 1</title>
  </head>
  <body>
    <ul>
      <li>Elemento de Lista 1</li>
      <li>Elemento de Lista 2</li>
    </ul>
    Una foto</img>
  </body>
</html>
```

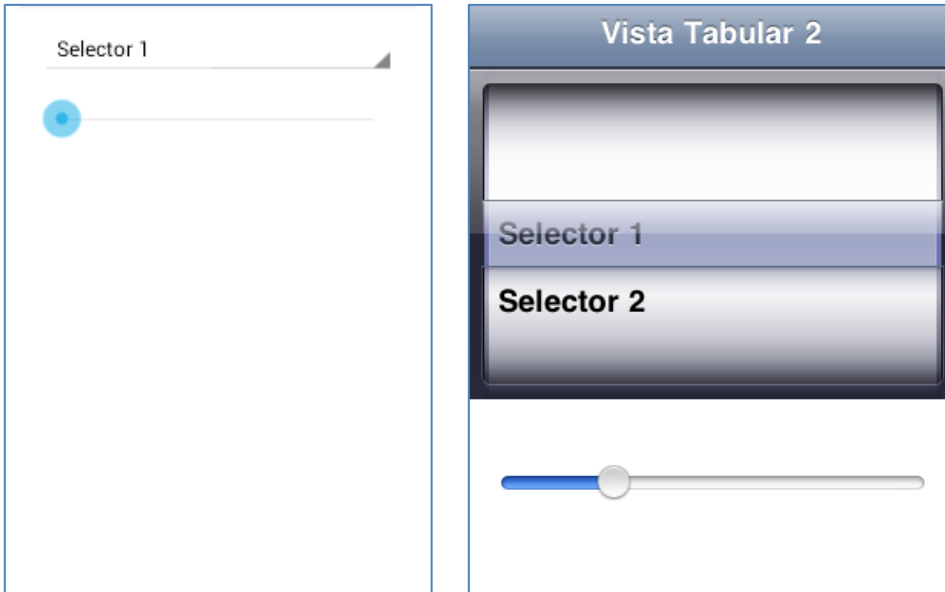
Las vistas tabulares se almacenan en otros contenedores como se apreciaba en el caso anterior. Se pueden observar las diferencias en la tabla y en la imagen mostrada en la parte inferior.



Vista tabular número 2 (VistaTabular2.html):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Vista Tabular 2</title>
  </head>
  <body>
    <select name="mySelector">
      <option value="Seleccion1">Selector 1</option>
      <option value="Seleccion2">Selector 2</option>
    </select>
    <input type="range" name="slider1" min="0" max="100"/>
  </body>
</html>
```

La segunda vista tabular muestra aún mayores diferencias entre algunos componentes, por ejemplo el selector. La barra deslizante muestra una mayor similitud.



## 9 Conclusiones y trabajos futuros

El trabajo ha mostrado una propuesta para desarrollar un generador abstrayendo conceptos comunes a ambas plataformas, centrándose en la capa de usuario y proponiendo diversas vías de investigación futuras para continuar este trabajo.

Se ha definido una estructura basada en contenedores, vistas y controles, así como conceptos de navegación, lineal y tabular, presentes en ambas plataformas de forma nativa. Dicha estructura está diseñada con objeto de ser aplicable al resto de plataformas del mercado actual, por ejemplo RIM y Windows Phone. Los sistemas ofrecidos por estas plataformas son muy similares a los ya estudiados y sus peculiaridades podrían encajar perfectamente en la propuesta de este trabajo.

Por otra parte, el trabajo se puede enriquecer añadiendo un mayor detalle y mayor exhaustividad en los elementos gráficos que se contemplan. Cada elemento gráfico cuenta con tantos detalles que cada uno por sí mismo podría considerar una pequeña investigación independiente.

Otros aspectos que pueden completar la investigación de este trabajo incluyen el análisis detallado de la distribución de elementos gráficos. Cada plataforma cuenta con su propio sistema de layouts y, de igual forma, se pueden establecer aspectos comunes y al mismo tiempo contemplar detalles que puedan aprovechar el potencial de cada plataforma.

Uno de los aspectos más interesantes es el del lenguaje de control asociado a los elementos gráficos. El código necesario para definir algoritmos o las acciones que debe ejecutar un botón necesita también ser transformado desde el DSL (en este caso se propuso Javascript por su flexibilidad y cercanía a HTML5) a Objective-C o Java según la plataforma. Este punto abarca tantos aspectos que puede requerir igualmente estudios separados.

Finalmente, cabe mencionar que las capas de datos y proceso requerirán estudios similares al estar presentes en todas las plataformas y ser de suma importancia en el desarrollo de aplicaciones móviles.

## **10 Bibliografía**

iOS Dev Center: <https://developer.apple.com/devcenter/ios/index.action>

Android Developers: <http://developer.android.com/index.html>



## 11 Siglas

Ant: Another Neat Tool (Herramienta para construir proyectos automáticamente)

API: Application Programming Interface (Interfaz de Programación de Aplicaciones)

DSL: Domain Specific Language (Lenguaje Específico de Dominio)

HTML: HyperText Markup Language (Lenguaje de Marcado Hipertextual)

iOS: iPhone Operating System (Sistema Operativo para iPhone)

MVC: Model View Controller (Modelo Vista Controlador)

SDK: Software Development Kit (Kit de Desarrollo Software)

SQL: Structured Query Language (Lenguaje que Peticiones Estructurado)

UI: User Interface (Interfaz de Usuario)

XML: eXtensible Markup Language (Lenguaje de Marcas Extensible)