Máster Universitario de Investigación en
Ingeniería de Software y Sistemas Informáticos

Definición de un marco multiplataforma bajo un lenguaje de dominio propio para el manejo de recursos multimedia en aplicaciones móviles

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

Autor: Pablo Berguño Díaz

DIRECTOR: Ismael Abad Cardiel

Curso Académico 2014/2015
Convocatoria Junio

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO: Ingeniería del Software

CÓDIGO DE ASIGNATURA: 31105128

TÍTULO DEL TRABAJO: Definición de un marco multiplataforma bajo un lenguaje de dominio propio para el manejo de recursos multimedia en aplicaciones móviles

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

AUTOR: Pablo Berguño Díaz

DIRECTOR: Ismael Abad Cardiel

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

Resumen del trabajo

En este trabajo se presenta el desarrollo de un lenguaje de dominio específico (DSL) para gestionar recursos multimedia en el marco de las aplicaciones móviles.

Para ello, se ha realizado un estudio que abarca la forma de gestionar los recursos multimedia entre las diferentes SDK de las principales plataformas móviles, y la gestión de estos recursos en herramientas de terceros que permiten un desarrollo multiplataforma.

Se realiza un prototipo donde se aplica este lenguaje y se evalúan los beneficios de su utilización.

Palabras clave: DSL, desarrollo móvil multiplataforma, recursos multimedia

Abstract

This work presents the development of a domain-specific language (DSL) for managing multimedia resources within mobile applications.

To do this, we conducted a study that covers how to manage media assets among different SDK major mobile platforms, and management of these resources in third-party tools that allow multiplatform development.

This language is applied to a prototype and benefits of their use are assessed.

Keywords: DSL, mobile development platform, multimedia resources

Índice

Listas de figuras

Tablas

1 INTRODUCCIÓN

Los sistemas operativos de los dispositivos móviles son un mercado fragmentado en el que se producen cambios cada vez más rápidos. Según Gartner [1] los sistemas operativos Android, iOS, y Windows se encuentran en las 3 primeras posiciones de ventas en el año 2014. El hecho de que sean plataformas diferentes, implica que los desarrolladores de software se vean obligados a desarrollar sus aplicaciones por separado.

El enfoque del desarrollo multi-plataforma *(cross-platform)* emerge para ayudar a los desarrolladores a implementar sus aplicaciones una única vez y las puedan replicar para las diferentes plataformas de forma casi automática, incrementando así la productividad. Siguiendo esta misma idea de ayudar a los desarrolladores, se plantea este estudio, cuyo objetivo es el desarrollo de un lenguaje DSL que permita gestionar los recursos multimedia en las principales plataformas móviles.

Cada una de las plataformas móviles tiene su propio lenguaje de programación (C# Windows Mobile, Xcode iOS, Java en Android), y la manera en que se gestionan sus recursos multimedia también varía. En este aspecto un lenguaje DSL va a permitir mejorar la portabilidad y reusabilidad de las aplicaciones así como proporcionar un medio más claro de comunicar las intenciones del sistema.

Este trabajo se organiza de la siguiente forma: primero se realiza un estudio inicial sobre la gestión de los recursos multimedia dentro de las SDKs de las plataformas para después analizar cómo se realiza esta gestión en herramientas de terceros. Este análisis se centra sólo en las operaciones necesarias para dar una funcionalidad básica al lenguaje DSL, pero sienta las bases para poder añadir nuevas extensiones al mismo. Después se explicará los aspectos clave de los lenguajes DSL para ubicar al lector en el contexto de estos lenguajes. Posteriormente se presenta la propuesta del lenguaje junto a una aplicación del mismo. Por último las conclusiones, trabajos futuros y referencias del trabajo.

2 ÁMBITO DEL PROYECTO Y OBJETIVOS

El ámbito de este proyecto se enmarca dentro del desarrollo móvil multiplataforma. Sobre este marco se trata de desarrollar un lenguaje DSL que facilite la gestión de los recursos multimedia en las diferentes plataformas.

El objetivo de este proyecto es implementar un DSL que pueda utilizarse en el desarrollo de aplicaciones multiplataforma, en las 3 principales plataformas: Android, iOS y Windows Phone. Dado que el DSL en este punto del trabajo, es sobre todo una prueba de concepto, nuestros objetivos ideales pasarían por uno de estos dos:

- Uno de los objetivos ideales sería el desarrollo de una solución de lenguaje de dominio específico (DSL) que genere una API que pueda utilizarse para manejar recursos multimedia en las diferentes plataformas. Este DSL llevaría implícita la

distinción entre la plataforma sobre la que se despliegue la aplicación móvil, de manera que para los desarrolladores sería transparente la gestión de los recursos multimedia en las diferentes plataformas, ya que el DSL contendría toda la lógica para distinguir los controles y código de cada SDK.

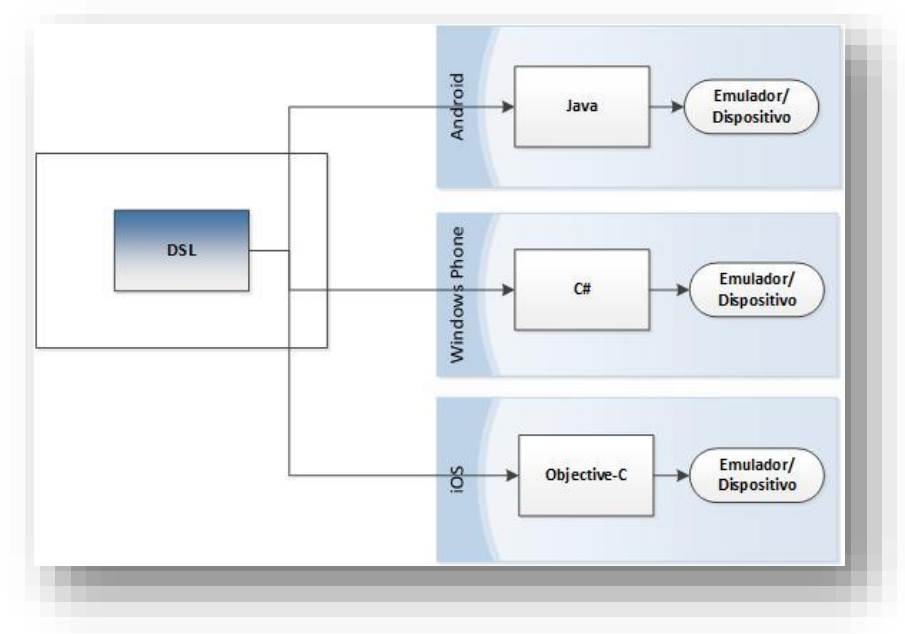


Ilustración 1. Posible objetivo 1

- Otro objetivo ideal, sería parecido al primero pero la solución DSL pasaría a estar desarrollada sobre una herramienta de terceros, de modo que cuando se utilice esa herramienta, los desarrolladores puedan importar una biblioteca con las especificaciones del lenguaje DSL, y puedan gestionar sus recursos multimedia a través de este lenguaje, de forma sencilla y con un lenguaje amigable.

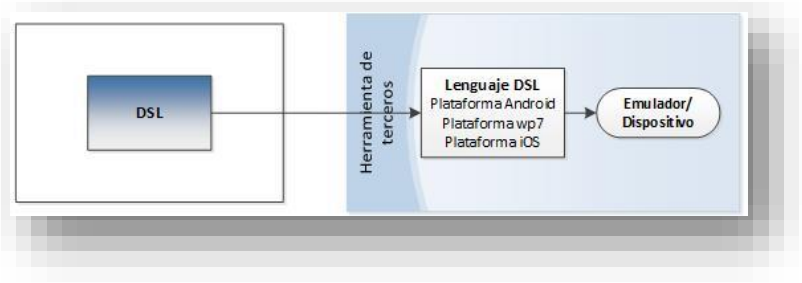


Ilustración 2. Posible objetivo 2

Tras avanzar el estudio, se opta por el objetivo 2. Se desarrolla un lenguaje DSL sobre una herramienta de terceros. Los motivos: conocer una herramienta de terceros de estas características y el hecho de hacer uso de una gestión de recursos multimedia ya implementada, probada por una gran comunidad de usuarios y con un proceso de mejora continua.

3 ESTADO DEL ARTE

Este apartado se divide en:

- Cómo se gestionan los recursos multimedia entre las diferentes SDK de las plataformas Android, iOS y Windows Phone.
- La forma de resolver la gestión de recursos multimedia en productos de terceros.

3.1 GESTIÓN DE RECURSOS MULTIMEDIA EN LAS APIS DE LAS SDKS

En este apartado se presenta un análisis de cómo se gestionan los recursos multimedia entre las diferentes plataformas. Este análisis funcional se hace sobre la documentación y código fuente de las propias SDK, intentando buscar elementos comunes que puedan ayudar a plantear el DSL.

3.1.1 ANDROID

La plataforma Android proporciona un API a la que pueden acceder las aplicaciones para interactuar con el sistema. Las partes en las que se compone el API son las siguientes:

- Una serie de paquetes básicos y clases.
- Un conjunto de elementos y atributos XML para la declaración y el uso de recursos y de un archivo *manifest*, donde podremos aplicar las configuraciones básicas de la aplicación móvil.
- Un conjunto de *Intents*, que sirven para invocar componentes (*activities*). Ya sean componentes de Interfaz Gráfica, *services*, código ejecutándose en segundo plano, etc. Básicamente nos permiten llamar a aplicaciones externas a la nuestra, lanzar eventos a los que otras aplicaciones puedan responder, lanzar alarmas, etc.
- Un conjunto de permisos que la aplicación pueda solicitar, así como refuerzos de permisos ya incluidos en el sistema.

El análisis de este estudio se basa en la gestión de recursos multimedia sobre la plataforma Android que aplica sobre el nivel de API 21. Este nivel identifica de forma exclusiva el número de revisión del API que ofrece la plataforma Android.

A continuación se analiza cómo se lleva a cabo la gestión de audio, video e imagen a través de la SDK de esta plataforma. El análisis se centra en los métodos sobre los que va a aplicar el desarrollo del lenguaje DSL.

* En los bloques de código se utiliza el lenguaje de programación Java

3.1.1.1 Gestión de audio y vídeo

El sistema operativo Android proporciona muchas formas de controlar la reproducción de archivos de tipo audio o video [2]. Una de estas formas es a través de la clase denominada *MediaPlayer,* que consta de una máquina de estados interna que gestiona todos los estados en el ciclo de vida de un objeto de estas características. A través de la API de esta clase se pueden reproducir desde ficheros almacenados en nuestro dispositivo, URLs externas (streaming) o URIs internas.

La lista de los formatos media que soporta Android, se encuentra disponible en el documento de soporte de formatos media [3].

A la hora de invocar a la clase Media podemos hacerlo a través de su constructor por defecto, o haciendo uso de sus métodos estáticos. Vamos a poner un ejemplo de cómo se procesaría un archivo de audio a través de estas dos formas:

```
private void playAudio(Integer media) {
    try {
        switch (media) {
            case LOCAL_AUDIO:
                path = "/sdcard/Download/music/1.mp3";
                mMediaPlayer = new MediaPlayer();
                mMediaPlayer.setDataSource(path);
                mMediaPlayer.prepare();
                mMediaPlayer.start();
                break;
            case RESOURCES_AUDIO:
                mMediaPlayer = MediaPlayer.create(this, R.raw.test_cbr);
                mMediaPlayer.start();
        }
    } catch (Exception e) {
        Log.e(TAG, "error: " + e.getMessage(), e);
    }
}
```

Dentro de la opción *LOCAL_AUDIO* se invoca a la clase *Media* a través del constructor por defecto. El método *prepare()* se encarga de preparar el reproductor para la reproducción de forma síncrona. Después de establecer el origen de datos y la superficie de la pantalla, es necesario invocar al método *prepare()* o *prepareAsync()* en función del tipo de media que se desee abrir. Para los archivos, se invoca a *prepare()* que bloquea hasta que *MediaPlayer* esté listo para reproducir. Para los *streams* se debe invocar a *prepareAsync()* que no bloquea la actividad mientras se almacenan los datos que se van descargando.

Dentro de la opción *RESOURCES_AUDIO* tenemos la invocación al método estático *create,* donde el primer parámetro *this* proporciona una interfaz a la información global sobre el entorno de la aplicación. El segundo parámetro es el nombre de la canción que se quiera reproducir. Se tendrá que crear un nuevo directorio denominado *raw,* que cuelgue del proyecto y donde ubicaremos los archivos de música.

Cuando se invoque la clase a través del método estático *create* no será necesaria la posterior invocación al método *prepare()* ya que irá inmerso en el primero.

Una vez creado el objeto *MediaPlayer* se pueden invocar los métodos para reproducir, pausar o parar la música.

```
mediaPlayer.start();
mediaPlayer.pause();
mediaPlayer.stop();
```

Cuando hayamos terminado de utilizar el objeto *MediaPlayer*, o bien cuando la actividad de la aplicación esté pausada (método *onPause()* invocado), o parada (método *onStop()* invocado), debemos hacer uso del método *release()*. De esta forma conseguimos que se liberen los recursos asociados a él (memoria, instancias de codecs,..). En caso de no invocarse podría hacer que demasiadas instancias de este objeto deriven en una excepción.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    if (mMediaPlayer != null) {
        mMediaPlayer.release();
        mMediaPlayer = null;
    }
}
```

La gestión del vídeo sería similar a la del audio:

```
private void playVideo(Integer Media) {
    doCleanUp();
    try {
        switch (Media) {
            case LOCAL VIDEO:
                path = "/sdcard/Download/video/2.mp4";
                break;
            case STREAM_VIDEO:
                path = "Your URL here";
                break;
        }
        mMediaPlayer = new MediaPlayer();
        mMediaPlayer.setDataSource(path);
        mMediaPlayer.setDisplay(holder);
        mMediaPlayer.prepare();
        mMediaPlayer.setOnBufferingUpdateListener(this);
        mMediaPlayer.setOnCompletionListener(this);
        mMediaPlayer.setOnPreparedListener(this);
        mMediaPlayer.setOnVideoSizeChangedListener(this);
        mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    } catch (Exception e) {
        Log.e(TAG, "error: " + e.getMessage(), e);
    }
}
```

Donde los métodos:

1. *setDataSource:* Establece el origen de datos.
2. *setDisplay:* Fija el contenedor que se va a utilizar para la visualización de la parte de vídeo.

3.1.1.2 Gestión de imágenes

Para la gestión de imágenes en Android vamos a analizar la clase *ImageView.* Esta clase puede representar cualquier imagen que se pueda sacar por pantalla.

Para establecer una imagen sobre un objeto *ImageView* se pueden utilizar cuatro métodos que varían en función de cómo se define la imagen que se pasa. La imagen y su método asociado pueden ser los siguientes:

- Un bitmap -> *setImageBitmap():* Establece una imagen 'drawable' como el contenido del objeto *ImageView.*
- Una URI -> *setImageURI():* Establece un objeto URI al contenido del objeto *ImageView*.
- Un identificador de recurso -> *setImageResource():* Utiliza un identificador de recurso para establecer su contenido sobre el objeto *ImageView.*
- Una imagen 'drawable' -> *setImageDrawable().* La imagen 'drawable' se entiende como cualquier imagen que puede ser pintada en pantalla y recuperada con el API (ejemplo: *getDrawable(int)),* o que se pueda aplicar sobre un recurso XML a través de unos atributos, como por ejemplo *Android:drawable* y *Android:icon.*

Es muy común utilizar un identificador de recurso para establecer el objeto *ImageView.* En este caso se podría utilizar un código similar a este:

```
ImageView mainImage = (ImageView) findViewById(R.id.imageView1);
mainImage.setImageResource(R.drawable.mainImage)
```

Propiedad *ScaleTypes*

Esta propiedad define la forma en la que la imagen se muestra dentro de *ImageView,* por ejemplo se puede hacer que la imagen ocupe toda la *ImageView,* o bien que se recorte y se centre. Las opciones sobre esta propiedad se definen sobre *ImageView.ScaleType,* por ejemplo:

- *ImageView.ScaleType.CENTER:* Esta propiedad centra la imagen sin escalarla, de manera que las dimensiones de la imagen no se alteran.
- *ImageView.ScaleType.CENTER_CROP:* Escala la imagen hasta que la anchura y la altura de la imagen sea la misma que la anchura o la altura de *ImageView.* Por ejemplo en una imagen pequeña, esto tendrá efecto de ampliación de toda la imagen.
- *ImageView.ScaleType.CENTER_INSIDE:* Escala la imagen y mantiene la relación de su aspecto.
- *ImageView.ScaleType.FIT_CENTER:* Ajusta la imagen al centro de *ImageView.*
- *ImageView.ScaleType.FIT_START:* Ajusta la imagen a la izquierda y margen superior de *ImageView.*
- *ImageView.ScaleType.FIT_END:* Ajusta la imagen a la derecha y margen inferior de *ImageView.*

- *ImageView.ScaleType.FIT_XY:* Ajusta la imagen dentro del largo y ancho de *ImageView* sin mantener la proporción.
- *ImageView.ScaleType.MATRIX:* Realiza el escalado a través de una matriz. Permite acciones más complejas sobre la imagen.

Rotación de una imagen con propiedad *MATRIX*

En la programación de gráficos las matrices se utilizan para transformar imágenes. Estas transformaciones pueden incluir desde el escalado, traslado o rotación de una imagen. Android incluye una clase *Android.graphics.Matrix* para dar soporte a esas transformaciones.

Si queremos aplicar rotación sobre una imagen, podríamos guiarnos por el siguiente código:

```
Matrix matrix = imageView.getImageMatrix();
imageView.setScaleType(ImageView.ScaleType.MATRIX);
matrix.postRotate(30, imageView.getWidth()/2, imageView.getHeight()/2);
imageView.setImageMatrix(matrix);
```

Primero se obtiene la matriz asociada con el objeto *ImageView* para después invocar al método *setScaleType* con el fin de utilizar una matriz modificada en el *ImageView*. En la siguiente línea se rota la imagen 30 grados sobre su punto central y finalmente se establece la matriz al objeto, para que surta efecto.

Establecer opacidad

La opacidad de una imagen se define en los siguientes estados: completamente transparente, completamente opaca, o bien fijarla con un estado intermedio. Para fijar la opacidad de una imagen se hace uso del método *setAlpha(),* o desde el nivel 11 del API, se puede utilizar el método *setImageAlpha().* A ambos métodos se les pasa un parámetro entero, donde 0 indica completamente transparente, mientras que 255 indica completamente opaco.

3.1.2 WINDOWS PHONE

La SDK de Windows Phone permite desarrollar Apps utilizando una variedad de lenguajes y herramientas. Por ejemplo, podemos desarrollar una aplicación usando XAML junto a un lenguaje administrado, lo que nos va a permitir mantener las inversiones en aplicaciones existentes [4].

Para proporcionar mayor flexibilidad y rendimiento, Windows 8 introduce la capacidad de utilizar C++ dentro de las aplicaciones desarrolladas con XAML y los juegos escritos con Direct3D.

El siguiente diagrama representa el conjunto de APIs que forman parte de la plataforma de Windows Phone.

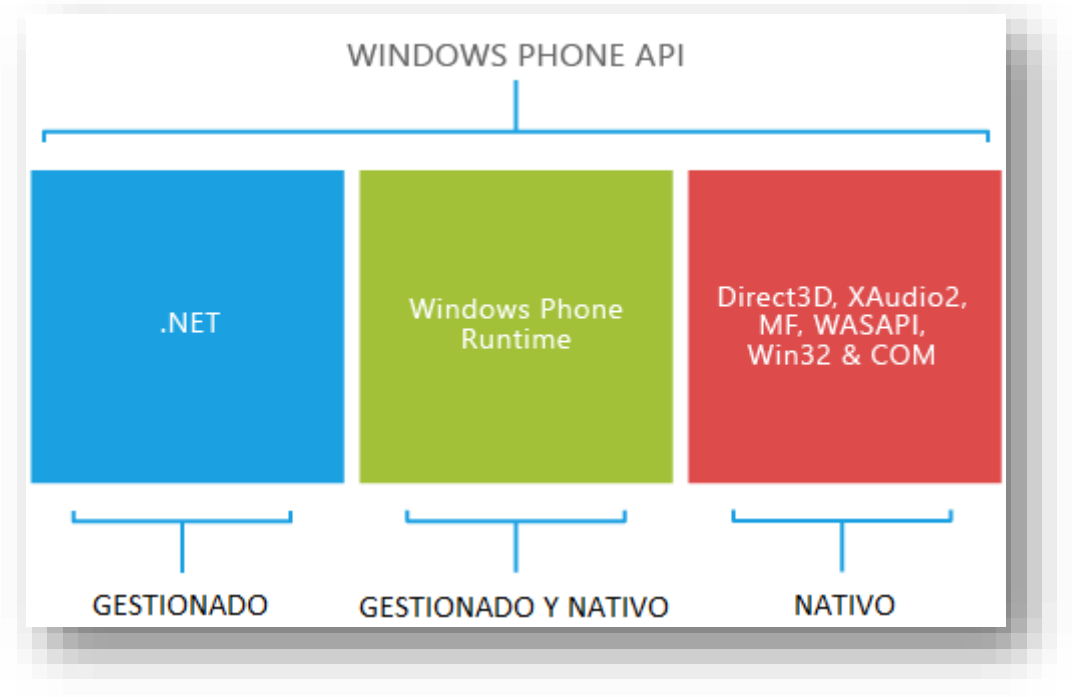


Ilustración 3. Windows Phone API

Dentro del API de .Net nos encontramos clases y tipos de los espacios de nombres *System* y *Microsoft.Phone* junto con funcionalidad añadida para Window Phone 8.

El API de Windows Phone Runtime es un subconjunto de APIs nativas que está integrado en el sistema operativo. Está implementado en C++ y proyectado en C#, VB.NET y C++ dando al usuario la opción de elegir el lenguaje de desarrollo.

Por último, las API de Win32 y COM tienen acceso a características de bajo nivel de la plataforma.

* En los bloques de código se utiliza el lenguaje de programación C# y XAML

3.1.2.1 Gestión de audio y vídeo

Dentro de Windows Phone 8 se puede incorporar audio y video haciendo uso de las siguientes características:

- A través de la clase *MediaPlayerLauncher*, podemos embeber audio o video mediante el reproductor multimedia del dispositivo.
- La clase *MediaElement* nos va a permitir reproducir audio y video a través de una interfaz mucho más personalizable.
- Se utilizará la clase *MediaStreamSource* para soluciones de streaming adaptativo.

- A través de la clase *Microsoft.Phone.BackgroundAudio* podemos crear aplicaciones que van a continuar ejecutándose aunque otra aplicación se encuentre en primer plano.

En este estudio, vamos a centrarnos en la clase *MediaElement* por ser la que contiene la lógica necesaria para las funciones que queremos dar a nuestro lenguaje DSL.

El siguiente ejemplo de uso de esta clase, asume que los ficheros de sonido están almacenados en el directorio *Sounds* de nuestro dispositivo. El método *InitializeSound()* es invocado una vez para leer los archivos de ese directorio e inicializar los *MediaElements.* Tras la inicialización, el método *PlaySound()* puede ser invocado para reproducir el sonido. Este método toma como argumento el *MediaElement* a reproducir.

```
private MediaElement  cardSound = new MediaElement();
private MediaElement _dingSound = new MediaElement();

private void InitializeSound()
{
    LoadSound("CardFlip.wav",   cardSound);
    LoadSound("success.wav",  _dingSound);
}

private async void LoadSound(string SoundFilePath, MediaElement SoundElement)
{
    // all sounds are stored in the Sounds folder
    try
    {
        Windows.Storage.StorageFolder folder = await
Package.Current.InstalledLocation.GetFolderAsync("Sounds");
        Windows.Storage.StorageFile file = await folder.GetFileAsync(SoundFilePath);
        var stream = await file.OpenReadAsync();
        SoundElement.AutoPlay = false;
        SoundElement.SetSource(stream, file.ContentType);
    }
    catch
    {
        System.Diagnostics.Debug.WriteLine("Problem opening sound file: " +
SoundFilePath);
    }
}

private void PlaySound(MediaElement SoundElement)
{
    // only play sounds if the user has 'play sounds' enabled
    if (_useSound)
    {
        try
        {
            SoundElement.Play();
        }
        catch
        {
            System.Diagnostics.Debug.WriteLine("Problem playing sound: " +
SoundElement.ToString());
        }
    }
}
```

Con la opción *AutoPlay=false* antes del método *SetSource* nos aseguramos que la reproducción no comience de forma automática durante la inicialización.

Nota: Cualquier aplicación que incorpora contenido Media, debe ajustarse a los requisitos de certificación de aplicaciones para Windows Phone. [5]

3.1.2.2 Gestión de imágenes

Dentro de Windows Phone existen diferentes maneras de mostrar imágenes en las aplicaciones móviles [6]. Para mostrar una imagen, se puede hacer uso de la clase *Image (Sytem.Windows.Controls)* o bien del objeto *ImageBrush (System.Windows.Media).*

A través del lenguaje XAML podremos incluir una imagen de la siguiente manera:

```
<Image Source="Image1.JPG" />
```

Donde la propiedad *Source* se indica la localización de la imagen a mostrar. Se puede establecer el origen ya sea a través de una URL pública o bien a través de una URL que apunte a un fichero incluido dentro del fichero instalable (*.xap) de la aplicación móvil.

La lista de los formatos de imagen que soporta Windows Phone, se encuentra disponible en la especificación de la clase *BitmapImage* [7].

Estiramiento de una imagen

Si no se establece el ancho y alto de una imagen, se mostrará con las dimensiones que vengan especificadas desde el origen. Cuando se establece el ancho y alto se crea un área rectangular para mostrar la imagen en su interior. Se puede especificar la forma en la que imagen rellene esa área:

- *None:* Con esta propiedad, la imagen no se estira para llenar las dimensiones de salida.
- *Uniform:* La imagen se escala para ajustarse a las dimensiones de salida, pero conservando la relación con respecto a su contenido. Este es el valor por defecto.
- *UniformToFill:* La imagen se escala hasta que llene completamente el área de salida, pero conservando su aspecto original.
- *Fill:* La imagen se escala hasta llenar las dimensiones de salida. Este escalado no se realiza de forma proporcional, es decir, el ancho y el alto se escalan de forma independiente, lo que puede hacer que la imagen aparezca distorsionada para llenar completamente el área de salida.

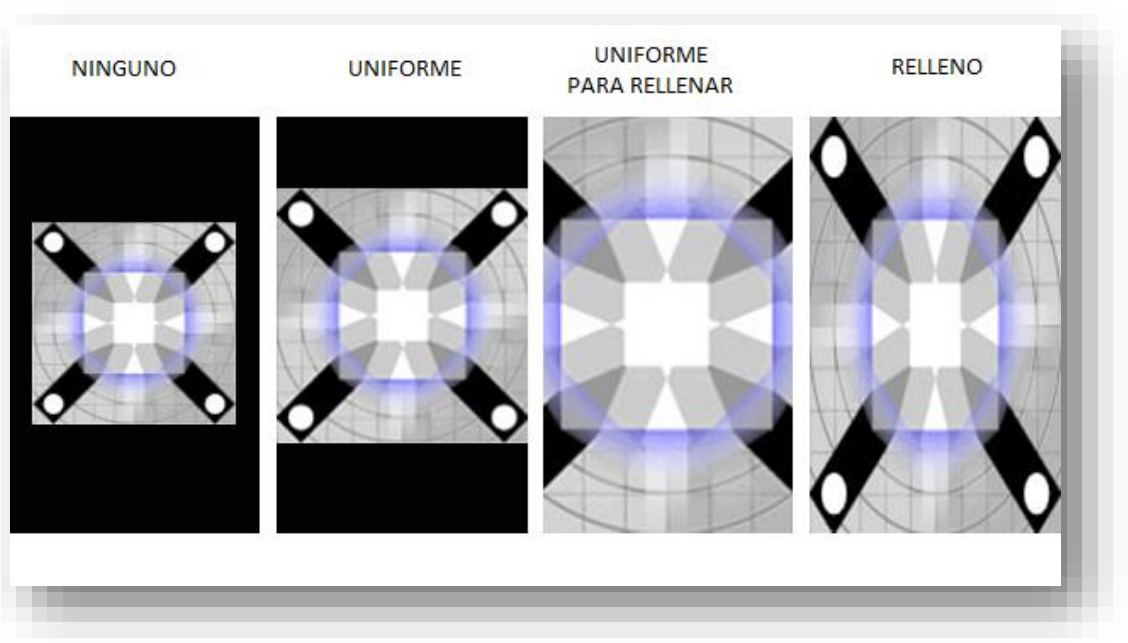


Ilustración 4. Propiedades de escalado de imagen

Establecer opacidad

Sobre las imágenes se pueden aplicar diferentes máscaras relacionadas con la opacidad de forma que podamos aplicar degradados con efectos, por ejemplo desvaneciendo sólo los bordes de la misma.

Para crear una máscara de opacidad es necesario aplicar un objeto *Brush* [8][9] a la propiedad *OpacityMask* de un elemento. El objeto *Brush* se asigna al elemento y el valor de cada pixel se utiliza para determinar la opacidad resultante.

Una máscara de opacidad permite que ciertas porciones de un elemento pasen a un estado transparente o parcialmente transparente. Para poder aplicarla es necesaria asignar su contenido a un elemento. Por ejemplo:

```
<Button Height="87" Width="145" Background="{StaticResource
PhoneContrastBackgroundBrush}">
    <Button.OpacityMask>
        <ImageBrush Stretch="Fill" ImageSource="icons/appbar.feature.video.rest.png"/>
    </Button.OpacityMask>
</Button>
```

El canal alfa de cada uno de los píxeles del objeto *Brush* se utilizan para determinar la opacidad resultante del elemento.

Otro ejemplo, donde se define la opacidad programáticamente:

```
Button btn = new Button();
btn.Height = 100;
btn.Width = 100;
btn.Background = Application.Current.Resources["PhoneContrastBackgroundBrush"] as
SolidColorBrush;
ImageBrush image = new ImageBrush();
```

```
image.ImageSource = new BitmapImage( new Uri("icons/appbar.feature.email.rest.png",
UriKind.Relative));
image.Stretch = Stretch.Fill;
btn.OpacityMask = image;

this.ContentPanel.Children.Add(btn);
```

Rotar imagen

Dentro del espacio de nombres *System.Windows.Media* encontramos la clase *RotateTransform* que convierte un objeto en el sentido de las agujas del reloj alrededor de un punto especificado en un sistema de coordenadas x-y de 2D.

Para realizar esta rotación podríamos optar por hacerlo a través de XAML:

```
<Image.RenderTransform>
  <RotateTransform Angle="90" />
</Image.RenderTransform>
```

O bien a través del lenguaje C#

```
RotateTransform rt = new RotateTransform();
rt.Angle = 90;
image.RenderTransform = rt;
```

3.1.3 IOS

Las aplicaciones nativas de iPhone son escritas en Objective-C, un lenguaje de alto nivel, orientado a objetos e influenciado por el lenguaje C. Para desarrollar aplicaciones nativas, se necesita la SDK, disponible sólo para el sistema operativo Mac OS X. Consiste en diferentes frameworks, donde uno de los más notables es Cocoa Touch, un framework de interfaz de usuario para crear aplicaciones para dispositivos móviles que ejecutan el sistema operativo iOS. La SDK también dispone de un simulador para iPhone, que permite testear las aplicaciones sin la necesidad de un dispositivo físico.

El IDE más común utilizado para desarrollar aplicación iPhone es Xcode. Este incluye el compilador *gcc*, que da soporte al debug, así como el editor Interface Builder que facilita el diseño de una interface de usuario sin necesidad de escribir código. Sus interfaces se guardan como ficheros con extensión *.xib* y contienen XML que puede ser cargado programáticamente.

Dentro de la arquitectura iOS nos encontramos cuatro niveles de abstracción:

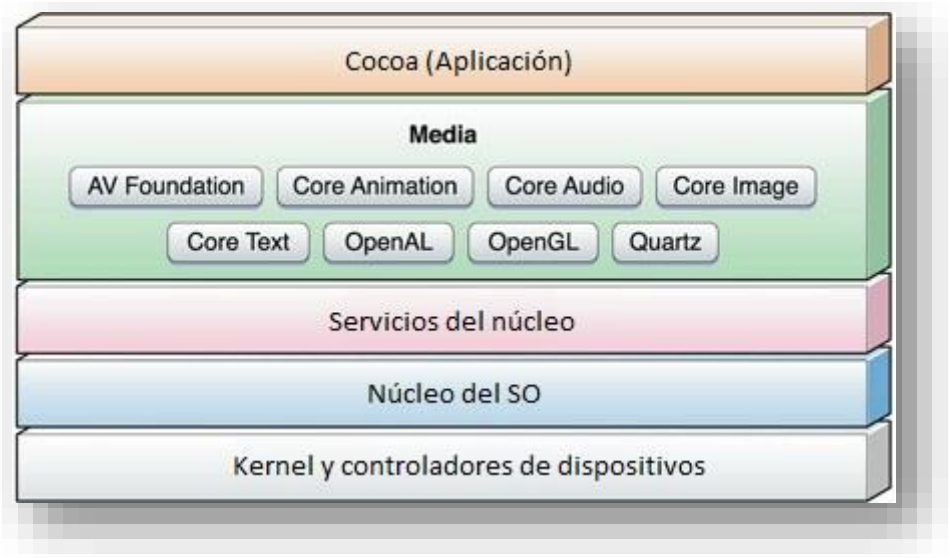


Ilustración 5. Niveles de abstracción en iOS

Dentro de la capa Media tenemos el framework Media Player [10] donde el desarrollador obtiene facilidades para reproducir vídeos, música o audio. Este framework proporciona acceso de solo lectura a la librería iPod, permitiendo buscar y reproducir elementos multimedia basados en audio sincronizado desde iTunes.

A través de la clase MPVolumeView se permite presentar un control al usuario para ajustar el sistema de salida audio y seleccionar las rutas de salida disponibles, como por ejemplo enviar audio a un dispositivo AirPlay.

Las aplicaciones pueden recibir y dar respuesta a eventos enviados por los reproductores de medios externos a través de los medios de comunicación de API del reproductor.

El protocolo MPPlayableContentDataSource proporciona un origen de datos al gestor de contenidos, y a través de la clase MPContentItem tenemos las especificaciones sobre los elementos que se pueden reproducir.

3.1.3.1 Gestión de audio y vídeo

Dentro del framework Media Library tenemos tres clases que nos van a permitir reproducir música desde nuestra aplicación. Son las siguientes:

MPMusicPlayerController [11]

Existen dos tipos de MPMusicPlayerController:

- Los reproductores de música en la propia aplicación ([MPMusicPlayerController applicationMusicPlayer]) que reproduce música sólo en la aplicación, sin afectar que esté el iPod incorporado.

- El reproductor de música iPod ([MPMusicPlayerController iPodMusicPlayer]) que emplea el iPod incorporado para reproducir música. En este caso la música se puede controlar desde el iPod y se mantiene reproduciendo incluso después de salir de la aplicación.

Entre los métodos de esta clase nos encontramos:

- *setQueueWithQuery – setQueueWithItemCollection:* Son utilizados para fijar una lista de reproducción.
- *Play, pause, stop, beginSeekingForward, beginSeekingBackward, endSeeking, skipToNextItem, skipToPreviousItem, skipToBeginning*: Son utilizados para controlar el reproductor.
- *currentPlaybackTime, nowPlayingItem, playbacksState, repeatMode, shuffleMode, volumen:* son propiedades del reproducto que se pueden utilizar para cambiar el comportamiento.

MPMediaItem

Representa un único ítem (ej. Una canción) dentro de la librería media. Es la propiedad *nowPlayingItem* del reproductor. Cada ítem tiene metadatos asociados a él, a los que se puede acceder invocando al método *valueForProperty.* Ejemplos: *MPMediaItemPropertyTitle* (título de la canción) o *MPMediaItemPropertyArtist* (artista que canta la canción).

MPMediaQuery

Se utiliza para realizar consultas sobre la librería iPod. Por ejemplo:

```
MPMediaQuery* query = [MPMediaQuery songsQuery];
[query addFilterPredicate:[MPMediaPropertyPredicate predicateWithValue:@"Led Zeppelin"
forProperty:MPMediaItemPropertyArtist comparisonType:MPMediaPredicateComparisonEqualTo]];
[query setGroupingType:MPMediaGroupingAlbum];
```

Estas sentencias crearían una consulta que busca todas las canciones cuyo artista sea “Led Zeppelin” y las agruparía por álbum. Todas las canciones resultado de la búsqueda pueden ser enviadas al objeto MPMusicPlayerController para reproducirlas

Para reproducir vídeo, disponemos de la clase MPMoviePlayerController [12], la cual gestiona la reproducción de un video ya sea desde un archivo local o como stream de datos.

Esta clase da soporte programáticamente, tanto a la reproducción de vídeos, como al control basado en el usuario a través de los botones suministrados por el reproductor de películas. La mayoría de los aspectos de la reproducción, se pueden controlar programáticamente a través de los métodos y propiedades del protocolo

MPMediaPlayback. Los métodos y propiedades de este protocolo, permiten al desarrollador reproducir, pausar, parar, avanzar y retrocede a través del contenido del vídeo, e incluso cambiar la velocidad de reproducción.

A la hora de añadir un reproductor dentro de nuestra aplicación, es importante redimensionar el frame (contenedor) correctamente, por ejemplo:

```
MPMoviePlayerController *player =
        [[MPMoviePlayerController alloc] initWithContentURL: myURL];
[player prepareToPlay];
[player.view setFrame: myView.bounds];  // player's frame must match parent's
[myView addSubview: player.view];
// ...
[player play];
```

Lo habitual es especificar el vídeo que se quiere reproducir en el momento en el que se crea un nuevo objeto MPMoviePlayerController. Aunque también se puede modificar el valor de la propiedad contentURL, lo que permite reutilizar el mismo objeto MPMoviePlayerController desde diferentes lugares, pero sólo un reproductor de vídeo a la vez puede reproducir un vídeo.

Para facilitar la creación de marcas de video o enlaces a capítulos dentro de vídeos de larga duración, la clase MPMoviePlayerController define métodos para la generación de imágenes en determinados momentos dentro de un vídeo.

3.1.3.2 Gestión de imágenes

A través de la clase UIImageView [13] podemos obtener un contenedor basado en una vista para mostrar una imagen o varias, pudiendo incluso incorporar animación a las mismas.

Cuando un objeto UIImageView muestra una de sus imágenes, el actual comportamiento se basa en las propiedades de la imagen y la vista. Si cualquiera de las propiedades: leftCapWidth o topCapHeight tiene un valor que no sea cero, entonces la imagen se estira de acuerdo a los valores de las propiedades. En cualquier caso la imagen se escala para ajustarse o posicionarse en la vista, acorde a la propiedad contentMode de la vista. Es recomendable que se utilicen imágenes del mismo tamaño, porque si las imágenes son de diferente tamaño, cada una va a ajustarse por separado. Del mismo modo, las imágenes que estén asociadas con un objeto UIImageView deberían utilizar la misma escala.

Establecer opacidad

La transparencia de una vista de la imagen se define a través de las propiedades, tanto de la propia imagen, como de la vista.

- Si se establece el flag *Opaque*, ya sea programáticamente o a través del inspector de Xcode, el canal alfa de la imagen se mezcla con el color de fondo de la vista, y la vista por si misma se presenta opaca. La configuración alfa de la vista se ignora.

- Si no está establecido el flag *Opacque,* el canal alfa de cada pixel se multiplica por la configuración alfa de la vista. El valor resultante determina la transparencia de ese pixel.

Rotar imagen

A través de la estructura de datos CGAffineTransform podemos aplicar transformaciones sobre una imagen. Una transformación permite aplicar un sistema de coordenadas e otro sistema de coordenadas. El escalado, rotación o traslación son las manipulaciones más utilizadas.

Se entiendo por transformaciones afines como un tipo especial de mapeo que preserva las líneas paralelas dentro de unas coordenadas, pero no preserva necesariamente la longitud ni los ángulos.

Por ejemplo, si queremos aplicar una transformación afín para rotar una imagen, podemos aplicar la función CGAffineTransformMakeRotation:

```
float degrees = 20; //the value in degrees
imageView.transform = CGAffineTransformMakeRotation(degrees * M_PI/180);
```

3.2 GESTIÓN DE RECURSOS MULTIMEDIA EN HERRAMIENTAS DE TERCEROS

Dentro de los entornos de desarrollo multiplataforma, podemos decir que existen diferentes enfoques [14]:

Nativos

Cada una de las principales plataformas móviles utilizan su propio lenguaje de programación, de modo que si deseamos desarrollar aplicaciones nativas, será necesario tener un desarrollador especializado en cada plataforma, o bien la misma persona termina el desarrollo en una plataforma y comience con la siguiente. Ambas opciones incrementan considerablemente los costes y el tiempo.

A pesar de que puede suponer un incremento de coste y tiempo para las empresas, el desarrollo de aplicaciones nativas es recomendable por los siguientes motivos:

- En las aplicaciones móviles, donde se utiliza un porcentaje considerable de los recursos del sistema (por ejemplo los juegos), es recomendable que su desarrollo se dirija hacia una implementación nativa ya que proporcionará el mejor aprovechamiento del sistema.
- Cada una de las plataformas proporciona su guía de mejoras prácticas en el diseño de las aplicaciones móviles, por ejemplo la ubicación de los menús en una posición u otra. El hecho de que se desarrolle una aplicación nativa, mejora la sensación

que el usuario pueda tener sobre la aplicación, ya que si los controles se visualizan en base a las recomendaciones de la plataforma, el usuario ve esa aplicación como una extensión de su dispositivo, en lugar de verlo como un objeto extraño.

- Las aplicaciones nativas proporcionan una mejor experiencia de interacción al tacto. Los objetos de la aplicación (gestos, desplazamientos,..) se comportan según lo esperado, y el usuario está familiarizado con ellos. Se omite cualquier capa intermedia mantiene el rendimiento del dispositivo, evitando los retardos entre que el usuario interaccione con el sistema y este le proporcione una respuesta.

Web móvil

El primer impulso que ha tenido el desarrollo multiplataforma en dispositivos móviles, ha sido a través de la web móvil. El hecho de que los desarrolladores trabajen con HTML5, CSS3 y JavaScript proporciona a este enfoque un grado de familiaridad que facilita la rapidez en el desarrollo.

El inconveniente de este tipo de enfoques es que los desarrolladores están restringidos al nivel de interacción hardware que es compatible con HTML5 y el navegador. Por ejemplo características como el uso de la cámara sobre la que no podemos acceder en la mayoría de los navegadores podría ser un factor decisivo para impedir elegir este tipo de enfoque.

Otro inconveniente puede ser el hecho de que la interrupción del servicio y la mala señal pueden evitar que los sitios web móviles sean accesibles a los usuarios cuando necesitan información. A menos que se almacenen en caché los datos, se requerirá que el usuario vuelva a cargar todos los datos cada vez que visitan el sitio Web. La capacidad de respuesta de un sitio web móvil también puede estar limitada por la lenta ejecución de JavaScript en los dispositivos móviles.

Web híbrida

Este tipo de aplicaciones surge a través del cruce entre las aplicaciones web móviles con empaquetado de aplicaciones nativas. Se aprovechan los conocimientos que todo desarrollador Web tiene sobre HTML5, CSS3 y JavaScript y lo pueden complementar con una API que permite acceder a características de hardware.

Uno de los inconvenientes de este enfoque es que las aplicaciones híbridas son representadas a través de los navegadores, por lo que no es posible utilizar componentes de interfaz de usuario realmente nativas. De hecho hay casos en los que debido a esta falta de poder incrustar componentes de IU nativa, algunos desarrolladores deciden crear sus propios estilos para sus aplicaciones.

Otro inconveniente puede ser la demora que existe entre que se presenta una nueva característica sobre un dispositivo móvil, hasta que esta es implementada para ser accesible desde una aplicación híbrida. Para solventar esto, muchas de las plataformas

ofrecen la posibilidad de desarrollar plugins que sirvan de puente hacia la nueva característica.

Cross compiled

Este enfoque permite el desarrollo de aplicaciones móviles multiplataforma a través de un lenguaje de programación. Gracias a este enfoque los desarrolladores pueden escribir una aplicación en un lenguaje y compilarlo independientemente para cada plataforma de destino, reduciendo tiempo de desarrollo.

Dependiendo de la plataforma se podrá acceder a los elementos de la interfaz nativa. En cuanto el rendimiento, es mejor que en las aplicaciones híbridas pero no siempre tan bueno como las aplicaciones nativas por el hecho de que a menudo se requiere la carga de muchas bibliotecas en memoria.

Para cada una de las plataformas planteadas en este estudio se comenta brevemente los siguientes apartados:

- **Enfoque** que se sigue de entre los descritos anteriormente: nativo, web móvil, web híbrida o cross compiled.
- **Tipo de aplicación generada y tecnologías utilizadas**.
- **Interfaz y APIS**. En este apartado se indica si la interfaz es nativa o no, y las diferentes APIs soportadas.
- **Desarrollo.** Se especifican las herramientas disponibles para que el usuario desarrolle las aplicaciones en esta plataforma.
- **Documentación y comunidad.**
- **Generación de la aplicación.** Herramientas necesarias y proceso a seguir para generar la aplicación final.

Posteriormente nos centramos en cómo gestionan los recursos multimedia cada una de estas plataformas.

3.2.1 APACHE CORDOVA

3.2.1.1 Enfoque

Apache Cordova [15] es un framework que sigue un enfoque de web híbrida. Cordova está disponible para las plataformas iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada y Symbian. Apache Cordova se presenta liberado y de código abierto bajo la licencia de Apache, versión 2.0.

3.2.1.2 Tipo de aplicación generada y tecnologías utilizadas

Utiliza HTML, CSS y JavaScript junto con librerías propias, para generar aplicaciones que se ejecutarán sobre el motor del navegador Web. No está pensado para desarrollar aplicaciones que requieran de potentes cálculos ni que presenten animaciones 3D. Tampoco proporciona soporte para base de datos, en caso de ser necesaria esta última, habrá que utilizar las APIs de persistencia de HTML5.

3.2.1.3 Interfaz y APIs

Apache Cordova es un conjunto de APIs vinculadas con un dispositivo concreto, que permite a los desarrolladores, a través del lenguaje JavaScript, acceder a las funcionalidades nativas de cada dispositivo tales como la cámara o el acelerómetro.

El diseño de la interfaz en Apache Cordova se realiza mediante HTML/CSS/JavaScript que combinado con un framework de interfaz de usuario como puede ser JQuery Mobile [16], Dojo Mobile [17] o Sencha Touch [18], permite el desarrollo de Apps.

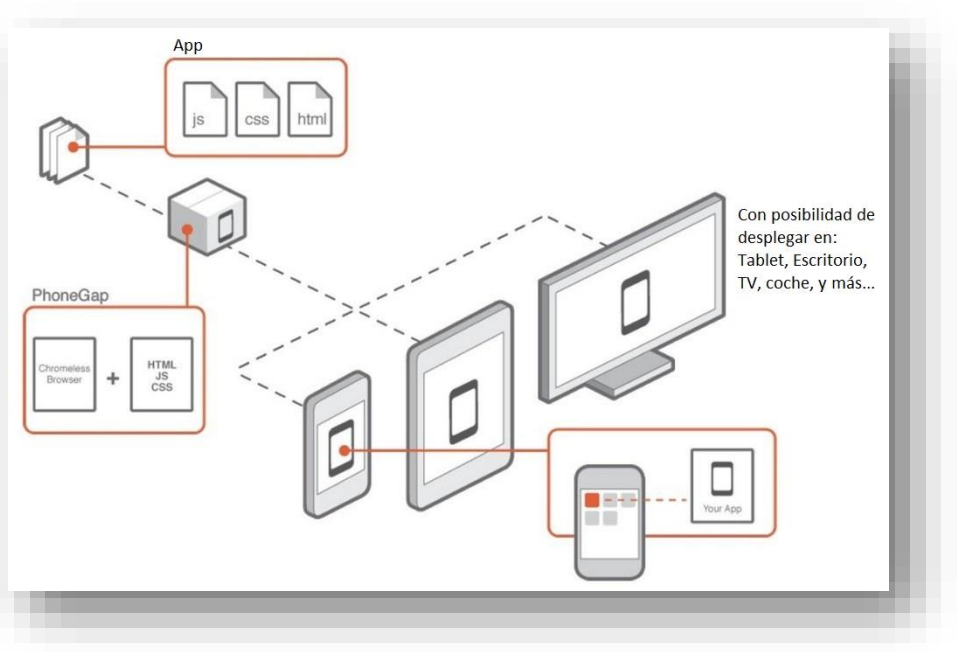


Ilustración 6. Arquitectura de Apache Córdova

Al utilizar las APIs de Cordova, una app puede ser desarrollada sin utilizar ningún código nativo, en su lugar se utiliza código HTML, CSS y JavaScript que se alojará internamente en la App. Como estas APIs son consistentes entre las diferentes plataformas y basadas en estándares Web, la App puede ser utilizada entre las diferentes plataformas, con los mínimos cambios.

3.2.1.4 Desarrollo

Según el proyecto que se vaya a desarrollar, el programador tendrá que decidir que librerías necesita, el IDE más apropiado o el framework táctil a utilizar.

3.2.1.5 Documentación y comunidad

La gran documentación existente [19], ofrece a los desarrolladores un buen punto de entrada a la plataforma. En ella se incluyen los pasos necesarios para configurar el entorno de desarrollo, así como un gran número de herramientas disponibles.

En cuanto a la comunidad, existe un grupo de Google, canales en IRC así como una extensa Wiki donde se tratan temas de configuración y desarrollo.

3.2.1.6 Generación

Las Apps que utilizan Cordova, pueden ser empaquetadas utilizando las correspondientes SDKs, y pueden estar disponibles para la instalación desde la App Store de cada plataforma.

Además, se puede generar el ejecutable específico para cada plataforma a través de la herramienta PhoneGap Build. El primer paso es subir el proyecto al servicio PhoneGap Build [20]. Una vez subido se configuran para cada plataforma* sus certificados, claves, etc, y por último se genera la aplicación, que ya estará preparada para ser distribuida.

*Para la plataforma iOS es necesario disponer de un Mac con las herramientas de desarrollo instaladas.

3.2.1.7 Gestión de audio y vídeo

Para la gestión de audio, se puede utilizar el plugin media [20.a] que permite grabar y reproducir archivos de audio en un dispositivo. Los parámetros que recibe un objeto de este tipo son los siguientes:

```
var media = new Media(src, mediaSuccess, [mediaError], [mediaStatus]);
```

Donde cada uno especifica lo siguiente:

- *src:* Especifica una URI con el contenido del audio
- *mediaSuccess:* (opcional) Esta es la llamada de vuelta que se ejecuta después de que un objeto Media haya completado la reproducción, grabación o acción de parada actual.
- *mediaError:* (opcional) Llamada de vuelta que se ejecuta si ocurre un error.
- *mediaStatus:* (opcional) Llamada de vuelta que se ejecuta para indicar cambios de estado.

Los métodos que soporta este objeto son los siguientes:

- *media.getCurrentPosition:* Devuelve la posición actual dentro de un archivo de audio.

- *media.getDuration:* Duración de un archivo de audio.
- *media.play:* Comienzo o pausa sobre la reproducción de un archivo de audio.
- *media.pause:* Pausa la reproducción.
- *media.release:* Libera los recursos de audio del sistema operativo.
- *media.seekTo:* Mueve la posición a un punto determinado.
- *media.setVolume:* Fija el volumen.
- *media.startRecord:* Comienza la grabación de un archivo de audio.
- *media.stopRecord:* Finaliza la grabación.
- *media.stop:* Finaliza la reproducción.

Ejemplo para reproducir un archivo de audio:

```
// Play audio
function playAudio(url) {
    // Play the audio file at url
    var my media = new Media(url,
        // success callback
        function() {
            console.log("playAudio():Audio Success");
        },
        // error callback
        function(err) {
            console.log("playAudio():Audio Error: "+err);
    });

    // Play audio
    my_media.play();

    // Mute volume after 2 seconds
    setTimeout(function() {
        my media.setVolume('0.0');
    }, 2000);

    // Set volume to 1.0 after 5 seconds
    setTimeout(function() {
        my media.setVolume('1.0');
    }, 5000);
}
```

Para la gestión de video, se puede utilizar el plugin VideoPlayer [20.b]. Este plugin permite mostrar videos desde la aplicación de Apache Cordova a través de un reproductor.

La instalación nos crea el objeto *window.plugins.videoPlayer.* Para utilizarlo basta con invocar al método *play()*

```
window.plugins.videoPlayer.play("http://path.to.my/video.mp4");
```

3.2.1.8 Gestión de imágenes

Para manipular imágenes en Apache Cordova, se puede hacer uso de toda la funcionalidad que aporta HTML, JavaScript y CSS. Por ejemplo si queremos aplicar opacidad sobre una imagen podríamos aplicar el atributo CSS *opacity* sobre esa imagen. O bien si queremos realizar una manipulación más compleja sobre imágenes, podríamos buscar si ya está

implementada esa funcionalidad en algún plugin desarrollado para la plataforma, o directamente desarrollarlo nosotros mismos.

3.2.2 XAMARIN

3.2.2.1 Enfoque

Xamarin [21] es un framework que sigue un enfoque cross compiled. Permite crear aplicaciones móviles para las plataformas iOS, Android y Windows, accediendo a todas las funciones nativas de los dispositivos a través de sus APIs.

Xamarin se distribuye con una licencia Open Source LGPLv2.

3.2.2.2 Tipo de aplicación generada y tecnologías utilizadas

Xamarin utiliza C# como lenguaje de programación para desarrollar una aplicación móvil completa. Gracias a ello podemos reutilizar librerías existentes escritas en .Net, de modo que nos sirvan para iOS, Android y Windows.

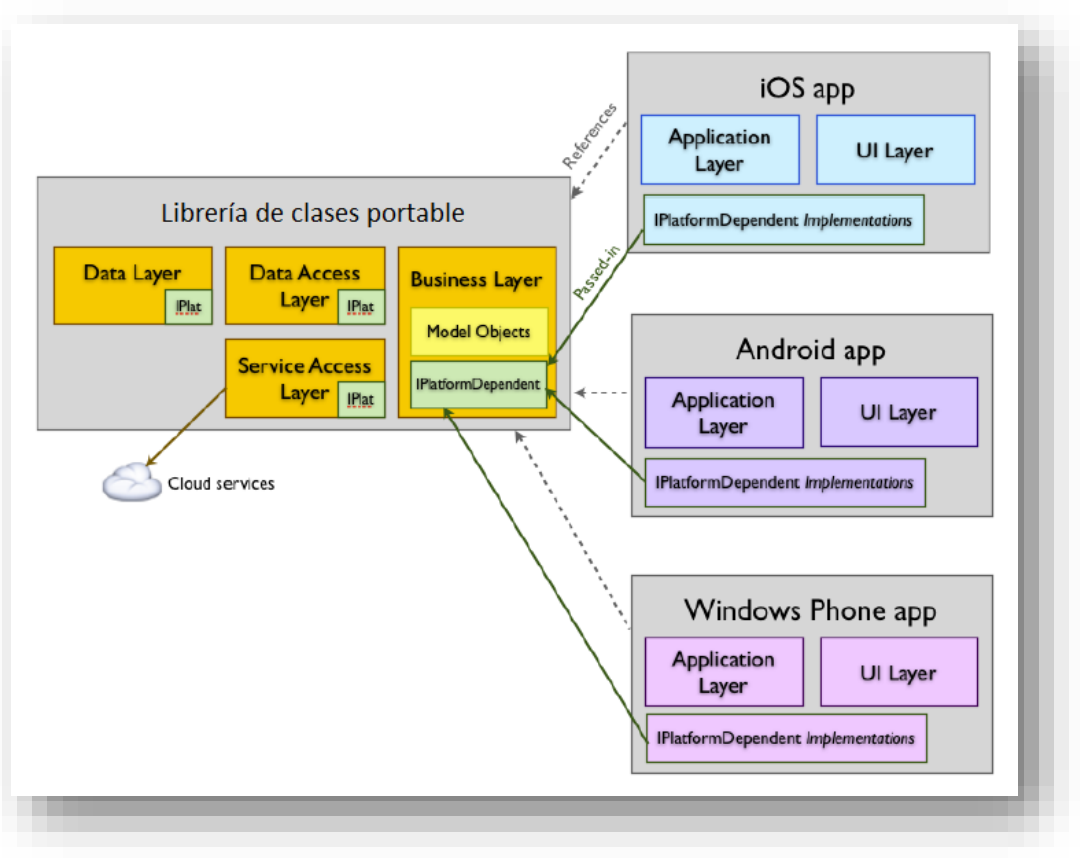


Ilustración 7. Reutilizar librerías entre plataformas

Xamarin simplifica el desarrollo dando soporte a todas las características desde la creación hasta la publicación, pasando por el diseño de la interfaz, debug, compilación, test, empaquetado y despliegue del proyecto. [22]

3.2.2.3 Interfaz y APIs

Xamarin permite los enlaces nativos, que son un mecanismo disponible para los proyectos de iOS y Android que permiten acceder a bibliotecas nativas desde la aplicación móvil. Esto se hace mediante la creación de un enlace o asignación, de las llamadas que se realizan desde C# a la biblioteca nativa. Uno de los beneficios de utilizar los enlaces nativos es que se pueden incorporar librerías externas, lo que permite reducir tiempos de desarrollo. Por ejemplo si tenemos una aplicación en el que uno de los requisitos es el reconocimiento óptico (OCR), en lugar de escribir todo el código, se podría incluir a través de enlaces nativos una librería de código abierto que cubra ese requisito.

Xamarin utiliza una librería principal que contiene todas las reglas de negocio, manejo de datos y servicios de forma que pueda ser compartido por las aplicaciones móviles de cada plataforma. Cada plataforma también tiene su propio código fuente para manejar la interfaz de usuario y la capa de aplicación. También ofrece la posibilidad de personalizar funcionalidades y diseños en cada plataforma, así como tener una sola base de código para dicha interfaz.

Ilustración 8. Plataforma Xamarin

Las aplicaciones que desarrollemos con Xamarin tienen un look and feel nativo.

Xamarin mantiene actualizadas sus APIs con las últimas versiones de Google y Apple para acceder a las funcionalidades de los dispositivos.

3.2.2.4 Desarrollo

Los desarrolladores habituados a programar en .Net, van a poder desarrollar aplicaciones iOS y Android con Visual Studio, pudiendo así reutilizar los conocimientos sobre esta herramienta, así como sus librerías y estándares de diseño.

También se dispone del IDE Xamarin Studio [23] que ofrece un entorno de desarrollo propio.

3.2.2.5 Documentación y comunidad

La plataforma aporta completas guías generales sobre el desarrollo multiplataforma, instalación, introducción a Xamarin studio, etc. También presenta una completa guía de cada una de las plataformas, así como foros y ejemplos prácticos [24].

3.2.2.6 Generación

Xamarin.iOS produce los instaladores para la plataforma iOS a través de la compilación AOT *(Compilación AOT: "Ahead of time", se refiere al acto de compilar un lenguaje intermedio a una librería ejecutable para una plataforma específica, antes de que el código o programa sea ejecutado)*, mientras que Xamarin.Android utiliza la compilación JIT *(Compilación JIT: "just in time", consiste en compilar el código existente en tiempo de ejecución, es decir, cuando se está ejecutando el programa)* para optimizaciones sofisticadas. En ambos casos la aplicación es convertida en el binario entendible por cada plataforma.

3.2.2.7 Gestión de audio y vídeo

Para la plataforma Android podemos reproducir audio [24.a] con la clase MediaPlayer. Se requiere Android 2.0 (Nivel de API 5) o superior.

La clase MediaPlayer puede reproducir tanto archivos locales como remotos, a través de una ruta donde se indique su origen. Dado que invocar a MediaPlayer con el estado incorrecto causa una excepción, es importante interactuar con esta clase en el siguiente orden:

1. Instanciar un nuevo objeto *MediaPlayer*.
2. Configurar el archivo para reproducir lo especificado a través del método *SetDataSource*.
3. Invocar al método *Prepare* para inicializar el reproductor.
4. Invocar al método *Start* para comenzar la reproducción

Por ejemplo:

```
protected MediaPlayer player;
public void StartPlayer(String  filePath)
{
  if (player == null) {
    player = new MediaPlayer();
  } else {
    player.Reset();
    player.SetDataSource(filePath);
    player.Prepare();
    player.Start();
  }
}
```

Para la reproducción de audio en la plataforma iOS, se puede utilizar el objeto MPMoviePlayerController ya descrito en el apartado de la SDK de iOS.

En cuanto a la reproducción de vídeos en la plataforma Android [24.b] podemos optar por cualquiera de estos pasos:

- Añadir un objeto *VideoView* a una disposición en XML.

```
<VideoView android:id="@+id/SampleVideoView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
</VideoView>
```

- Obtener una referencia al objeto *VideoView* desde el código.

```
var videoView = FindViewById<VideoView> (Resource.Id.SampleVideoView);
```

- Asociar un objeto Android.Net.URI para el video y construir el objeto *VideoView* a partir de él

```
var uri = Android.Net.Uri.Parse ("video url goes here");
videoView.SetVideoURI (uri);
```

- Reproducir el vídeo

```
videoView.Start();
```

*Sobre las opciones de proyecto hay que añadir un fichero android manifest, y bajo los permisos requeridos hay que establecer los permisos de Internet.

Sobre la reproducción de vídeos en la plataforma iOS [24.c] podemos utilizar la clase MPMoviePlayerController. El siguiente fragmento de código presenta un ejemplo sobre cómo asociar la reproducción de vídeo a través de esta clase, manejando el evento *TouchUpInside*:

```
MPMoviePlayerController moviePlayer;
playMovie.TouchUpInside += delegate {
    moviePlayer = new MPMoviePlayerController (NSUrl.FromFilename ("sample.m4v"));
```

```
    View.AddSubview (moviePlayer.View);
    moviePlayer.SetFullscreen (true, true);
    moviePlayer.Play ();
} ;
```

3.2.2.8 Gestión de imágenes

En la plataforma Android, las imágenes se gestionan a través la clase ImageView, pudiendo aplicar las transformaciones ya comentadas en el apartado de gestión de imágenes de la SDK de Android.

Lo mismo ocurre en la plataforma iOS, donde también se pueden aplicar las mismas transformaciones que las ya comentadas dentro del apartado SDK de iOS. Por ejemplo, si queremos rotar una imagen:

```
// nombramos los objetos
UIImage  image;
UIImageView _imageView;
// los creamos
 image = UIImage.FromFile("monkey.png");
 imageView = new UIImageView(new RectangleF(50,50,100,100));
 imageView.ContentMode = UIViewContentMode.ScaleAspectFit;
_imageView.Image = _image;
// aplicamos la propiedad de transformación sobre el objeto
 imageView.Transform = CGAffineTransform.MakeRotation((float)Math.PI/4);
View.AddSubview (_imageView);
```

3.2.3 RHOMOBILE

3.2.3.1 Enfoque

Rhomobile es una plataforma para el desarrollo multiplataforma de aplicaciones móviles que a través de una serie de APIs permite crear una aplicación nativa para cada uno de los dispositivos: Android, iOS, BlackBerry, Nokia y Windows Mobile.

A diferencia de PhoneGap, Rhomobile pertenece al grupo de frameworks cuya finalidad es crear una aplicación nativa.

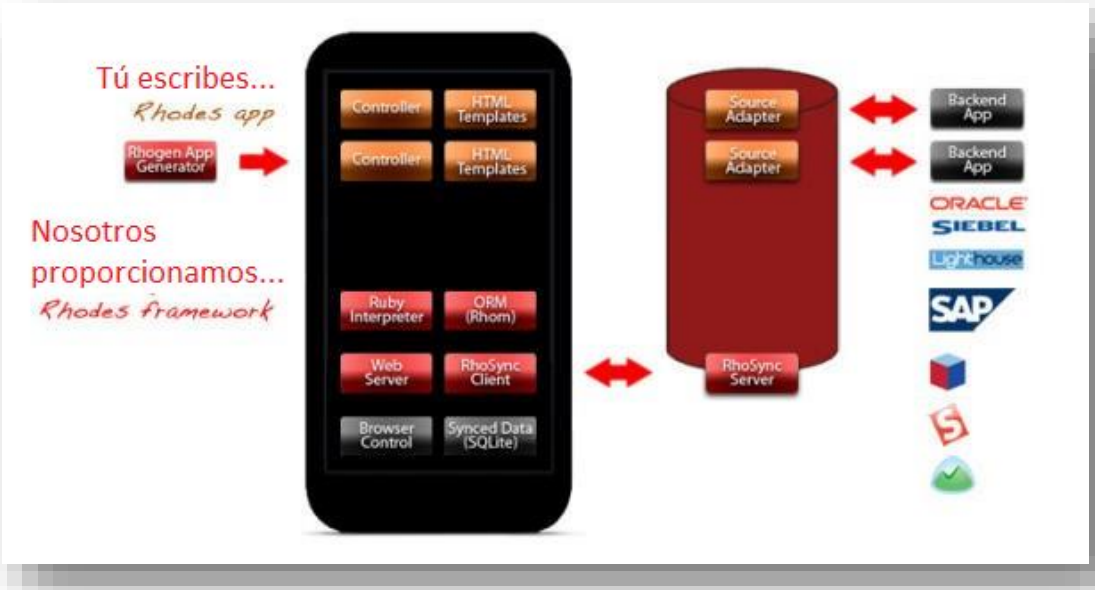


Ilustración 9. Plataforma Rhomobile

Esta plataforma consta de una serie de productos que permiten gestionar todo lo relacionado con el desarrollo de aplicaciones. En concreto Rhodes es el framework MVC con el que se van a desarrollar y generar las aplicaciones. El proceso de instalación y ejecución de las aplicaciones nativas, es muy similar al seguido para el desarrollo Web.

Rhodes se distribuye libre y open source bajo la licencia MIT.

3.2.3.2 Tipo de aplicación generada y tecnologías utilizadas

Dentro del patrón MVC de este framework, distinguimos:

- **View:** Podemos utilizar HTML, CSS y JavaScript para definir las vistas. Este código se entremezcla con Ruby para generar ficheros ERB (Embedded Ruby) y conseguir así añadir lógica de negocio a las mismas, de manera similar a como se hace en ASP o PHP.
- **Controller:** Se utiliza código Ruby para la definición de controladores.
- **Model:** Del mismo modo que para el controlador, en el modelo hacemos uso de scripts en Ruby.

3.2.3.3 Interfaz y APIs

La interfaz gráfica de una aplicación Rhomobile no tiene aspecto enteramente nativo. El acceso a las diferentes funcionalidades nativas del teléfono se hace a través de una serie de APIs. Para el desarrollo de aplicaciones, las llamadas a estas APIs van a ser independientes de la plataforma destino sobre la que se ejecuten.

3.2.3.4 Desarrollo

Rhomobile no incluye su propio IDE de desarrollo. Se puede desarrollar en cualquier editor que soporte HTML y Ruby. RhoStudio [25] es un plugin de Rhodes para Eclipse, que se encuentra en fase Beta.

3.2.3.5 Documentación y comunidad

Rhomobile no tiene detrás una comunidad tan potente como otras plataformas, aunque en la página oficial existen gran cantidad de tutoriales y guías que permiten conocer los principales aspectos de la herramienta.

También existe un grupo en Google Groups donde los desarrolladores pueden interactuar con otros miembros para resolver dudas, compartir experiencias, etc.

3.2.3.6 Generación

Una vez que la aplicación haya sido desarrollada, se genera el código nativo de la plataforma que haya sido seleccionada, a través de código Ruby. Para ello se hace uso del producto RhoHub que se encarga de generar las aplicaciones.

3.2.3.7 Gestión de audio y vídeo

La reproducción de archivos de audio se realiza mediante el API MediaPlayer [25.a]. Para habilitar este API se debe incluir la siguiente extensión en el fichero *build.yml*

```
extensions: ["mediaplayer"]
```

Un ejemplo de utilización:

```
var platform = Rho.System.platform;
var audiolocation = "";

if (platform == "WINDOWS")
{
    audiolocation = "\\thermo.wav";
}
else if (platform == "ANDROID")
{
    audiolocation = "/mnt/sdcard/test.mp3";
}

Rho.Mediaplayer.start(audiolocation);
```

Al método *Mediaplayer.start* se le pasa la ruta apuntando al archivo de audio. En la plataforma iOS cuando se pasa una ruta a un archivo remoto en un servidor, se abre el navegador predeterminado para reproducirlo.

La plataforma Rhodes también puede reproducir archivos de audio a través del navegador. De manera que los desarrolladores pueden añadir un link apuntando al archivo de audio. Por ejemplo:

```
<a
href="http://videos3.netprofesseur.com/1/videos/130/BRICOLAGE Changer un joints.mp4">Play
mp4 online video</a>
```

Para reproducir vídeo se utiliza una lógica semejante al audio:

```
var platform = Rho.System.platform;
var videolocation = "";

if (platform == "WINDOWS")
{
    videolocation = "\\test.mp4";
}
else if (platform == "ANDROID")
{
    videolocation = "/mnt/sdcard/test.mp4";
}

Rho.Mediaplayer.startvideo(videolocation);
```

3.2.3.8 Gestión de imágenes

Para manipular imágenes en esta plataforma, se puede hacer uso de toda la funcionalidad que aporta HTML, JavaScript y CSS. Por ejemplo si queremos aplicar opacidad sobre una imagen podríamos aplicar el atributo CSS *opacity* sobre esa imagen. O bien si queremos realizar una manipulación más compleja sobre imágenes, podríamos buscar si ya está implementada esa funcionalidad en algún plugin desarrollado para la plataforma, o directamente desarrollarlo nosotros mismos.

3.2.4 TITANIUM

3.2.4.1 Enfoque

Titanium es un framework open-source que permite crear, ejecutar y empaquetar aplicaciones nativas para Android, iOS, Windows Phone y BlackBerry a través del uso de sus APIs propias.

Titanium se presenta con una licencia open source de Apache 2.0. También hay disponibles licencias comerciales y empresariales.

3.2.4.2 Tipo de aplicación generada y tecnologías utilizadas

Ilustración 10. Plataforma Titanium

Dentro de la arquitectura de Titanium encontramos 4 capas que nos permiten crear nuestra aplicación:

- Capa 1: Se forma con el código de nuestra aplicación HTML, CSS, JavaScript.
- Capa 2: Son las APIs que nos permiten acceder a la funcionalidad nativa.
- Capa 3: Puente que une el código Web en código nativo de la plataforma donde se despliegue la aplicación.
- Capa 4: Conjunto de instrucciones en tiempo de ejecución que empaquetan la aplicación para ser distribuida.

El lenguaje fundamental dentro de una aplicación Titanium es JavaScript. Las aplicaciones hechas con este framework se ejecutan sobre un motor JavaScript que invoca a las APIs nativas.

3.2.4.3 Interfaz y APIs

Titanium utiliza una API JavaScript que enlaza el código Web de la aplicación en su correspondiente código nativo de la plataforma. De esta forma podremos tener aplicaciones cuyo interfaz es igual al de una aplicación nativa de esa plataforma.

Del mismo modo, podremos acceder a las funcionalidades del teléfono, a través de unas APIs que nos permitan el acceso al GPS, cámara, reproducción de vídeo, etc.

3.2.4.4 Desarrollo

La plataforma Titanium consta de un IDE denominado Titanium Studio para el desarrollo de aplicaciones, permitiendo el desarrollo en un único lenguaje y la publicación automática en las plataformas nativas.

El entorno de desarrollo está basado en el IDE Eclipse, que es el entorno más utilizado por la comunidad de desarrolladores Java.

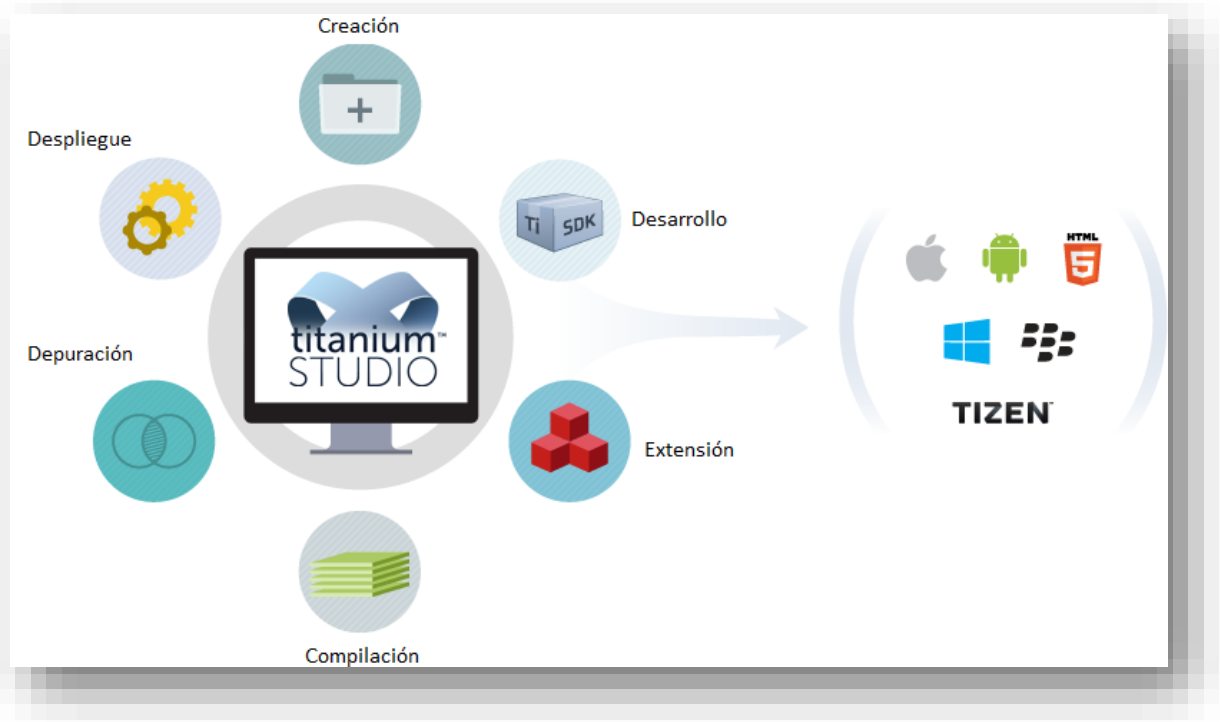


Ilustración 11. Titanium Studio

La plataforma Appcelerator Titanium es abierta y permite el desarrollo de aplicaciones para diferentes entornos a través del lenguaje JavaScript. La plataforma está compuesta por un SDK de código abierto, un IDE de desarrollo basado en Eclipse, un framework MVC llamado Alloy y un conjunto de servicios en la nube.

3.2.4.5 Documentación y comunidad

Titanium pone a disposición del desarrollador multitud de guías y tutoriales que permiten conocer a fondo la plataforma. Todo ello junto a un gran número de ejemplos englobados la mayoría de ellos en *Kitchen Sink* [26] un proyecto que muestra casos de uso para casi todas las opciones disponibles dentro del API de la SDK de Titanium Mobile.

3.2.4.6 Generación

Una vez desarrollada la aplicación, la generación de la misma se realiza a través de Titanium Studio. Este último incorpora las herramientas de generación incluidas en su predecesora Titanium Developer, además de la posibilidad de ejecutar el simulador.

En el caso de generar el código para iOS, es necesario disponer de un Mac con las herramientas de desarrollo instaladas, además de seguir los pasos necesarios: darse de alta como desarrollador, obtener licencia, obtener certificados, firmar fuentes, etc.

3.2.4.7 Gestión de audio y vídeo

En Titanium, un objeto reproductor de audio se utiliza para el streaming de audio en el dispositivo y para el control de bajo nivel en la reproducción de audio.

Para crear un reproductor de audio se utiliza el método *Titanium.Media.createAudioPlayer.* Por ejemplo:

```
var win = Titanium.UI.createWindow({
    title:'Audio Test',
    backgroundColor:'#fff',
    layout: 'vertical'
});

var startStopButton = Titanium.UI.createButton({
    title:'Start/Stop Streaming',
    top:10,
    width:200,
    height:40
});

var pauseResumeButton = Titanium.UI.createButton({
    title:'Pause/Resume Streaming',
    top:10,
    width:200,
    height:40,
    enabled:false
});

win.add(startStopButton);
win.add(pauseResumeButton);

// allowBackground: true on Android allows the
// player to keep playing when the app is in the
// background.
var audioPlayer = Ti.Media.createAudioPlayer({
    url: 'www.example.com/podcast.mp3',
    allowBackground: true
});

startStopButton.addEventListener('click',function() {
    // When paused, playing returns false.
    // If both are false, playback is stopped.
    if (audioPlayer.playing || audioPlayer.paused)
    {
        audioPlayer.stop();
        pauseResumeButton.enabled = false;
        if (Ti.Platform.name === 'android')
        {
            audioPlayer.release();
        }
    }
    else
    {
        audioPlayer.start();
        pauseResumeButton.enabled = true;
    }
});
```

```
pauseResumeButton.addEventListener('click', function() {
    if (audioPlayer.paused) {
        audioPlayer.start();
    }
    else {
        audioPlayer.pause();
    }
});

audioPlayer.addEventListener('progress',function(e) {
    Ti.API.info('Time Played: ' + Math.round(e.progress) + ' milliseconds');
});

audioPlayer.addEventListener('change',function(e)
{
    Ti.API.info('State: ' + e.description + ' (' + e.state + ')');
});

win.addEventListener('close',function() {
    audioPlayer.stop();
    if (Ti.Platform.osname === 'android')
    {
        audioPlayer.release();
    }
});

win.open();
```

En función de la plataforma para la que desarrollemos, la documentación oficial de la herramienta nos ofrece recomendaciones específicas de cada una de ellas, tanto para la gestión de audio, como para en cualquier otro tipo de cometidos.

Por ejemplo, dentro del ámbito de la gestión de audio, nos encontramos en la documentación las siguientes recomendaciones:

Notas en la implementación sobre Android

Cuando desarrollemos para la plataforma Android, se debe invocar al método *reléase* cuando haya terminado la reproducción de un archivo de audio. De esta forma se detiene el búfer de datos de audio y se liberan recursos del sistema.

Notas en la implementación sobre iOS

En la plataforma iOS se puede controlar la forma en la que el flujo de audio interactúa con otros sonidos del sistema, a través de la clase *Titanium.Media.audioSessionMode.*

En cuanto a la reproducción de vídeos, se utiliza la clase *Titanium.Media.VideoPlayer,* que es un control nativo que reproduce vídeos almacenados en el dispositivo o bien por streaming desde un servidor Web. El reproductor puede ocupar la pantalla completa o puede ser utilizado como una vista que puede ser añadida a otras vistas.

En la plataforma Android, el reproductor de vídeo no puede intercambiar modos. Para crear un productor a pantalla completa, se debe especificar el atributo *fullscreen: true* en el momento de su creación. Este reproductor se crea como una *Activity* en lo más alto de la pila de *Activities.* A diferencia de una vista normal, este reproductor a pantalla completa

aparece tan pronto como se crea. El usuario puede cerrar el reproductor pulsando sobre el botón Atrás.

En la plataforma iOS, el contenido de vídeo también se puede especificar como un objeto Blob. Su forma puede cambiar dinámicamente entre pantalla completa y modo estándar.

El siguiente ejemplo crea un reproductor de vídeo para reproducir un archivo local:

```
var vidWin = Titanium.UI.createWindow({
    title : 'Video View Demo',
    backgroundColor : '#fff'
});

var videoPlayer = Titanium.Media.createVideoPlayer({
    top : 2,
    autoplay : true,
    backgroundColor : 'blue',
    height : 300,
    width : 300,
    mediaControlStyle : Titanium.Media.VIDEO CONTROL DEFAULT,
    scalingMode : Titanium.Media.VIDEO SCALING ASPECT FIT
});

videoPlayer.url = 'movie.mp4';
vidWin.add(videoPlayer);
vidWin.open();
```

3.2.4.8 Gestión de imágenes

Para la gestión de imágenes se utiliza la clase *Titanium.UI.ImageView.*

Especificando las propiedades *width* o *height* se consigue escalar esta imagen hasta un tamaño máximo que no exceda la vista contenedora.

Por ejemplo, si queremos crear un objeto ImageView:

```
Ti.UI.backgroundColor = 'white';
var win = Ti.UI.createWindow();
var image = Ti.UI.createImageView({
  image:'/images/myimage.png'
});
win.add(image);
win.open();
```

Para manipular imágenes podemos hacer uso de toda la funcionalidad que aporta JavaScript y CSS. Por ejemplo si queremos aplicar opacidad sobre una imagen podríamos aplicar el atributo CSS *opacity* sobre esa imagen. O bien si queremos realizar una manipulación más compleja sobre imágenes, podríamos buscar si ya está implementada esa funcionalidad en algún plugin desarrollado para la plataforma [27], o directamente desarrollarlo nosotros mismos.

4 ELECCIÓN DE LA HERRAMIENTA DE TERCEROS

Tras el estudio realizado sobre las herramientas de terceros, podemos resumir los elementos más relevantes de cada una de las plataformas para concluir con la elección de una de ellas como plataforma de desarrollo.

	APACHE CORDOVA	XAMARIN	RHOMOBILE	TITANIUM	
SOPORTE ANDROID	✓	✓	✓	✓	
SOPORTE IOS	✓	✓	✓	✓	
SOPORTE WINDOWS PHONE	✓	✓	✓	✓	
INTERFAZ NATIVA	⚠	✓	⚠	✓	Apache Cordova hace uso de componentes Web. Rhomobile soporta unos pocos de componentes nativos
GENERACIÓN SIN MAC	✗	✗	✗	✗	En todas las plataformas, el desarrollo de apps para iOS requiere un Mac
IDE PROPIO	✗	✓	✓	✓	
COMUNIDAD	✓	✓	✓	✓	
LENGUAJES	JS	C#	Ruby, HTML, CSS	JS	

Tabla 1. Comparativa herramientas desarrollo móvil multiplataforma

- Las aplicaciones desarrolladas con Titanium tienen un aspecto prácticamente idéntico a las nativas.
- Da soporte a las 3 plataformas deseadas: Android, iOS y Windows Phone.
- La plataforma es utilizada por una amplia comunidad de desarrolladores.
- Framework open-source.
- Potente IDE para el desarrollo de aplicaciones; integra edición y generación

5 LENGUAJES DSL

Un Lenguaje Específico de Dominio (DSL – Domain-specific language) es un lenguaje de programación con una limitada expresividad que se centra en un dominio particular.

Existen cuatro elementos clave en esta definición:

- **Lenguaje de programación**: Un DSL es utilizado para instruir a una computadora a que haga algo. El diseño de los lenguajes de programación modernos tiende a que cada vez sea más sencillo de entender por las personas, pero siempre conservando que sea ejecutable por un computador.
- **Naturaleza del idioma**: Un DSL en un lenguaje de programación, y como tal debería tener una sensación de fluidez. Esta fluidez debe venir tanto de las expresiones individuales, como de la forma en la que se componen las expresiones.
- **Expresividad limitada**: Un lenguaje de programación de propósito general proporciona multitud de capacidades: soporte a diferentes datos, control, y estructuras de abstracción. Todas estas capacidades son útiles, pero hacen que el lenguaje sea más complicado de aprender y utilizar. No se puede desarrollar un sistema software con un DSL, pero sí que se puede utilizar en un aspecto particular de ese sistema.
- **Enfoque de dominio**: Un lenguaje limitado sólo es útil si se enfoca en un pequeño dominio. Que el enfoque sea el correcto, será lo que haga que el lenguaje merezca la pena.

5.1 ¿POR QUÉ UTILIZAR UN DSL?

Los DSLs son una herramienta con un enfoque determinado. No son como la orientación a objetos o los procesos ágiles, que introducen un cambio fundamental en la forma en que pensamos sobre el desarrollo de software. En lugar de eso, los DSLs son una herramienta específica para unas condiciones muy particulares. Por ejemplo, en un proyecto típico podríamos llegar a utilizar media docena de DSL en diferentes lugares: gestión de recursos multimedia, acceso a base de datos, automatización de pruebas, etc.

Un DSL en una fina capa encima del modelo, donde el modelo podría ser una librería o framework. Esta frase debería sirve parar recordar lo importante que es separar los beneficios proporcionados por el modelo, de los beneficios que aporta el DSL. Es un error común confundir los dos.

Los DSLs tienen el potencial de realizar ciertos beneficios. Cuando se está considerando la utilización de un DSL, se deberían evaluar esos beneficios y decidir cuáles de ellos son

aplicables a las circunstancias. A continuación se comentan varias de las razones por las que podríamos aplicar la utilización de un DSL en nuestros desarrollos.

5.1.1 MEJORA DE LA PRODUCTIVIDAD EN EL DESARROLLO

El mayor atractivo de un DSL es que proporciona un medio más claro de comunicar la intención de una parte de un sistema. Todo esto repercute en una forma más clara de leer un trozo de código, encontrar errores y modificar el sistema.

Los defectos en el desarrollo no sólo afectan a la calidad externa del software, sino que también ralentizan a los desarrolladores invirtiendo tiempo en investigar y solucionar errores, aparte de sembrar confusión sobre el comportamiento del sistema. Por el contrario, la limitada expresividad de un DSL hace que sea más complicado introducir errores en el desarrollo y sea más fácil ver cuándo se ha cometido un error.

El modelo por sí solo proporciona una mejora considerable en lo que a productividad se refiere. Evita la duplicidad recopilando el código común. Principalmente proporciona una abstracción que permite que sea más fácil especificar lo que está pasando de una manera más entendible. Un DSL realza esto proporcionando una manera más expresiva de leer y manipular esta abstracción. Un DSL puede ayudar a las personas a aprender cómo utilizar una API, ya que permite ver cómo los diferentes métodos de esta pueden ser combinados juntos.

Un DSL sólo tiene que dar soporte a la utilización actual del cliente, lo que puede reducir significadamente el área que los desarrolladores necesitan aprender.

5.1.2 COMUNICACIÓN CON LOS EXPERTOS DE DOMINIO

Una de las partes más difíciles de los proyectos software y la causa más común de fallo, es la comunicación entre los clientes y los usuarios de ese software. El hecho de que DSL proporcione un lenguaje claro y preciso para tratar con los dominios, permite mejorar esta comunicación.

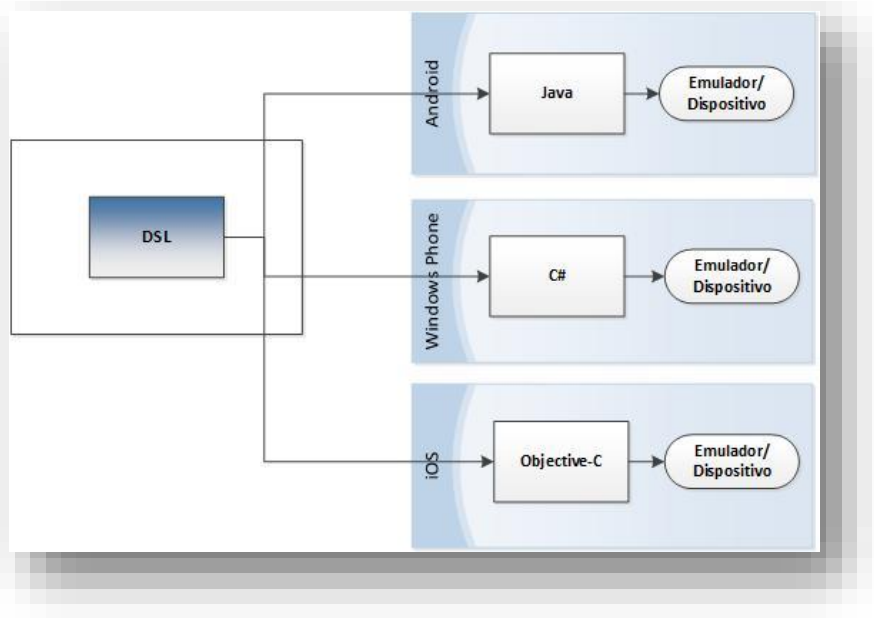
Con esto no se quiere decir que los expertos de dominio vayan a escribir los DSLs, pero pueden leerlos y entender lo que está haciendo el sistema. Siendo capaces de leer código DSL, los expertos de dominio pueden detectar errores. Del mismo modo, la comunicación entre los expertos y los programadores puede ser más efectiva, incluso escribiendo algunos borradores que pueden ser refinados en reglas DSL adecuadas. Es decir, la mayor ganancia en este sentido es que los expertos de dominio puedan leer los DSL.

Del mismo modo, también sería interesante que los expertos de dominio comprendan el contenido del modelo semántico (159), quizás proporcionando una visualización del modelo. Si se involucran a los expertos de dominio en un DSL también sería muy útil involucrarlos en el desarrollo de un modelo

A modo de conclusión, envolver a los expertos de dominio en un DSL es difícil de lograr, pero tienen una gran recompensa, e incluso si no se pueden involucrar los expertos, se habrá ganado mucho en cuanto a la productividad de los desarrolladores.

5.1.3 CAMBIOS EN EL CONTEXTO DE EJECUCIÓN

Otra de las facilidades que nos aporta un lenguaje DSL es que la definición puede ser evaluada en tiempo de ejecución, en lugar de hacerlo en tiempo de compilación. Por ejemplo en el ámbito de la gestión de los recursos multimedia, y volviendo al objetivo 1 que se planteaba dentro del apartado de ámbito y objetivos:



Si tuviésemos un método *play* que se comporte de una manera u otra en función de la plataforma en la que esté desplegada la App, la definición de este método se haría en tiempo de ejecución. De esta forma a través del DSL podríamos expresar lo que deseamos hacer a través del lenguaje DSL y después generar el código para el entorno actual de ejecución donde se va a utilizar.

5.1.4 MODELO DE COMPUTACIÓN ALTERNATIVO

La mayoría de la programación implica un modelo imperativo de computación. Esto significa que le decimos al computador las cosas que tiene que hacer a través de una secuencia de órdenes manejadas con instrucciones condiciones, bucles y variables. Este tipo de programación se ha hecho popular por ser relativamente fácil de comprender y de aplicar a multitud de problemas. Aunque esta no es siempre la mejor opción.

No se necesita un DSL para poder utilizar un modelo computacional. El comportamiento principal de un modelo computacional viene de un modelo semántico. Sin embargo un DSL puede hacer una gran diferencia, ya que hace que sea mucho más fácil para las personas, el manipular programas declarativos que pueblan el modelo semántico.

5.2 DISEÑO DE UN DSL

El objetivo general en el diseño de un DSL, como ocurre con cualquier escritura, es la claridad para el lector, que en este caso van a ser los programadores y/o los expertos de dominio. Se trata de hacer entender el significado de las sentencias DSL tan pronto y claramente sea posible.

Un buen diseño necesita ser presentado a la audiencia y ver cómo lo acoge. Hay que estar preparado para proporcionar múltiples alternativas y ver cómo reacciona. Todo el proceso de diseño envuelve un proceso de probar y volver a ejecutar multitud de pasos perdidos. No hay que preocuparse sobre las elecciones equivocadas, lo importante es corregirlas y al final encontrar el camino correcto.

No hay que preocuparse de utilizar la jerga del dominio y su modelo semántico en el DSL. Si los usuarios del DSL están familiarizados con la jerga, entonces deberían verla en el DSL. La jerga está ahí para mejorar la comunicación dentro de un dominio.

Se pueden hacer uso de las convenciones que estés acostumbrado a utilizar. Por ejemplo, si todo el mundo utiliza Java o C#, entonces puedes utilizar “//” para los comentarios y “{“para las estructuras jerárquicas.

Es importante no intentar hacer que el DSL se lea como un lenguaje natural. Ha habido varios intentos de hacer esto con lenguajes de propósito general, con Applescript como ejemplo más obvio. El problema es que estos ejemplos introducen mucha miga sintáctica lo que hace que se complique la semántica. Cuando se intenta hacer esto, parece que el lenguaje natural te sitúa en el contexto equivocado. Cuando se manipula un programa, hay que recordar siempre que estamos en un entorno de lenguaje de programación.

5.3 ARQUITECTURA DEL PROCESAMIENTO CON DSL

Un DSL en una fina capa sobre un modelo. Cuando se habla de modelo en este contexto, se refiere al patrón de modelo semántico. La idea básica detrás de este patrón es que todo el comportamiento semántico importante se captura en un modelo, y el rol del DSL es poblar ese modelo a través de un analizador. Esto significa que el modelo semántico juega un rol muy importante.

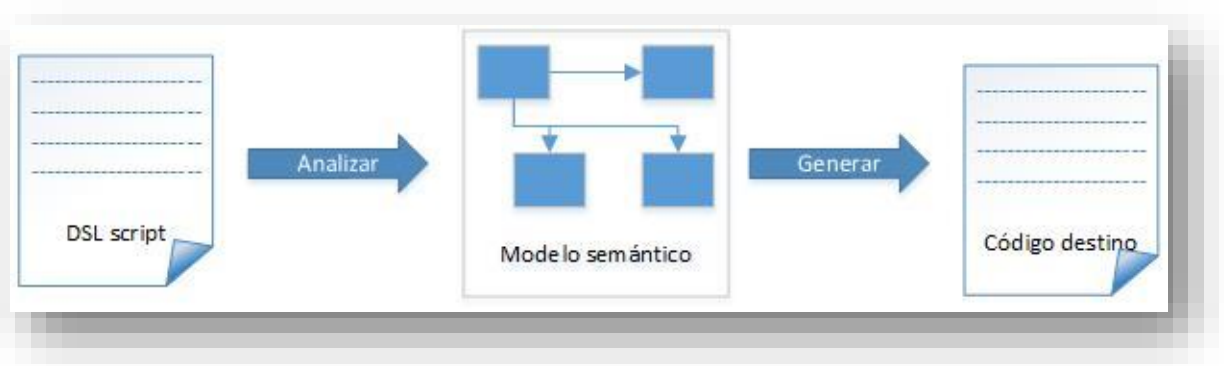


Ilustración 12. Arquitectura DSL

Un modelo semántico no tiene la necesidad de ser un modelo de objetos que combine datos y procesamiento. Basta con que sea una estructura de datos. Pero el modelo semántico también puede ser un modelo de objetos completamente normal, y se puede manipular de la misma forma que cualquier otro modelo de objetos.

Mantener un modelo semántico separado del DSL tiene varias ventajas. El principal beneficio es que podemos pensar en la semántica del domino sin enredarnos en la sintaxis del DSL o analizador. En definitiva, podemos probar el modelo semántico creando objetos en el modelo y manipulándolos directamente. Se pueden crear estados y transiciones para probarlas y ver si los eventos y los comandos se ejecutan. Todo esto sin tener que lidiar con el análisis sintáctico en absoluto.

El hecho de tener un modelo semántico por separado nos permite avanzar el modelo y el lenguaje por separado. Si queremos cambiar el modelo, podemos avanzar sin cambiar el DSL, añadiendo los constructores necesarios al DSL una vez que esté funcionando.

Una vez que tengamos el modelo semántico, tenemos que hacer que el modelo haga lo que queramos.

5.4 TIPOS DE DSL

Un lenguaje específico de dominio externo se representa en un lenguaje diferente al del lenguaje de programación principal con el que está trabajando. Este lenguaje podría utilizar una sintaxis personalizada o bien seguir la sintaxis de otra representación como XML.

Un lenguaje específico de dominio interno, también llamado DSL embebido, se representa dentro de la sintaxis de un lenguaje de propósito general. Es un uso más estilizado de ese lenguaje para un propósito de dominio específico.

Los DSLs se pueden dividir en tres categorías principales:

- External DSL: Es un lenguaje separado del lenguaje principal de la aplicación con la que trabaja. Normalmente se utiliza un lenguaje personalizado para estos DSL, pero también se utilizan sintaxis conocidas como XML.
- Internal DSL: Es una forma particular de utilizar un lenguaje de propósito general. Para su desarrollo se utiliza el mismo lenguaje de propósito general, pero sólo se utiliza un subconjunto de las características del lenguaje que permitan manejar un pequeño aspecto del sistema.
- Language workbench: Es un IDE especializado que nos permite la definición y el desarrollo de DSLs. Nos va a permitir determinar la estructura de un DSL así como la edición de scripts DSL.

5.5 DIFERENCIAS ENTRE DLS Y API

Dentro de los DSL es importante definir el concepto de **modelo** y su relación con un DSL. Se entiende por modelo la maquinaria que proporciona la forma de comportarse a la máquina de estados. Todo lo que hace el DSL es proporcionar una forma legible de poblar ese modelo, que es la diferencia entre los API. Desde el punto de vista de los DSL este modelo se entiende por **modelo semántico**. La semántica de un programa indica lo que tiene que hacer cuando se ejecuta.

El modelo semántico es una parte vital de un DSL bien diseñado, ya que proporciona una clara separación de conceptos entre el análisis de un lenguaje y los resultados semánticos.

5.6 PROBLEMAS CON LOS DSLS

La razón más importante para no utilizar un DSL es el hecho de no ver que ninguno de los beneficios asociados con los DSL pueda aplicar en nuestro contexto.

5.6.1 APRENDER VARIOS LENGUAJES

El hecho de utilizar varios lenguajes de programación dentro de un proyecto suele dar a entender que aumentará la complejidad del mismo. Si tenemos que conocer múltiples lenguajes nos será más difícil trabajar en el sistema e introducir nuevas personas en el proyecto.

En el caso de los DSL hay que aclarar que normalmente se equipara el esfuerzo de aprender estos, con el esfuerzo que supone aprender un lenguaje de propósito general. Los DSLs son más simples y sencillos de aprender que estos últimos.

A pesar de la sencillez y simplicidad, la mayoría de las críticas inciden en el hecho de que si tenemos múltiples DSLs dentro de un proyecto, será más difícil centrarse en lo que está pasando en un proyecto. La contraposición a esta afirmación reside en que dentro de un proyecto siempre tendremos áreas complicadas difíciles de aprender, incluso si no tenemos DSLs, tendremos muchas abstracciones dentro de nuestro código, que tendremos que aprender de todos modos. Por norma general, esas abstracciones se manejan a través de librerías para hacerlas más manejables, de manera que aunque no tengamos que aprender a manejar un DSL, tendremos que aprender las librerías que utilicemos.

Dado el caso, se puede considerar si es más costoso aprender un DSL que aprender el modelo subyacente del proyecto. El coste de aprender un DSL suele ser mucho más pequeño que el coste de entender el modelo subyacente, por lo que si una persona del equipo desarrolla un DSL para trabajar sobre el modelo, será más sencillo de manipular y entender, tanto para las personas que estén trabajando en el proyecto, como para otras personas que se pudieran incorporar.

5.6.2 COSTE DE DESARROLLO

El desarrollo de un DSL puede suponer un pequeño coste añadido al esfuerzo que puede suponer entender la librería subyacente. Siempre hay código que escribir y que mantener, por ello siempre hay que saber evaluar si merece la pena invertir tiempo y recursos en el desarrollo de un DSL.

No todas las librerías pueden enriquecerse si desarrollamos un DSL que las envuelva. Si el API de esta librería hace bien su trabajo no hay valor añadido en tener un DSL por encima de ella. Incluso aunque ayude, puede suponer mucho esfuerzo desarrollarlo y mantenerlo.

El mantenimiento de los DSL también es un factor importante que puede sumar coste al desarrollo. Ya sea por defectos en su uso o por que el equipo de desarrollo encuentre dificultades al entender cómo se utiliza, hará necesario mantener este código.

Otra de las cosas que puede inflar el coste de añadir un DSL, es el hecho de que las personas no están acostumbradas a desarrollar con ellos. Aunque no se deberían ignorar estos costes, también hay que tener en cuenta que el coste de la curva de aprendizaje puede amortizarse en las sucesivas veces que se pudiera utilizar el DSL en el futuro.

También hay que tener en cuenta que el coste de desarrollar un DSL es el coste sobre el otro coste de desarrollar el modelo.

En conclusión, se debe evaluar si merece la pena desarrollar un DSL. Hay que tener en cuenta que un buen DSL puede envolver una mala librería y hacer que sea más sencillo su manejo. Sin embargo un DSL malo es un desperdicio de recursos que hay que mantener (pero esto último puede decirse de cualquier código malo que se desarrolle).

6 PROPUESTA DE DSL

El DSL que se propone es un DSL de tipo interno, ya que su sintaxis va a estar contenida en el lenguaje anfitrión, que en este caso es JavaScript. Su objetivo es tener un medio de elevar el nivel de abstracción en el que se programe.

En esta propuesta de DSL interno se utiliza JavaScript como lenguaje a utilizar, por lo que el DSL está sujeto a la capacidad sintáctica de este lenguaje.

Se consideran varios enfoques a la hora de crear DSLs internos:

- Método de encadenamiento

Podríamos plantear el DSL interno al que podamos acceder a su funcionalidad, de la siguiente manera:

```
Audio()
  .open(path);
  .forward()
  .from("2:30")
  .to("3:00")
  .play();
```

Para poder crear un DSL similar al anterior, necesitaríamos desarrollar un constructor de expresiones que nos permita poblar el modelo semántico y proporcione un interface fluida. Este es el enfoque elegido, ya que la forma de invocar a las operaciones sobre los recursos multimedia, se considera que es la más amigable con el usuario.

- Funciones anidadas

En este enfoque el estilo del DSL es diferente. Se enlazan funciones dentro de funciones para poblar el modelo semántico. Por ejemplo:

```
Audio(
  play(path, forward(from("2:30"), to("3:00"))
);
```

- Expresiones/cierres lambda

Este enfoque podría presentar el siguiente modelo:

```
Audio(a -> {
        a.play( p -> {
                  p.forward();
                  p.from("2:30");
                  p.to("3:00");
                  p.play();
                  });
        });
```

Se sobrecarga con signos de puntuación y complicaría el desarrollo del constructor de expresiones, por lo que se descarta esta opción.

6.1 SEMANTIC MODEL

Según Martin Fowler: *en el contexto de un DSL, un modelo semántico es una representación, como un modelo de objetos en memoria, del mismo sujeto que describe al DSL.* Si por ejemplo un DSL describe una máquina de estados, entonces el modelo semántico podría ser un modelo de objetos con clases para el estado, eventos, etc.

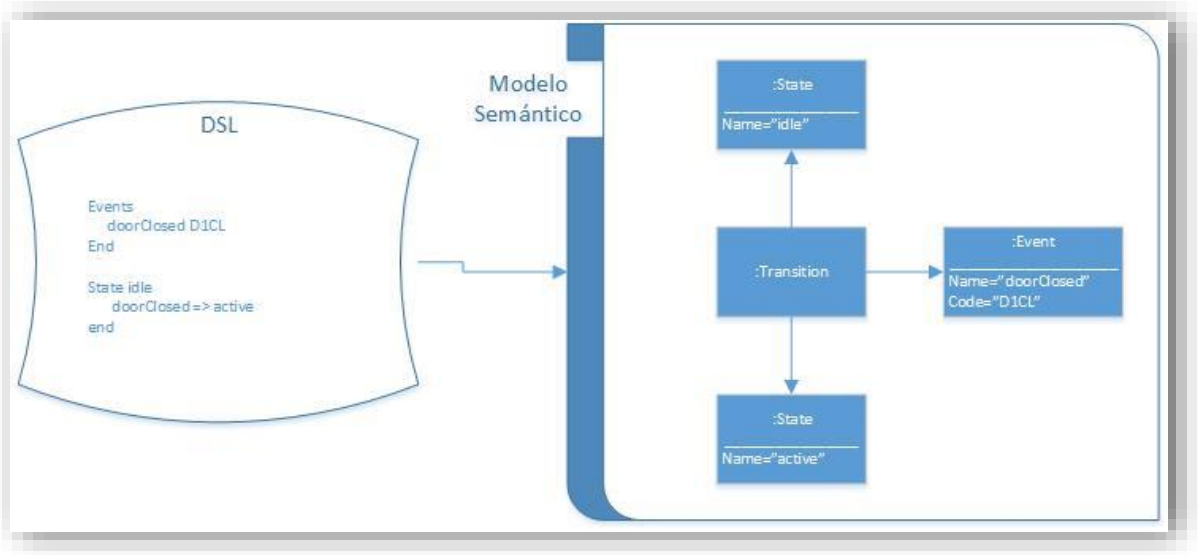


Ilustración 13. Modelo semántico poblado por un DSL

Es importante la necesidad de mantener el modelo semántico diferente del DSL, e introducir un constructor de expresiones intermedias que pueble el modelo semántico del DSL.

El hecho de que el modelo semántico se mantenga separado del DSL permite que este pueda evolucionar, en el sentido de que se puedan añadir nuevas APIs para recuperar, buscar o modificar los datos. De lo contrario si se mantiene el modelo semántico estrechamente ligado al DSL va a ser más complicado que el modelo semántico evolucione, y más difícil probar el modelo semántico y el DSL independientemente.

Un modelo semántico bien construido, debe ser usable sin la presencia del DSL, de manera que seamos capaces de poblar el modelo semántico a través de una interface de consulta que pueda asegurar que el modelo semántico captura realmente la semántica y permite el testeo independiente por sí mismo.

En este proyecto el modelo semántico va a estar compuesto por tres clases que representen los recursos multimedia que se desean gestionar:

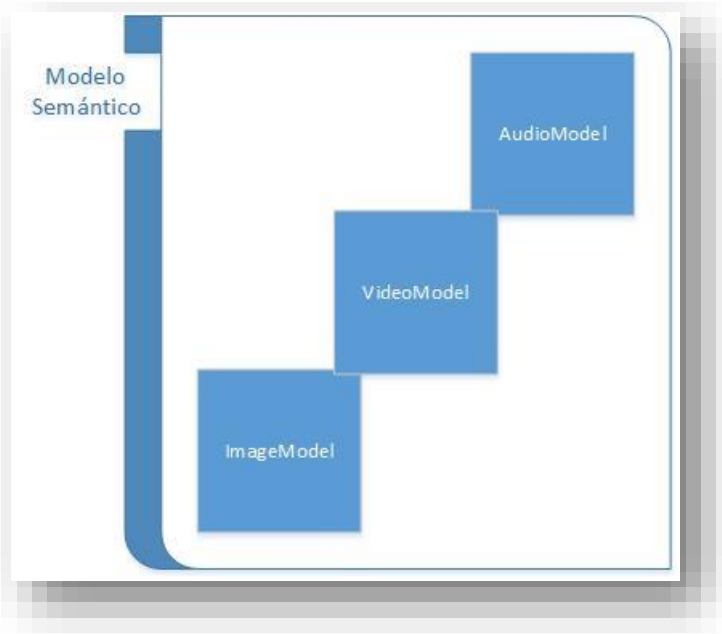


Ilustración 14. Modelo Semántico

A continuación se describen cada una de las clases:

6.1.1 AUDIOMODEL

Para representar al objeto Audio, se ha desarrollado la clase *AudioModel* que va a contener los métodos que le aporten una funcionalidad básica al mismo.

Presenta el siguiente formato:

```
// Audio Model
var AudioModel = function (src) {
	var sound = null;
	var soundSrc = src;
	var size = 0;
	var length = 0;
	var playbackSpeed = 0;

	this.AudioModel = new function (){
	};

	this.Open = function (src){
		sound = Titanium.Media.createSound({
				url: src,
				preload:true
				});
		soundSrc = src;
	};

	this.GetSound = function(){
		return sound;
	};

	this.GetSoundSrc = function (){
		return soundSrc;
	};
```

```
        this.SetSoundSrc = function (src){
                soundSrc = src;
        };

        this.GetSize = function (){
                return size;
        };

        this.GetLength = function (){
                return length;
        };

        this.SetLength = function (len){
                if (typeof(len) == 'number' && len >= 0){
                        length = len;
                }else{
                        length = 0;
                }
        };

        this.GetPlaybackSpeed = function (){
                return playbackSpeed;
        };

        this.SetPlaybackSpeed = function (playSpeed){
                if (typeof(playbackSpeed) == 'number' && playbackSpeed >= 0){
                        playbackSpeed = playSpeed;
                }else{
                        playbackSpeed = 0;
                }
        };

        this.Play = function (){
                sound.play();
        };

        this.Pause = function(){
                sound.pause();
        };

        this.Stop = function(){
                sound.stop();
        };

        this.Backward = function (time){
                sound.time = sound.time - time;
        };

        this.Forward = function (time){
                sound.time = sound.time + time;
        };
};
```

El método *Open* crea un objeto de tipo Sound a través del método *Titanium.Media.createSound* del API de Titanium. Se especifica la ruta desde el parámetro *src.* Una vez abierto, el objeto puede ser manipulado a través de los métodos *play(), pause(), stop()*, y alterar el tiempo de reproducción a través de su propiedad *time.*

Tiene métodos Get/Set para cada una de sus propiedades.

6.1.2 VIDEOMODEL

Para representar al objeto Video, se ha desarrollado la clase *VideoModel* que va a contener los métodos que le aporten una funcionalidad básica al mismo.

Presenta el siguiente formato:

```
var VideoModel = function (src) {
	var video = null;
	var videoSrc = src;
	var size = 0;
	var length = 0;
	var playbackSpeed = 0;

	this.VideoModel = new function (){
	};

	this.Open = function (src, win2){
		video = Titanium.Media.createVideoPlayer({
				url : src,
				movieControlMode : Titanium.Media.VIDEO_CONTROL_DEFAULT,
				movieControlStyle : Titanium.Media.VIDEO_CONTROL_EMBEDDED,
				scalingMode : Titanium.Media.VIDEO SCALING NONE,
				fullscreen : false,
				autoplay : false
			});
		videoSrc = src;
		win2.add(video);
	};

	this.GetVideo = function(){
		return video;
	};

	this.GetVideoSrc = function (){
		return videoSrc;
	};

	this.SetVideoSrc = function (src){
		videoSrc = src;
	};

	this.GetSize = function (){
		return size;
	};

	this.SetSize = function (s){
		size = s;
	};

	this.GetLength = function (){
		return length;
	};

	this.SetLength = function (len){
		if (typeof(len) == 'number' && len >= 0){
			length = len;
		}else{
			length = 0;
		}
	};

	this.GetPlaybackSpeed = function (){
		return playbackSpeed;
	};

	this.SetPlaybackSpeed = function (playSpeed){
```

```
                if (typeof(playbackSpeed) == 'number' && playbackSpeed >= 0){
                        playbackSpeed = playSpeed;
                }else{
                        playbackSpeed = 0;
                }
        };

        this.Play = function (){
                video.play();
        };

        this.Pause = function(){
                video.pause();
        };

        this.Stop = function(){
                video.stop();
        };

        this.Backward = function (time){
                video.time = video.time - time;
        };

        this.Forward = function (time){
                video.time = video.time + time;
        };
};
```

El método *Open* crea un objeto de tipo Video a través del método *Titanium.Media.createVideoPlayer* del API de Titanium. Se especifica la ruta desde el parámetro *src.* Una vez abierto, el objeto puede ser manipulado a través de los métodos *play(), pause(), stop()*, y alterar el tiempo de reproducción a través de su propiedad *time.*

Tiene métodos Get/Set para cada una de sus propiedades.

En este caso, el método *Open* recibe el parámetro *win2* que es el contenedor donde se va a presentar el vídeo y que está definido de manera que ocupe una porción de la pantalla del dispositivo. En caso de que no se le asocie un contenedor, el vídeo se presenta ocupando la pantalla completa dejando el resto de controles en una capa por debajo, algo que no nos interesa ya que ocultaría los botones sobre los que el usuario puede realizar acciones sobre el vídeo.

6.1.3 IMAGEMODEL

Para representar al objeto Imagen, se ha desarrollado la clase *ImageModel* que va a contener los métodos que le aporten una funcionalidad básica al mismo.

Presenta el siguiente formato:

```
var ImageModel = function (src) {
        var image = null;
        var imageSrc = src;
        var size = 0;
        var opacity = 0;
        var t = Ti.UI.create2DMatrix();

        this.ImageModel = new function (){
        };

        this.Open = function (src, win){
```

```
                image = Ti.UI.createImageView({
                  image: imageURL
                });
                imageSrc = src;
                win.add(image);
        };

        this.Rotate = function (){
                t = t.rotate(90);
                var anim = Titanium.UI.createAnimation();
                anim.transform = t;
                image.animate(anim);
        };

        this.Hide = function (){
                image.animate({
                        curve:Ti.UI.ANIMATION CURVE EASE IN OUT,
                        opacity:0,
                        duration:200
                });
        };

        this.Show = function (){
                image.animate({
                        curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT,
                        opacity:1,
                        duration:200
                });
        };

        this.Src = function (){
                return imageSrc;
        };

        this.Size = function (){
                return size;
        };

        this.Opacity = function (){
                return opacity;
        };

        this.Opacity = function (val){
                if (typeof(val) == 'number' && val >= 0){
                        opacity = val;
                }else{
                        opacity = 0;
                }
        };
};
```

El método *Open* crea un objeto de tipo *Image* a través del método *Ti.UI.createImageView* del API de Titanium, donde se especifica la ruta a la imagen desde el parámetro *src.*

Tiene métodos Get/Set para cada una de sus propiedades.

Del mismo modo que en el objeto Video, el método *Open* recibe el parámetro *win* que es el contenedor donde se va a presentar la imagen, y que está definido de manera que sólo ocupe una porción de la pantalla del dispositivo. Una vez abierto, el objeto puede ser manipulado a través de los métodos *rotate, show, hide,* que hacen uso del API de Titanium a través de los métodos *Titanium.UI.createAnimation* y la propiedad *animate.*

6.2 EXPRESION BUILDER

Un constructor de expresiones es un objeto que proporciona una interface fluida que traslada las invocaciones que se hacen sobre ella, a llamadas de una capa subyacente, que en este caso será nuestro modelo semántico.

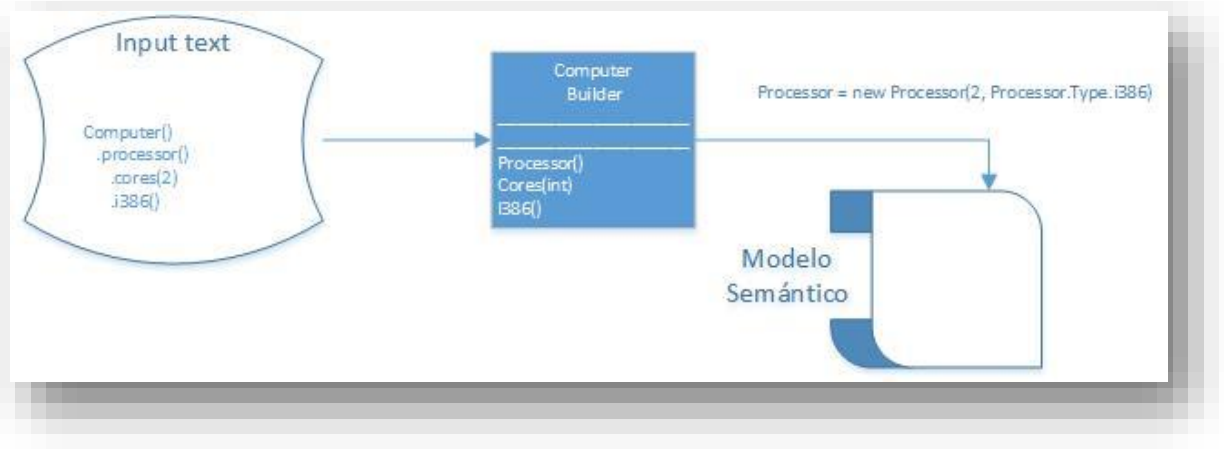


Ilustración 15. Ejemplo de constructor de expresiones

Se podría pensar en prescindir de esta capa, e insertar el código de ella directamente en el modelo semántico, pero hay que intentar que el modelo semántico sea sencillo de depurar y el hecho de que se añada código que tenga que ver con la construcción de expresiones no va a facilitar esta tarea.

Es importante que el modelo semántico tenga objetos con interfaces de consulta, que puedan ser manipulados sin necesidad de ningún constructor de expresiones. Para verificar esto, deberíamos ser capaces de escribir pruebas sobre el modelo semántico que no utilicen nada del constructor de expresiones. De manera que podamos realizar pruebas independientes sobre el modelo semántico y sobre el constructor de expresiones y los resultados deberían ser los mismos.

En este proyecto el generador de expresiones va a estar compuesto por tres clases (cada una asociada a su clase del modelo semántico) que representen los recursos multimedia que se desean gestionar:

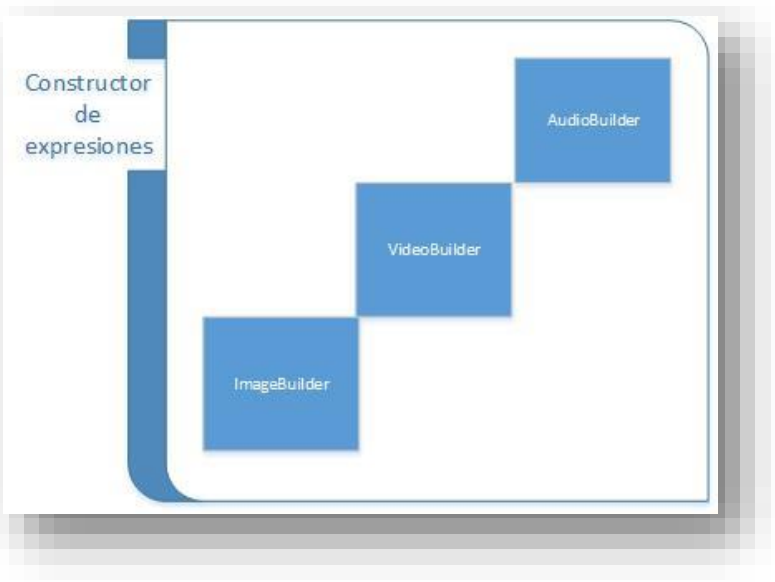


Ilustración 16. Constructor de expresiones

A continuación se describen cada una de las clases:

6.2.1 AUDIOBUILDER

A través de nuestro constructor de expresiones queremos acceder a la funcionalidad del modelo semántico, pero proporcionando al usuario que lo utilice un lenguaje más amigable, para ello definimos esta clase con el siguiente modelo:

```
Ti.include('../semanticModel/AudioModel.js');

var AudioBuilder = function () {

	var audioModel;

	this.AudioBuilder = new function (){
		audioModel = new AudioModel();
	};

	this.Open = function (soundSrc){
		audioModel.Open(soundSrc);
	};

	this.Play = function(){
		audioModel.Play();
	};

	this.Pause = function (){
		audioModel.Pause();
	};

	this.Stop = function (){
		audioModel.Stop();
	};

	this.Forward = function(time){
```

```
                audioModel.Forward(time);
        };

        this.Backward = function(time){
                audioModel.Backward(time);
        };
};

var SingletonAudio = (function () {
    var instance;

    function createInstance() {
        var object = new AudioBuilder();
        return object;
    }

    return {
        getInstance: function () {
            if (!instance) {
                instance = createInstance();
            }
            return instance;
        }
    };
})();

var Audio = function(){
        return SingletonAudio.getInstance();
};
```

Dentro de ella se define la clase *Audio* que invoca a través del patrón Singleton a una instancia única de la clase *AudioBuilder.* Esta última se encarga de invocar a la funcionalidad del modelo semántico. Todo ello para tener una forma más amigable de invocar al modelo semántico, y que se verá posteriormente en la capa de *Interacción con el DSL*, descrita en el apartado siguiente.

***NOTA**:* JavaScript es un lenguaje basado en prototipos que no contiene ninguna declaración de clase como se encuentra, por ejemplo, en C++ o Java. En su lugar, JavaScript utiliza funciones como clases. Definir una clase es tan sencillo como definir una función. De ahí que las funciones dentro de nuestro código se denominen clases.

6.2.2 VIDEOBUILDER

Esta clase define un formato similar a la de Audio, creando una clase *Video* que hace uso del patrón Singleton para acceder a la funcionalidad del modelo semántico a través de la clase *VideoBuilder.*

```
Ti.include('../semanticModel/VideoModel.js');

var VideoBuilder = function () {

        var videoModel;

        this.VideoBuilder = new function (){
```

```
                videoModel = new VideoModel();
        };

        this.Open = function (videoSrc, win2){
                videoModel.Open(videoSrc, win2);
        };

        this.Play = function(){
                videoModel.Play();
        };

        this.Pause = function (){
                videoModel.Pause();
        };

        this.Stop = function (){
                videoModel.Stop();
        };

        this.Forward = function(time){
                videoModel.Forward(time);
        };

        this.Backward = function(time){
                videoModel.Backward(time);
        };
};

var SingletonVideo = (function () {
    var instance;

    function createInstance() {
        var object = new VideoBuilder();
        return object;
    }

    return {
        getInstance: function () {
            if (!instance) {
                instance = createInstance();
            }
            return instance;
        }
    };
})();

var Video = function(){
        return SingletonVideo.getInstance();
};
```

6.2.3 IMAGEBUILDER

Al igual que en Video y Audio, el objeto Image retorna una instancia única hacia la clase *ImageBuilder* que accede a la funcionalidad del modelo semántico.

```
Ti.include('../semanticModel/ImageModel.js');

var ImageBuilder = function () {

        var imageModel;

        this.ImageBuilder = new function (){
                imageModel = new ImageModel();
        };
```

```
        this.Open = function (src, win){
                imageModel.Open(src, win);
        };

        this.Rotate = function (){
                imageModel.Rotate();
        };

        this.Hide = function (){
                imageModel.Hide();
        };

        this.Show = function(){
                imageModel.Show();
        };
};

var SingletonImage = (function () {
    var instance;

    function createInstance() {
        var object = new ImageBuilder();
        return object;
    }

    return {
        getInstance: function () {
            if (!instance) {
                instance = createInstance();
            }
            return instance;
        }
    };
})();

var Image = function(){
        return SingletonImage.getInstance();
};
```

6.3 INTERACCIÓN CON EL DSL

En este apartado se representa la forma en la que el usuario va a hacer uso de la funcionalidad de nuestro DSL a través del constructor de expresiones, en base a lo que se ha desarrollado en el prototipo.

El prototipo, como se verá en el apartado de *Aplicación del DSL,* consta de una serie de pestañas que se utilizan para separar la funcionalidad que sirve: audio, video e imagen.

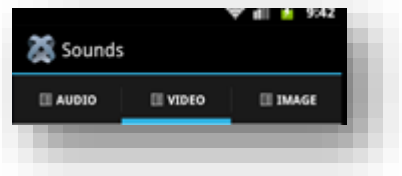


Ilustración 17. Pestañas del prototipo

A nivel de código esto se traduce en lo siguiente:

6.3.1 APP.JS

Este elemento se considera el punto de entrada en la aplicación. En él reside la carga del constructor de expresiones, el modelo semántico y aquellos elementos que forman parte de la interfaz gráfica, como la definición del color de fondo de la app o la carga de las pestañas de audio, video e imagen.

Presenta el siguiente modelo:

```
// this sets the background color of the master UIView (when there are no windows/tab
// groups on it)
Titanium.UI.setBackgroundColor('#000');

// create tab group
var tabGroup = Titanium.UI.createTabGroup();

Ti.include('expresionBuilder/AudioBuilder.js');
Ti.include('expresionBuilder/ImageBuilder.js');
Ti.include('expresionBuilder/VideoBuilder.js');

Ti.include('audioTab.js');
Ti.include('videoTab.js');
Ti.include('imageTab.js');

//  add tabs
tabGroup.addTab(tab1);
tabGroup.addTab(tab2);
tabGroup.addTab(tab3);

// open tab group
tabGroup.open();
```

6.3.2 AUDIOTAB.JS

Dentro de este elemento se definen los elementos gráficos que interactúan con el usuario junto a sus eventos asociados. En este caso nos referimos a los botones de abrir, reproducir, pausar, parar, avanzar y retroceder, junto al evento asociado al momento en el que el usuario pulsa sobre ellos.

Cuando el usuario interactúa con estos controles, se invoca al constructor de expresiones para lanzar la funcionalidad que reside en el DSL. (Se resalta en amarillo dentro del código.)

Como se puede apreciar, se hace uso del método de encadenamiento descrito en el apartado de *Propuesta de DSL,* porque se considera que puede ser el más entendible por usuarios de negocio que no estén acostumbrados a tratar con otros lenguajes de programación.

```
var percent = '20%';

// create base UI tab and root window
var win1 = Titanium.UI.createWindow({
    title:'Audio',
```

```
    backgroundColor:'#fff'
});
var tab1 = Titanium.UI.createTab({
    icon:'KS nav ui.png',
    title:'Audio',
    window:win1
});

// create a button that when clicked will play the sound
var openButton = Ti.UI.createButton({
        title: 'Open',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
// when the button is clicked, play the sound
openButton.addEventListener('click', function() {
        Audio().Open('/sounds/funkysynth.mp3');
});

// create a button that when clicked will play the sound
var playButton = Ti.UI.createButton({
        title: 'Play',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
// when the button is clicked, play the sound
playButton.addEventListener('click', function() {
        Audio().Play();
});

// create a button that when clicked will play the sound
var pauseButton = Ti.UI.createButton({
        title: 'Pause',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
// when the button is clicked, play the sound
pauseButton.addEventListener('click', function() {
        Audio().Pause();
});

// create a button that when clicked will stop the sound
var stopButton = Ti.UI.createButton({
        title: 'Stop',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
// when the button is clicked, stop the sound
stopButton.addEventListener('click', function() {
        Audio().Stop();
});

// create a button that when clicked will go back the music in five seconds
var backwardButton = Ti.UI.createButton({
        title: 'Back',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
backwardButton.addEventListener('click', function() {
```

```
        Audio().Backward(5);
});

// create a button that when clicked will go forward the music in five seconds
var forwardButton = Ti.UI.createButton({
        title: 'Forward',
        width: percent,
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});
forwardButton.addEventListener('click', function() {
        Audio().Forward(5);
});

var btnContainer = Ti.UI.createView({layout:'horizontal', width:'100%'});
btnContainer.add(openButton);
btnContainer.add(playButton);
btnContainer.add(pauseButton);
btnContainer.add(stopButton);
btnContainer.add(backwardButton);
btnContainer.add(forwardButton);

win1.add(btnContainer);
```

6.3.3 VIDEOTAB.JS

Este elemento presenta una funcionalidad similar a la del Audio, pero sin los botones de avanzar y retroceder la reproducción.

Se indica la ruta hacia el vídeo almacenado en el proyecto: */videos/test.mp4*

```
var videoURL = '/videos/test.mp4';

// create controls tab and root window
var win2 = Titanium.UI.createWindow({
    title:'Video Demo',
    backgroundColor:'#fff'
});

var tab2 = Titanium.UI.createTab({
    icon:'KS nav ui.png',
    title:'Video',
    window:win2
});

var playButton = Ti.UI.createButton({
    title : "Start",
        width:'33%',
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top:20
});

playButton.addEventListener('click', function() {
        Video().Play();
});

var pauseButton = Ti.UI.createButton({
    title : "Pause",
        width:'33%',
        height:'60',
        font:{size:9, fontWeight:'bold'},
```

```
        top:20
});

pauseButton.addEventListener('click', function() {
        Video().Pause();
});

var stopButton = Ti.UI.createButton({
    title : "Stop",
        width:'33%',
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top : 20
});

stopButton.addEventListener('click', function(){
        Video().Stop();
});

Video().Open(videoURL, win2);
var btnContainer = Ti.UI.createView({layout:'horizontal', width:'100%'});
btnContainer.add(playButton);
btnContainer.add(pauseButton);
btnContainer.add(stopButton);
win2.add(btnContainer);
```

6.3.4 IMAGETAB.JS

Del mismo modo que en los dos elementos anteriores, en esta clase se definen los elementos gráficos con sus eventos asociados que invocan a la funcionalidad del DSL a través del constructor de expresiones.

```
var imageURL = '/images/tiger.png';

// create controls tab and root window
var win3 = Titanium.UI.createWindow({
    title:'Image',
    backgroundColor:'#fff'
});

var tab3 = Titanium.UI.createTab({
    icon:'KS nav ui.png',
    title:'Image',
    window:win3
});

var image = Ti.UI.createImageView({
  image: imageURL
});

var rotateButton = Ti.UI.createButton({
    title : "Rotation",
        width:'33%',
        height:'60',
        font:{size:9, fontWeight:'bold'},
        top : 20
});

rotateButton.addEventListener('click', function() {
        Image().Rotate();
});
```

```
var hideButton = Ti.UI.createButton({
    title : "Hide",
       width:'33%',
       height:'60',
       font:{size:9, fontWeight:'bold'},
       top : 20
});

hideButton.addEventListener('click', function() {
       Image().Hide();
});

var showButton = Ti.UI.createButton({
    title : "Show",
       width:'33%',
       height:'60',
       font:{size:9, fontWeight:'bold'},
       top : 20
});

showButton.addEventListener('click', function() {
       Image().Show();
});

Image().Open(imageURL, win3);
var btnContainer = Ti.UI.createView({layout:'horizontal', width:'100%'});
btnContainer.add(rotateButton);
btnContainer.add(showButton);
btnContainer.add(hideButton);
win3.add(btnContainer);
```

6.4 RESUMEN

De forma gráfica la separación por capas del DSL implementado en el prototipo podría representarse de la siguiente forma:

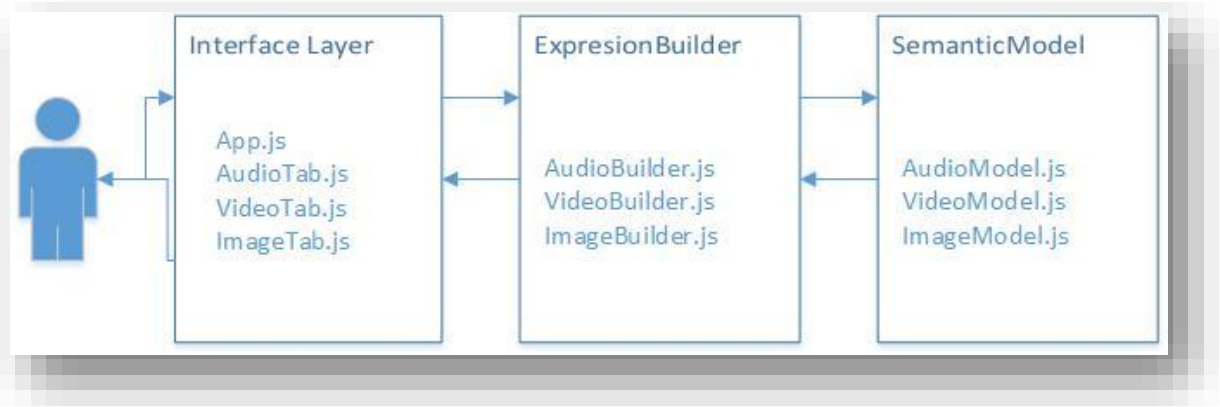


Ilustración 18. Separación por capas del DSL propuesto

En este caso, la complejidad del modelo semántico no es excesivamente alta, pero sirve para sentar las bases en la forma que puede seguir el lenguaje DSL que hemos desarrollado, e ir aumentando su complejidad o funcionalidad a través de nuevos métodos que se vayan incorporando a posteriori.

Para que un usuario pueda utilizar el DSL, bastaría que importase las clases asociadas con el constructor de expresiones en su código JavaScript, tal y como se hace en el archivo *tab.js*.

7 DESARROLLO DEL PROTOTIPO

Para llegar a traducir nuestro prototipo a cada una de las plataformas han sido necesarios una serie de pasos que se describen a continuación. Desde la puesta en marcha del entorno de desarrollo con la plataforma Titanium hasta la generación de código para Windows Phone, Android e iOS.

7.1 PUESTA EN MARCHA DEL ENTORNO DE DESARROLLO

Para este proyecto partimos de un equipo con Windows 8.1 Pro de 64 bits, en el que hemos instalado los siguientes requisitos, que varían respecto a la plataforma sobre la que queramos generar el código:

7.1.1 PASOS GENERALES

Los requisitos descritos a continuación son necesarios siempre, independientemente de la plataforma de destino sobre la que queramos que se genere la app:

- **Titanium Studio 3.1.3**. El entorno gráfico de la plataforma. Tras finalizar la instalación, descargamos e instalamos todas sus actualizaciones. En este caso hemos instalado la versión 3.1.3, porque la última versión no encontraba la máquina virtual de Android, a pesar de estar bien instalada y la ruta bien configurada. Con esta versión del IDE se corrigió este problema.
- **Titanium CLI**. Versión 4.1.0-dev. Es una herramienta por línea de comandos que nos va a permitir la creación y construcción de las aplicación móviles. Viene por defecto en la instalación de Titanium Studio, pero puede interesarnos más instalar sólo la interface por líneas de comandos, que el entorno gráfico. En el prototipo desarrollado se ha optado por esta herramienta, en lugar del entorno gráfico por problemas en este último que se describirán en apartados posteriores.

  El IDE 3.1.3 instala una versión anterior de Titanium CLI, pero nos interesa la última ya que es la que da opción a generar la app para la plataforma Windows.

- **Node.js** Instalando este entorno de programación tendremos el gestor de paquetes **npm** que nos va a permitir instalar y gestionar dependencias de una aplicación. En este caso particular se utilizará para actualizar las posibles últimas versiones de Titanium CLI.

- Configuración de las **variables de entorno** que apunten entre otros, a los correspondientes directorios de instalación

7.1.2 ANDROID

Para esta plataforma, se han necesitado los siguientes requisitos:

- **Máquina virtual de java** 1.7.0_79, en su versión de 32 bits, ya que la de 64 bits no es compatible con Titanium en Windows, para ninguna de sus versiones [31]. Titanium Studio ofrece instalar la versión 1.7 si no se encuentra ninguna JDK en el equipo.
- **Máquina virtual de Android**. [32]. Que va a permitir a Titanium generar la app para esta plataforma.
- **Genymotion** [33]. Este emulador nos permite probar la apk generada por Titanium. Tendremos que instalar al menos un dispositivo virtual a través de la opción *+Add* de su menú.

7.1.3 WINDOWS PHONE

Para esta plataforma, se han instalado los siguientes componentes:

- **Microsoft Visual Studio** Premium 2013. Versión 12.0.3 con Update 4.
- **Característica SDK de Windows Phone 8.0**. Se instala a través de Panel de control\Programas\Programas y características -> Microsoft Visual Studio Premium 2013 -> opción Cambiar -> opción Modificar y seleccionamos la característica que pone SDK de Windows Phone 8
- En caso de querer instalar emuladores: [34] Microsoft ofrece un paquete de instalación que añade imágenes de emuladores sobre una instalación existente de Visual Studio 2015 o Visual Studio 2013 Update 2 o superior. Con este paquete instalado podremos probar las apps que desarrollemos.

7.1.4 IOS

Para poder generar el código en esta plataforma es necesario un equipo Mac, y no se dispone de ninguno, por lo que se ha optado por la alternativa *MacInCloud.* Esta empresa ofrece un servicio de alquiler de equipo Mac remoto.

A través de un programa de escritorio remoto nos podemos conectar al equipo Mac, que tras validar el usuario y la contraseña nos ofrece un equipo con todo el entorno de desarrollo que necesitamos ya instalado, que en nuestro caso es Titanium CLI, Xcode, Node.js y la iOS SDK.

7.2 GENERACIÓN DE CÓDIGO EN CADA UNA DE LAS PLATAFORMAS

Una vez descritos los pasos necesarios para la puesta en marcha del entorno de desarrollo, explicamos la creación del proyecto en Titanium, la disposición del código en nuestro proyecto y la forma de generar el correspondiente código para cada una de las plataformas móviles.

Para crear un proyecto en Titanium, tenemos dos formas de hacerlo: a través del IDE *File -> New -> Mobile Project,* o bien a través de línea de comandos, con la orden:

```
ti create -t app --id <APP_ID> -n <APP_NAME> -p <PLATFORMS> -d <WORKSPACE_DIRECTORY> -u
<APP URL>

## Example

ti create -t app --id com.appcelerator.sample -n SampleProject -p android,ios -d
~/Documents/Titanium_Studio_Workspace -u http://www.appcelerator.com
```

En nuestro proyecto la creación y el manejo de código se realizarán a través del IDE, mientras que la generación se realizará a través de CLI.

Nota: Para generar el código en cada una de las plataformas móviles, utilizamos la herramienta CLI, ya que el IDE da problemas teniendo el emulador Genymotion instalado, se pierde la configuración de la máquina virtual de Android, algo que no ocurre desde CLI.

Cuando creamos un nuevo proyecto, el IDE nos da la opción de utilizar dos plantillas diferentes: *Alloy* o *Classic.* En nuestro caso seleccionamos Classic -> Tabbed Application.

El código desarrollado se organiza de la siguiente manera:

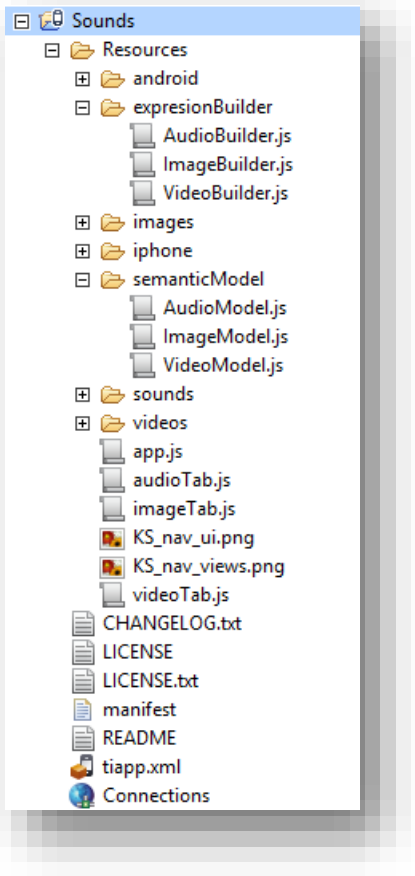


Ilustración 19. Disposición del código en nuestro proyecto

Dentro del proyecto convive el código propio que hemos generado junto a directorios que genera automáticamente la herramienta, por ejemplo, los directorios Android e iphone que contienen contenido estático como imágenes o ficheros de configuración (en el directorio raíz).

Para la generación del código y prueba del mismo hacemos los siguientes pasos:

1. Abrir una ventana de comandos CMD y nos situamos en el directorio donde tengamos el proyecto.
2. Escribimos ***ti build –b***. Esta instrucción nos hará rellenar la información necesaria para construir el proyecto, en este caso, la plataforma en la que queremos probar la aplicación.

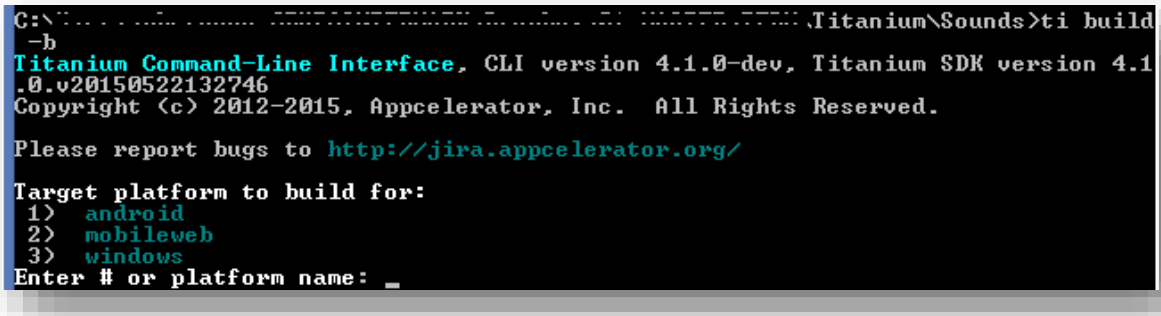


Ilustración 20. Captura de CLI. Generación de código para Android y Windows Phone

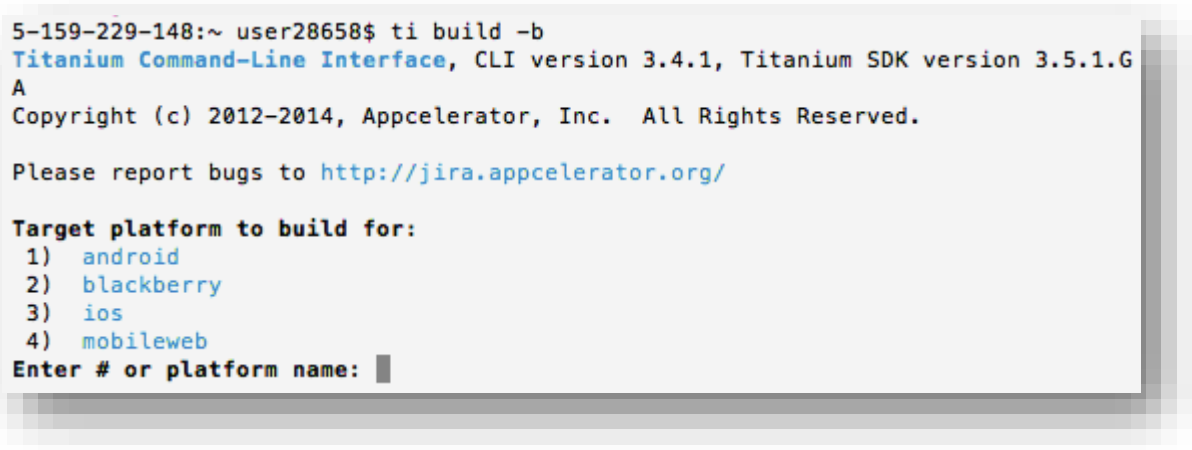


Ilustración 21. Captura de CLI. Generación de código para iOS dentro del equipo Mac de MacInCloud

Nota: La herramienta CLI evalúa la configuración del proyecto para ver que versión de la SDK tiene que utilizar para construir la aplicación. A continuación se indica por orden de preferencia una lista donde la herramienta CLI evalúa la versión de la SDK que tiene que utilizar:

1. *Tiapp.xml.* Dentro de este fichero se especifica la versión de la SDK con la etiqueta *sdk-version.* Para cambiar esta versión, habrá que editar manualmente este archivo.
2. A la hora de hacer el build, podemos especificar, con la opción --sdk la versión de la SDK a utilizar.
3. Podemos especificar la versión SDK a través de *App.sdk,* configuración que se puede indicar a través del comando *titanium config.* Por ejemplo: *titanium config app.sdk <sdk_version>.*
4. Seleccionar la SDK desde línea de comandos, a través del comando *titanium sdk select.*

3. Una vez indicada la plataforma, se crea el directorio build y Titanium vuelca todo el código asociado a la plataforma Android/Windows/iphone, dentro de este directorio.

4. Para probar la aplicación, abrimos el emulador *Genymotion* y una vez instalado y arrancado el terminal que tengamos configurado en él, podremos arrastar directamente la *apk* (en el caso de Android) hacia él, donde automáticamente se instalará en el dispositivo.

7.3 GENERACIÓN DE CÓDIGO NATIVO DESDE EL DSL

Dado que en el apartado anterior, se ha explicado los pasos a realizar para poder generar el código del DSL en cada una de las plataformas móviles, en este apartado se explica un paso más allá, que es la traducción del código utilizado en el DSL a código nativo.

En nuestro proyecto en particular la transformación del código escrito a código nativo de cada una de las plataformas móviles, se realiza a través de *Titanium.* Gracias a esta plataforma nuestro código JavaScript puede interactuar con el código y componentes nativos como botones, tablas, etiquetas, etc.

En nuestro prototipo se han utilizado invocaciones al API de Titanium como *Titanium.Media.createSound, Ti.UI.createImageView, Titanium.Media.createVideoPlayer*, que internamente se traducen como se describe a continuación.

Los pasos que realiza internamente la herramienta, a alto nivel, son los siguientes:

- Se analiza el código estáticamente buscando referencias a los módulos de Titanium.
- Los metadatos (tiapp.xml), cadenas de localización (strings.xml) y archivos descriptivos de la app, generan archivos análogos en cada una de las plataformas.
- En iOS:
    - Se genera y se configura un proyecto XCode.
    - El fuente JS escrito, se almacena como una variable en un archivo de objective C.
    - Se utiliza XCodeBuild para generar los binarios finales.
    - Se aplican los perfiles de aprovisionamiento y las claves de firma.
    - Se utiliza iTunes entre otras herramientas para enviar el instalable IPA al dispositivo iOS.
- En Android:
    - Se genera y se configura un proyecto en Android.
    - En el modo desarrollo, el fuente JS escrito se empaqueta como un complemento de la APK.
    - En el modo de distribución se compila el JS a bytecode de Java utilizando el compilador Rhino JSC [28].
    - Dex, aapt y otras herramientas de la SDK de Android se utilizan para construir y generar la APK final.

    - Por último, se utilizan las herramientas ADB y KeyTook para publicar la APK en el emulador o dispositivo.
- En Windows Phone:
    - Se genera y se configura un proyecto Windows Phone para Visual Studio. (No hay mucha información sobre el proceso que se sigue en Windows Phone, por ser una plataforma que se han introducido hace relativamente poco tiempo)

Dentro del proceso anterior entran en juego los objetos proxy [29], que son objetos nativos que exponen un API JavaSript. Todos los objetos proxy extienden de una clase de proxy específico (TiProxy en iOS, KrollProxy en Android) que proporciona un mecanismo para la unión de los métodos y las propiedades nativas de JavaScript.

Cada vez que se interactúa con una función o propiedad en el espacio de nombres de TI.*, consideramos al objeto resultante, un objeto proxy. Por ejemplo:

```
// 1. Almacenamos en un objeto la invocación a una función del espacio de nombres TI.*
var win = Ti.UI.createWindow();
```

Con el fin de facilitar la comunicación desde el contexto JavaScript de nuestro proyecto al código nativo de las plataformas, estos objetos proxy se han instrumentado con un comportamiento especial que permite que el código nativo (Java, C, Objective-C) pueda interceptar esas asignaciones e invocaciones. Este modelo de interacción es lo que permite a los desarrolladores escribir código nativo en JavaScript.

Vamos a verlo con un ejemplo:

```
// 1. Create a proxy object
var win = Ti.UI.createWindow();

// 2. Set a property
win.title = "Hello World";

// 3. Call a method on the proxy
win.open();
```

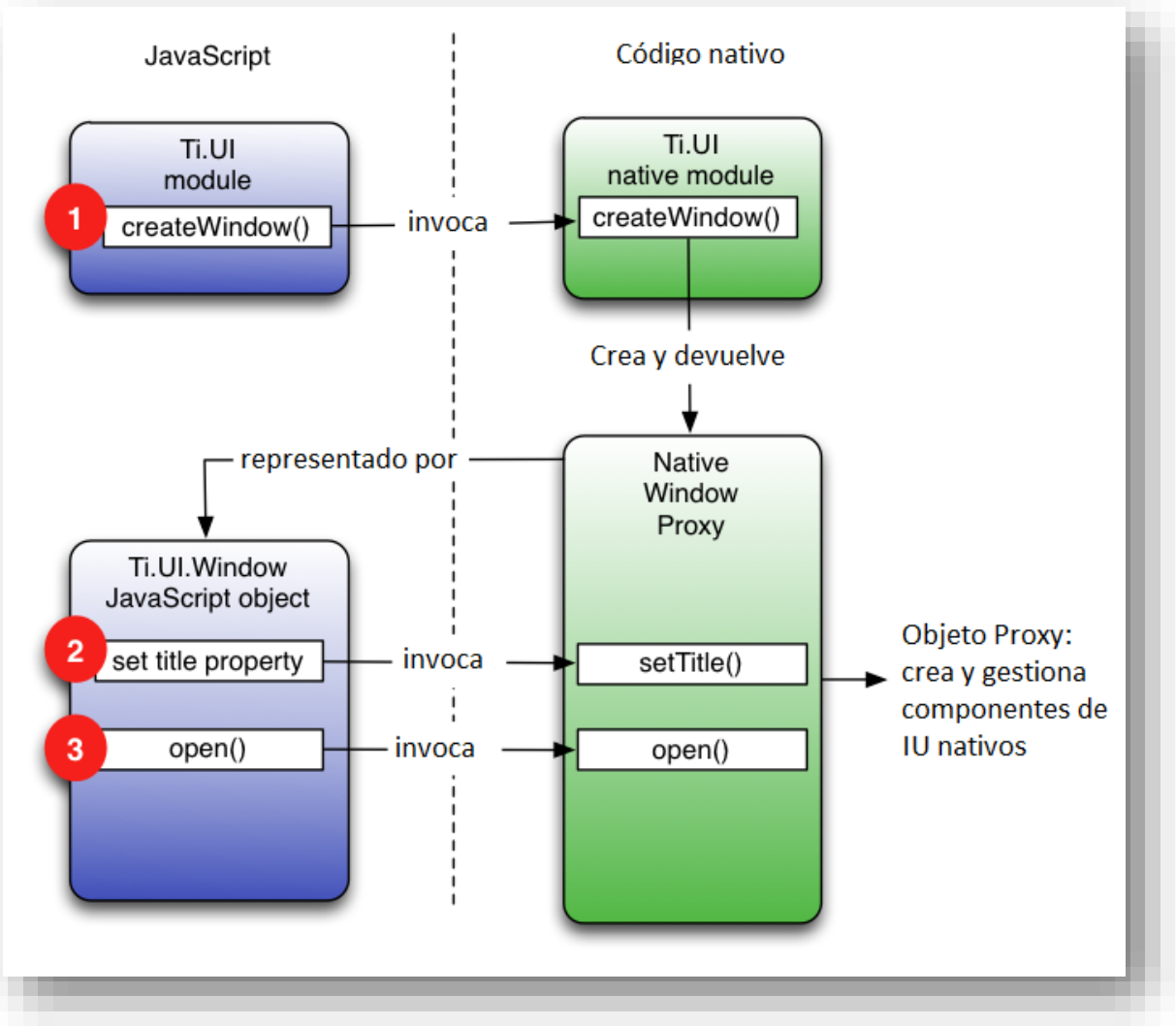


Ilustración 22. Representación de lo que hace el código cuando es ejecutado

En la representación anterior, aunque el objeto *win* parezca ser un simple objeto JavaScript, es importante tener en cuenta que habrá realmente dos objetos: el objeto JavaScript y el objeto proxy nativo.

Sólo las propiedades y métodos que estén ligados al objeto proxy nativo van a resultar en invocaciones al objeto proxy. Por ejemplo, si establecemos una propiedad arbitraria a un objeto después de su creación, esta se va a añadir al objeto JavaScript pero no al objeto proxy:

```
var win = Ti.UI.createWindow();

win.myProp = "Some other string";

win.title = "Hello World";
```

En el ejemplo anterior, la propiedad título (propia del objeto proxy) va a derivar en la invocación al método set del objeto proxy nativo, mientras que la propiedad *myProp* sólo se va a establecer en el objeto JavaScript y no se va a pasar a la capa nativa.

8 APLICACIÓN DEL DSL

Para poner en práctica el uso del DSL a través de la plataforma Titanium, se ha generado un prototipo en cada una de las tres plataformas más utilizadas en la actualidad: Android, Windows Phone e iOS.

Como ya se comentó en apartados anteriores, tenemos dentro del directorio de código fuente, el directorio *Sounds\build,* donde se encuentran:

- Android
- iPhone
- Windows

En cada uno de estos directorios se genera y configura un proyecto en cada una de las plataformas. De manera que podremos abrir su contenido con los IDE Eclipse para Android, Visual Studio para Windows y XCode para iPhone.

El prototipo en la plataforma presenta el siguiente modelo:

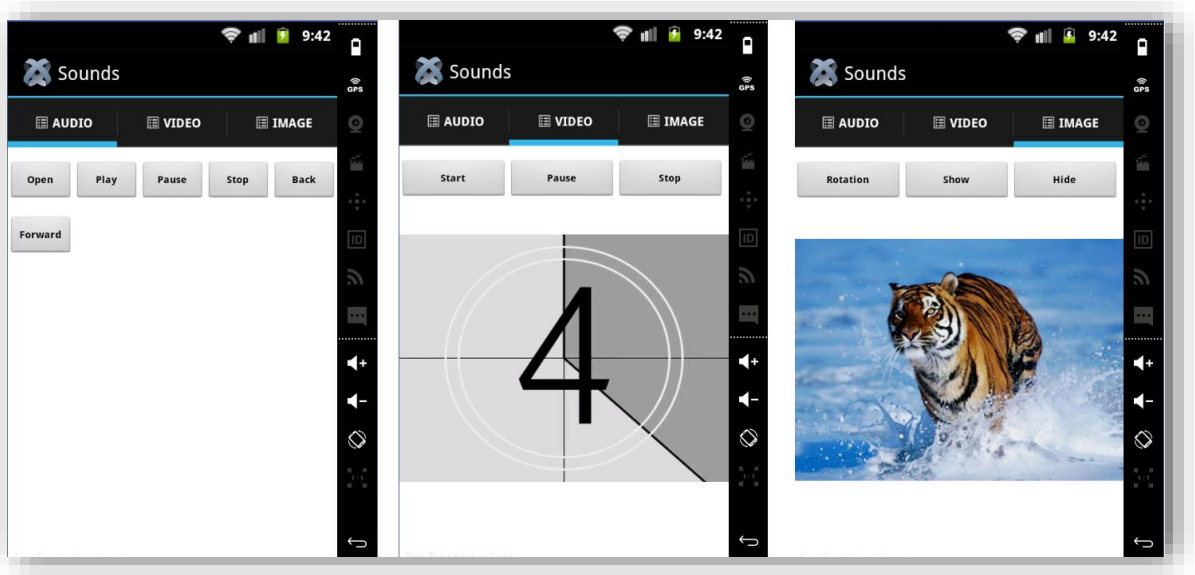


Ilustración 23. Modelo del prototipo en Android

El prototipo prescinde de validaciones JavaScript (por ejemplo que el usuario pulse dos veces sobre el botón de abrir archivo cuando este ya ha sido abierto) porque se considera que este no es el objetivo principal del prototipo.

Presenta tres opciones principales:

- ***Audio**:* Presenta las opciones básicas para manipular un archivo de audio.
  - ***Open***: Abre un archivo de audio localizado dentro del proyecto.
  - ***Play**:* Reproduce el archivo
  - ***Pause:*** Pausa la reproducción del archivo
  - ***Stop:*** Termina la reproducción
  - ***Backward:*** Cada pulsación implica que se resten unos segundos a la reproducción en curso.
  - ***Forward:*** Cada pulsación implica que se sumen unos segundos a la reproducción en curso.
- ***Video:*** Presenta las opciones básicas para manipular un video. Las opciones tienen la misma funcionalidad que en el caso del audio.
- ***Image:*** Se presenta una imagen sobre la que se puede aplicar una rotación de 90º y hacer visible u ocultarla.

9 CONCLUSIONES

Este trabajo ha servido para acercarnos por un lado al mundo de los lenguajes DSL y por otro a las herramientas multiplataforma para aplicaciones móviles.

Por un lado hemos podido comparar cómo según la tecnología avanza, el código puede ser expuesto a usuarios de negocio y audiencia no técnica, y gracias a los DSL esta labor será más sencilla de aplicar. En oposición tenemos el código de programación tradicional que por lo general se diseña para ser leído y entendido por programadores. Gracias a los DSL podemos desarrollar código expresivo que mejore la legibilidad y que permite una forma más sencilla de comunicación con los expertos en la materia que no sean técnicos.

Por otra parte se han visto las ventajas del desarrollo multiplataforma a través de la plataforma Titanium. Se han analizado diferentes enfoques (nativo, web móvil, web híbrida, cross compiled) y diferentes plataformas, y se ha generado un prototipo que ha reforzado la idea de que con el desarrollo de aplicaciones multiplataforma conseguimos reducción de costes y tiempo de desarrollo, y la curva de aprendizaje de este tipo de frameworks supone menos esfuerzos que para los desarrollos nativos.

Además este trabajo me ha servido para relacionar entre sí los conocimientos adquiridos en las otras asignaturas del máster, por ejemplo, los conocimientos adquiridos en Arquitecturas orientadas a servicios me han servido para valorar las ventajas que podrían obtener los dispositivos móviles integrándolos a las arquitecturas orientadas a servicios accediendo por ejemplo a servicios web que me permitirían acceder a beneficios como

escalabilidad, portabilidad e interoperabilidad. En el caso de que hubiese relación entre la aplicación móvil y un servidor, por ejemplo, para intercambio de información, podría haber aplicado los conocimientos de sistemas difusos de apoyo a la toma de decisiones para configurar el comportamiento del servidor. Gracias a la asignatura de computación ubicua, podría haber planteado la integración de la aplicación móvil en el entorno de la persona, por ejemplo estableciendo una comunicación transparente e inteligente con otros dispositivos del entorno para intercambiar música o información. Por último, la asignatura de Arquitectura para Sistemas Software me ha servido para tener presente formas de arquitectura de aplicaciones que podría haber aplicado.

10 TRABAJOS FUTUROS

Dentro de los posibles trabajos futuros nos encontramos con dos posibles caminos:

Por un lado, en lo que a desarrollo multiplataforma se refiere, se podría pensar en profundizar en el trabajo con la plataforma Titanium o probar a desarrollar con otros entornos multiplataforma para poder comparar la calidad de las aplicaciones resultantes. También se podría pensar en desarrollar una aplicación que haga uso de otras APIs como el uso de mapas o geolocalización.

Por otra parte, en lo que a DSL se refiere, se podría pensar en un lenguaje con mayor expresividad, bien en el ámbito de los recursos multimedia o bien dentro de otro contexto.

11 PASOS PARA REGENERAR EL PROTOTIPO

Si se desea incorporar nueva funcionalidad al prototipo desarrollado, bastaría con abrir con cualquier editor el fichero JavaScript asociado al punto donde queramos incorporar esa funcionalidad. Por ejemplo si queremos añadir un nuevo botón de recortar audio, abriríamos el fichero AudioTab.js, incorporamos el código correspondiente de un nuevo botón junto a la captura de su evento asociado, en el modelo semántico añadiríamos la lógica asociada a recortar audio, y en el constructor de expresiones la invocación a esa nueva funcionalidad, siguiente la mima línea de llamadas que ya existe. Para compilar el código en cada una de las plataformas, seguiríamos los pasos descritos en el apartado de *Generación de código en cada una de las plataformas.*

12 REFERENCIAS

[1]	http://www.gartner.com/newsroom/id/2996817
[2]	http://www.informit.com/articles/article.aspx?p=2143148&seqNum=1
[3]	http://developer.android.com/guide/appendix/media-formats.html
[4]	http://blogs.msdn.com/b/flecoqui/archive/2014/05/28/building-audio-video-applications-based-on-player-framework-v2-0-for-windows-phone-8-windows-8-1-and-windows-phone-8-1.aspx
[5]	https://msdn.microsoft.com/es-es/library/windows/apps/hh184843(v=vs.105).aspx
[6]	https://msdn.microsoft.com/en-us/library/windows/apps/jj206957%28v=vs.105%29.aspx
[7]	https://msdn.microsoft.com/en-us/library/windows/apps/system.windows.media.imaging.bitmapimage(v=vs.105).aspx
[8]	https://msdn.microsoft.com/en-us/library/windows/apps/jj206969(v=vs.105).aspx
[9]	http://www.geekchamp.com/articles/creating-theme-friendly-ui--in-wp7-using-opacitymask
[10]	https://developer.apple.com/library/prerelease/ios/documentation/MediaPlayer/Reference/MediaPlayer_Framework/index.html#//apple_ref/doc/uid/TP40006952
[11]	https://developer.apple.com/library/prerelease/ios/documentation/MediaPlayer/Reference/MPMusicPlayerController_ClassReference/index.html#//apple_ref/occ/cl/MPMusicPlayerController
[12]	https://developer.apple.com/library/prerelease/ios/documentation/MediaPlayer/Reference/MPMoviePlayerController_Class/index.html#//apple_ref/occ/cl/MPMoviePlayerController
[13]	https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImageView_Class/
[14]	Dickson, Jared, "Xamarin Mobile Development" (2013). Technical Library. Paper 167. http://scholarworks.gvsu.edu/cistechlib/167
[15]	https://cordova.apache.org/
[16]	https://jquerymobile.com/
[17]	http://dojotoolkit.org/features/mobile
[18]	https://www.sencha.com/products/touch/
[19]	http://cordova.apache.org/docs/en/4.0.0/
[20]	https://build.phonegap.com/
[20.a]	http://docs.phonegap.com/en/edge/cordova_media_media.md.html
[20.b]	https://github.com/macdonst/VideoPlayer
[21]	http://xamarin.com/
[22]	Apurva P. Pawar, Vandan S. Jagtap, Mamta S. Bhamare. *Survey on Techniques for Cross Platform Mobile Application Development.* http://ijarcet.org/wp-content/uploads/IJARCET-VOL-3-ISSUE-10-3551-3558.pdf
[23]	http://xamarin.com/studio
[24]	http://developer.xamarin.com/guides/
[24.a]	http://developer.xamarin.com/guides/android/application_fundamentals/working_with_audio/
[24.b]	http://developer.xamarin.com/recipes/android/media/video/play_video/
[24.c]	http://developer.xamarin.com/recipes/ios/media/video_and_photos/play_a_video_using_mpmoviplayercontroller/
[25]	http://docs.rhomobile.com/en/2.2.0/rhodes/rhostudio-eclipse
[25.a]	http://docs.rhomobile.com/en/5.0.0/api/mediaplayer
[26]	https://developer.appcelerator.com/doc/kitchensink
[27]	https://marketplace.appcelerator.com/home
[28]	https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/JavaScript_Compiler
[29]	http://www.appcelerator.com/blog/2012/02/what-is-a-titanium-proxy-object/

[30]	Martin Fowler. Domain-Specific Languages. http://martinfowler.com/books/dsl.html
[31]	http://docs.appcelerator.com/platform/latest/#!/guide/Installing_Oracle_JDK
[32]	https://developer.android.com/sdk/installing/index.html?pkg=tools
[33]	https://www.genymotion.com
[34]	https://www.microsoft.com/en-us/download/details.aspx?id=43719

Tabla 2. Referencias

13 SIGLAS ABREVIATURAS Y ACRÓNIMOS

CLI	Interfaz de Línea de Comandos, del inglés Command Line Interface. Es un método que permite a los usuarios dar instrucciones a un programa informático a través de una línea de texto simple.
PATH	Forma de referenciar un archivo o directorio en un sistema de archivos de un sistema operativo determinado.
API	Abreviatura de Interfaz de Programación de Aplicaciónes, del inglés Application Programming Interface. Es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son utilizadas generalmente en las bibliotecas.
SDK	Se entiende por Kit de Desarrollo de Software o SDK (siglas en inglés de Software Development Kit) al conjunto de herramientas de desarrollo que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, etc.
APP	Aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles.
XAML	Del inglés eXtensible Application Markup Language, Lenguaje Extensible de Formato para Aplicaciones en español. Es el lenguaje de formato para la interfaz de usuario para la Base de Presentación de Windows y Silverlight, el cual es uno de los “pilares” de la interfaz de programación de aplicaciones .NET en su versión 3.0.
OCR	Del inglés *Optical Character Recognition*. Generalmente conocido como reconocimiento de caracteres, es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente partir de una imagen, símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos, así podremos interactuar con estos mediante un programa de edición de texto o similar.
MVC	El Modelo Vista Controlador es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componenetes para la representación de la información, y por otro lado para la interaccción del usuario.

Tabla 3. Siglas, abreviaturas y acrónimos