Estudio de alternativas para el desarrollo del proyecto FastPark

Memória presentada por XAVIER BOUBÉS BARGALLÓ para optar a la titulación de Máster Oficial de Software Libre

Especialidad: Desarrollo de aplicaciones.
Consultor: Gregorio Robles Martínez

Índice de contenido

Revisiones del documento

Fecha modificación	Modificado por	Comentario
05/03/2011	Xavier Boubés	Versión inicial 0.1 - Entrega Pac 1
26/03/11	Xavier Boubés	Versión 0.2 - Entrega Pac 2 Modificaciones: Pasar a código nativo (Rhodes) Pasar a código nativo (Appcelerator) Pasar a código nativo (PhoneGap) Pruebas realizadas (PhoneGap) Pruebas realizadas (Rhodes)
16/05/2011	Xavier Boubés	Versión 0.3 - Entrega Final Modificaciones: Funcionalidades y plataformas (Appcelerator) Creación de un proyecto (Appcelerator) Resultados Conclusiones

Introducción.

WorldSening [1] es una compañía de *hardware* y *software* que utilizando las más recientes tecnologías ofrece soluciones completas en servicios industriales mediante la utilización de sensores inalámbricos locales y globales.

FastPark es una plataforma para la gestión de las zonas públicas de aparcamiento en entornos urbanos mediante una red de sensores inalámbricos y la utilización de las tecnologías de geolocalización de los dispositivos móviles.

El proyecto FastPark está asentado en dos pilares fundamentales: la instalación de las redes de sensores y el desarrollo de una aplicación que pueda funcionar en los principales dispositivos móviles actuales.

De esta aplicación móvil se deberán crear dos versiones. La primera de ellas estará destinada al usuario del estacionamiento y la segunda a la persona encargada de controlar estos lugares. La apariencia y funcionalidad de ambas aplicaciones será muy similar. La versión que ejecutará el usuario mostrará los aparcamientos disponibles más cercanos a su posición actual, en cambio el encargado de controlar las zonas de aparcamiento verá en un mapa los vehículos que han sobrepasado el tiempo máximo de estacionamiento.

Dicha aplicación móvil debe funcionar en el mayor número de dispositivos posible. La intención inicial es intentar ser capaces de reutilizar los esfuerzos de desarrollo que se realicen en una plataforma para el resto, por lo que se ha decidido realizar un estudio de viabilidad de las principales herramientas de desarrollo “cross-platform”. Este tipo de herramientas permiten la creación de aplicaciones capaces de ser ejecutadas en la mayoría de dispositivos móviles actuales.

Objetivos

La finalidad de este documento es ofrecer un punto de vista imparcial sobre las herramientas de desarrollo “cross platform” existentes, realizando una valoración de cada una de ellas siempre teniendo en cuenta las características del proyecto que WorldSensing desea implementar.

Este documento, una vez finalizado, debería convertirse en una herramienta que ofrezca la información suficiente como para poder tomar una decisión sobre la posible utilización de herramientas “cross-platform” para el desarrollo de FastPark y en caso afirmativo cual de las herramientas estudiadas sería la más adecuada.

Para realizar este estudio se ha tomado como referencia las herramientas de desarrollo “Cross-platform” más utilizadas, estas son PhoneGap, Titanium Appcelerator y Rhomobile.

Desarrollo

PhoneGap

PhoneGap ® encapsula una aplicación web para que sea posible ser ejecutada como una aplicación nativa sea cual sea la plataforma móvil de destino. Este tipo de aplicaciones no difiere en absoluto a una aplicación web distribuida que se ejecute en el navegador del dispositivo móvil.

El código de la aplicación utilizará las extendidas tecnologías html, javascript i CSS i se ejecuta utilizando el navegador web del dispositivo móvil. Lo que realmente aporta PhoneGap és un enlace entre javacript y las APIs nativas del dispositivo móvil donde se está ejecutando la aplicación web.

Funcionalidades y plataformas:

Funcionalidad	iPhone	Windows Mobile	Android	BlackBerry OS 6+	Palm	Symbian
Accelerómetro	✔	✔	✔	✔	✔	✔
Cámara	✔	✘	✔	✔	✘	✔
Brújula	✔	✘	✔	✘	✘	✘
Contactos	✔	✔	ⓘ	✔	✘	✔
Ficheros	✘	✘	✔	✔	ⓘ	✘
Geolocalización	✔	✔	✔	✔	✔	✔
Grabación Áudio	ⓘ	ⓘ	✔	✘	✘	✘
Notificación (sonido)	✔	✔	✔	✔	✔	✘
Notificación (vibración)	✔	✔	✔	✔	✘	✔
Almacenamiento	✔	✘	ⓘ	✔	✔	✘

✔: Funcionalidad soportada ✘: Funcionalidad no soportada ⓘ: Funcionalidad en desarrollo

Pasar a código nativo

PhoneGap tiene dos componentes principales [2]:

1. El API JavaScript que expone las funcionalidades nativas al navegador web.
2. El código nativo especifico de cada plataforma que está conectado a estas librerías javascript.

Con estas librerías PhoneGap permite realizar tareas genéricas como acceder a los datos de geolocalización o a los contactos del dispositivo móvil.

De todos modos, aunque el rendimiento de JavaScript ha mejorado tremendamente en los últimos años, aún existen procesos o tareas que solo pueden ser realizadas utilizando código nativo. Por otro lado también se deben considerar los servicios en segundo plano a nivel de servidor o incluso, pensando en la seguridad de la aplicación, funcionalidad de negocio que no se desea que sea descargada por el cliente.

En PhoneGap, para solucionar todas las anteriores limitaciones existen los plugins. Estos componentes deben ser escritos para cada una de las plataformas soportadas, concretamente se debe implementar:

1. Un fichero JavaScript por cada una de las plataformas que se desean soportar
2. El ejecutable en código nativo para cada plataforma soportada, por ejemplo un java para Android o un fichero .h y uno .m para IPhone.

Aunque ambos ficheros JavaScript deberían compartir la misma interfaz, existen pequeñas diferencias en la implementación de los mismos que hacen necesario realizar una versión para cada plataforma.

Creación de un plugin en Android [3]

El código nativo que se ejecutará será una clase Java que extenderá de la clase Plugin, a continuación se muestra un simple ejemplo que muestra los ficheros contenidos en la tarjeta SD del dispositivo:

```
public class DirectoryListPlugin extends Plugin {

	public static final String ACTION="list";

	@Override
	public PluginResult execute(String action, JSONArray data, String callbackId) {
		Log.d("DirectoryListPlugin", "Plugin Called");
		PluginResult result = null;
		if (ACTION.equals(action)) {
			try {
				String fileName = data.getString(0);
				JSONObject fileInfo = getDirectoryListing(new File(fileName));
					Log.d("DirectoryListPlugin", "Returning "+
			fileInfo.toString());
				result = new PluginResult(Status.OK, fileInfo);
```

```
                        } catch (JSONException jsonEx) {
                                Log.d("DirectoryListPlugin", "Got JSON Exception "+
                                        jsonEx.getMessage());
                                result = new PluginResult(Status.JSON_EXCEPTION);
                        }
                }
                else {
                        result = new PluginResult(Status.INVALID_ACTION);
                        Log.d("DirectoryListPlugin", "Invalid action : "+action+" passed");
                }
        return result;
        }

        private JSONObject getDirectoryListing(File file) throws JSONException {
                JSONObject fileInfo = new JSONObject();
                fileInfo.put("filename", file.getName());
                fileInfo.put("isdir", file.isDirectory());
                if (file.isDirectory()) {
                        JSONArray children = new JSONArray();
                        fileInfo.put("children", children);
                        if (null != file.listFiles()) {
                                for (File child : file.listFiles()) {
                                        children.put(getDirectoryListing(child));
                                }
                        }
                }
                return fileInfo;
        }
}
```

Una vez implementado el código nativo del plugin debemos crear la interfaz JavaScript que expondrá su método execute, a continuación se muestra un ejemplo:

```
var DirectoryListing = function() {
        }
        DirectoryListing.prototype.list = function(directory,successCallback, failureCallback){
                return PhoneGap.exec(successCallback, failureCallback, 'DirectoryListPlugin',
                 'list', [directory]); };
                PhoneGap.addConstructor(function() {
                //Register the javascript plugin with PhoneGap
                PhoneGap.addPlugin('directorylisting', new DirectoryListing());
                //Register the native class of plugin with PhoneGap
                PluginManager.addService("DirectoryListPlugin","com.trial.phonegap.plugin.director
        ylisting.DirectoryListPlugin");
});
```

En este punto el Plugin ya puede ser invocado desde la aplicación web, por ejemplo del siguiente modo:

```
<!DOCTYPE HTML>
<html>
<head>
<title>PhoneGap</title>
<script type="text/javascript" charset="utf-8" src="phonegap-0.9.3.js"></script>
<script type="text/javascript" charset="utf-8" src="directorylisting.js"></script>
<script type="text/javascript" charset="utf-8">
  document.addEventListener('deviceready', function() {
  var btn = document.getElementById("list-sdcard");
  btn.onclick = function() {
```

```
   window.plugins.directorylisting.list("/sdcard",
   function(r){printResult(r)},
   function(e){log(e)}
   );
   }
   btn.disabled=false;
 }, true);

function printResult(fileInfo){
 var innerHtmlText=getHtml(fileInfo);
 document.getElementById("result").innerHTML=innerHtmlText;
}

function getHtml(fileInfo){
var htmlText="<ul><li>"+fileInfo.filename;
if(fileInfo.children){
for(var index=0;index<fileInfo.children.length;index++){
htmlText=htmlText+getHtml(fileInfo.children[index]);
}
}

htmlText=htmlText+"</li></ul>";
return htmlText;
}

</script>
</head>
<body >

   <!-- Button -->
   <input disabled id="list-sdcard" type="button" value="List SDCard Contents"  />
   <hr>
                      <!-- Place Holder for placing the SD Card Listing -->
   <div id="result"></div>
   <hr>

</body>
</html>
```

Creación de un plugin para IPhone [4]

A continuación se muestra un ejemplo de un Plugin que lee códigos de barras, para empezar se debe implementar tanto la interficie .h:

```
#import "PhoneGapCommand.h"
#import "ZXingWidgetController.h"
#import "QRCodeReader.h"
@interface BarcodeScanner : PhoneGapCommand <ZXingDelegate> {
  <strong>NSString* successCallback;</strong>
  <strong> NSString* failCallback;</strong>
}
@property (nonatomic, copy) NSString* successCallback;
@property (nonatomic, copy) NSString* failCallback;
- (void) scan:(NSMutableArray*)arguments withDict:(NSMutableDictionary*)options;
@end
```

Como el ejecutable .m：

```
#import "BarcodeScanner.h"
```

```
@implementation BarcodeScanner
@synthesize successCallback, failCallback;
- (void) scan:(NSMutableArray*)arguments withDict:(NSMutableDictionary*)options
{
 NSUInteger argc = [arguments count];

 if (argc < 1) {
  return;
 }
 self.successCallback = [arguments objectAtIndex:0];
 if (argc > 1) {
  self.failCallback = [arguments objectAtIndex:1];
 }

 ZXingWidgetController *widgetController = [[ZXingWidgetController alloc] initWithDelegate:self
showCancel:YES OneDMode:NO];
 QRCodeReader* qrcodeReader = [[QRCodeReader alloc] init];
 NSSet *readers = [[NSSet alloc ] initWithObjects:qrcodeReader,nil];
 [qrcodeReader release];
 widgetController.readers = readers;
 [readers release];
 [[super appViewController ]  presentModalViewController:widgetController animated:YES];
 [widgetController release];
}
- (void)zxingController:(ZXingWidgetController*)controller didScanResult:(NSString *)result {
 NSString* jsCallBack = [NSString stringWithFormat:@"%@(\"%@\");", self.successCallback, [result
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding] ];
 [self writeJavascript: jsCallBack];
 [[super appViewController ]  dismissModalViewControllerAnimated:NO];
}
- (void)zxingControllerDidCancel:(ZXingWidgetController*)controller {
 [self writeJavascript: [NSString stringWithFormat:@"%@();", self.failCallback]];
 [[super appViewController ]  dismissModalViewControllerAnimated:YES];
}
@end
```

Posteriormente deberemos implementar el código javascript que expondra los métodos nativos anteriormente creados:

```
var BarcodeScanner = function() {

}

BarcodeScanner.Type = {
        QR_CODE: "QR_CODE"
}

BarcodeScanner.prototype.scan = function(types, success, fail) {
   return PhoneGap.exec("BarcodeScanner.scan", GetFunctionName(success), GetFunctionName(fail),
types);
};

PhoneGap.addConstructor(function()
{
        if(!window.plugins)
        {
        window.plugins = {};
        }
        window.plugins.barcodeScanner = new BarcodeScanner();
});
```

Es necesario remarcar que el codigo JavaScript que debe crearse para Iphone es diferente ya que para manejar la comunicación entre el ejecutable ObjC y javascript se utiliza la función writeJavaScript(), en cambio en android simplemente se propaga un objeto del tipo PluginResult.
También en Android es necesario registrar el servicio para que el JavaScript puede acceder a la clase Java, en cambio en IPhone se puede realizar una simple llamada del tipo Classname.funcion.

Para realizar una prueba del plugin solo deberemos ejecutar el siguiente JavaScript:

```
window.plugins.barcodeScanner.scan( BarcodeScanner.Type.QR_CODE, function(result) {
        alert("We got a barcode: " + result);
    }, function(error) {
        alert("Scanning failed: " + error);
    }
);
```

Características Importantes

Interacción con Android

Android es la única plataforma móvil que incorpora un servidor web en en el propio sistema operativo [5]. Dicho servidor web se utiliza para enviar mensajes, eventos i resultados desde el código Java hasta el código javascript que se ejecuta en un navegador. En el sentido contrario las clases Java son llamadas directamente desde javascript [6], por lo que no es necesario enviar la llamada nativa mediante Java. Por descontado este funcionamiento es transparente para el programador y gracias a las librerías de PhoneGap utilizando el método PhoneGap.exec() y extendiendo la clase Plugin en la Actividad principal.

Soporte multi-thread

En Javascript no existen los procesos multi-thread, solo hay un hilo de ejecución. De todas maneras las llamadas a las funcionalidades nativas de Android se hacen de manera asíncrona utilizando Ajax e implementado los métodos de respuesta correcta y de respuesta errónea (callback). Este funcionamiento previene que la aplicación deje de responder mientras se realizan estas llamadas. Por cada una de estas llamadas se crea un nuevo hilo de ejecución en el servidor web.

PhoneGap no soporta nativamente la ejecución de threads en paralelo, aunque existe la alternativa de utilizar web workers[7] de HTML 5, esta tecnología permite ejecutar código javascript en distintos hilos de ejecución.

Integración con GWT

Existen proyectos que permiten comunicarse con las librerías javascript, para PhoneGap existe uno en GitHub y otro en Google Code y otro para Titanium® también en Google Code utilizando el lenguaje de programación Java desde GWT.

De todos modos el proyecto asociado a Titanium® hace más de dos meses que no tiene actividad, en cambio los dos de Phonegap® gozan de buena salud y tiene numerosas aportaciones.

Gracias a estas librerías no seria necesario utilizar excesivo código "nativo" en la aplicación GWT además la velocidad de programación y depuración del código Java es superior a la del javascript.

A continuación se muestra un ejemplo simple de utilización de web workers, este sería el código fuente de una página html: [9]

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>Worker example: One-core computation</title>
 </head>
 <body>
  <p>
        <p id="ini"    style="display:block">Esperando a que se inicie el proceso.</output>
        <p id="process" style="display:none">Procesando...</output>
        <p id="result"  style="display:none">Proceso finalizado. Iteraciones realizadas: <output
        id="n"></output></p>
  <script>
        function iniciarProceso(){
        var worker = new Worker('worker.js');
        document.getElementById('ini').style.display='none';
        document.getElementById('process').style.display='block';
        document.getElementById('result').style.display='none';

        worker.onmessage = function (event) {
        document.getElementById('process').style.display='none';
        document.getElementById('result').style.display='block';
        document.getElementById('n').textContent = event.data;
        }
        }
  </script>
  <button onclick="javascript:iniciarProceso()">Iniciar proceso </button>
 </body>
</html>
```

Y este sería el código del fichero worker.js:

```
var n = 1;
while (n<1000000000) {
  n++;
}
postMessage(n);
```

Este proceso cargaría la página mostrando inicialmente el literal "Esperando a que se inicie el proceso.", una vez se pulse el botón se ejecutaría el código definido en el fichero worker.js, un bucle que ejecutará n iteraciones. Cuando se lanza el proceso el literal mostrado en pantalla se sustituirá por "Procesando... ". Una vez finalizado el bucle se actualizará el literal en pantalla con el mensaje "Proceso finalizado." y se indicará el número de iteraciones realizadas por el código javascript.

Esta tecnología permite definir hilos de ejecución en la parte cliente de la aplicación sin necesidad de utilizar interpretes de scripts o código de servidor. De todas maneras esta solución solo estaría soportada en navegadores que soporten HTML 5, por ejemplo este código no funcionaria en ninguna versión de internet explorer, incluyendo su versión mòbil.

Creación de un proyecto.

Requisitos para crear una aplicación en Android.

- Eclipse
- Android SDK
- ADT plugin
- Librerías de PhoneGap

Creación de un proyecto.

- Se crea un nuevo proyecto Android desde eclipse.
- Crear dos nuevos directorios en la raíz del proyecto::
    - /libs
    - /assets/www
- Copiar el archivo phongap.js descargado previamente a /assets/www.
- Copiar el archivo phonegap.jar descargado previamente a /libs
- Realizar los siguientes cambios en la clase principal del proyecto:
    - Cambiar la clase de la que extiende de Activity a DroidGap
    - Reemplazar la llamada a setContentView() por super.loadUrl("file:///android_asset/www/index.html"); donde index.html será la página principal de nuestra aplicación web.
    - Añadir el import com.phonegap.*;
- Si se produce un error de compilación en este punto se debe acceder a las propiedades del proyecto y en el apartado Build Paths > Configure Build Paths añadir en jar phonegap-0.x.x.jar al proyecto.
- Pegar el texto siguiente en el fichero AndroidManifest.xml justo debajo del texto versionName:

```
<supports-screens
android:largeScreens="true"
android:normalScreens="true"
```

```
android:smallScreens="true"
android:resizeable="true"
android:anyDensity="true"
/>
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Añadir el atributo `android:configChanges="orientation|keyboardHidden"` al tag activity del mismo fichero.
- Una vez realizados estos cambios ya se puede copiar el contenido de la aplicación web dentro del directorio /assets/www y ejecutarla en cualquier dispositivo mobil android.

Pruebas realizadas.

Para realizar pruebas en todas las herramientas de desarrollo, se ha implementado un script PHP que devuelve un objeto JSON de tamaño fijo.

Se creará una aplicación para cada una de las plataformas que parseará este JSON. Para valorar esta prueba se tendrá en cuenta tanto el tiempo dedicado a la programación de esta aplicación como el tiempo que invierta la aplicación en parsear este documento.

Este programa se creó usando GWT [10](Google Web Toolkit). La aplicación web resultante se ha ejecutado en un móvil Android utilizando PhoneGap. Los resultados obtenidos son los siguientes:

Tiempo invertido en el desarrollo de la aplicación: 8 horas.
Número de objetos JSON parseados: 10.000.
Número de atributos por cada objeto JSON: 2.
Tiempo invertido en el proceso: 14 segundos.

Por motivos de seguridad Google no permite que se obtenga código Javascript de un servidor diferente al que se está ejecutando la aplicación web [11], para que la aplicación sea capaz de obtener el objeto JSON remoto se ha tenido que utilizar el siguiente código:

```
    public native static void getJson(int requestId, String url,
                StockWatcher handler) /*-{
        var callback = "callback" + requestId;

        // [1] Creamos dinámicamente un tag script
        var script = document.createElement("script");
        script.setAttribute("src", url + callback);
        script.setAttribute("type", "text/javascript");

        // [2] Se define el método Callback
        window[callback] = function(jsonObj) {
        // [3]
handler.@com.google.gwt.sample.stockwatcher.client.StockWatcher::handleJsonResponse(Lcom/google/g
wt/core/client/JavaScriptObject;)(jsonObj);
                window[callback + "done"] = true;
        }

        // [4] Reintento en caso de no obtener respuesta
        setTimeout(
                        function() {
                                if (!window[callback + "done"]) {

handler.@com.google.gwt.sample.stockwatcher.client.StockWatcher::handleJsonResponse(Lcom/google/g
wt/core/client/JavaScriptObject;)(null);
                                }

                                // [5] Se elimina el tag script creado.
                                document.body.removeChild(script);
                                delete window[callback];
                                delete window[callback + "done"];
                        }, 1000);

        // [6] Se inserta el tag script al body del documento.
        document.body.appendChild(script);
    }-*/;
```

Anotaciones de la implementación:

- [1] la función comienza creando un elemento <script>. El atributo src apunta a la URL desde la que se obtendrán los datos JSON envueltos en la función callback.
- [2] La función callback se define como un objeto de la ventana del navegador. Este recibe como parámetro el objeto javascript que contiene los datos JSON obtenidos remotamente.
- [3] El método callback pasa como parámetro el objeto JSON al método Java handleJsonResponse que se encarga de realizar las acciones oportunas con los datos obtenidos.
- [4] Se define un timeout de un segundo para repetir la llamada en caso de no haber obtenido respuesta la primera vez.
- [5] Una vez se ha completado la ejecución del código se elimina el elemento <script> creado y la función callback de la ventana.

- [6] Finalmente se llama a la función appendChild() para cargar dinámicamente nuevo script creado al body del documento HTML.

Appcelerator Titanium

Appcelerator Titanium®, al contrario de PhoneGap® consigue realizar el proceso de conversión del código html / javascript a una aplicación casi nativa, en líneas generales Titanium® se encarga de analizar y pre-procesar el código javascript para convertirlo en código entendible por la implementación nativa de las APis de Titanium. En resumen la única diferencia que habría entre una aplicación nativa y una aplicación hecha con Titanium® sería que las llamadas a las APIs del sistema de la segunda deben pasar por las de Titanium®. Con esta implementación se consigue que el aspecto general de la aplicación sea idéntico a una aplicación nativa.

Las aplicaciones de escritorio de Titanium deben ser implementadas en PHP, Ruby o Python, las aplicaciones para dispositivos móviles mediante HTML5 i javascript.

Funcionalidades y plataformas:

Appcelerator Titanium solo soporta desarrollos en Android i Iphone, teniendo en fase de beta las aplicaciones para Blackberry, actualmente no está previsto el desarrollo de la compatibilidad para windows mobile.

Funcionalidad	iPhone	Android
Accelerómetro	✓	✓
Cámara	✓	✓
Brújula	✓	✓
Contactos	✓	ⓘ
Ficheros	✗	✓
Geolocalización	✓	✓
Grabación Áudio	ⓘ	✓
Notificación (sonido)	✓	✓
Notificación (vibración)	✓	✓
Almacenamiento	✓	ⓘ

✓: Funcionalidad soportada ✗: Funcionalidad no soportada ⓘ: Funcionalidad en desarrollo

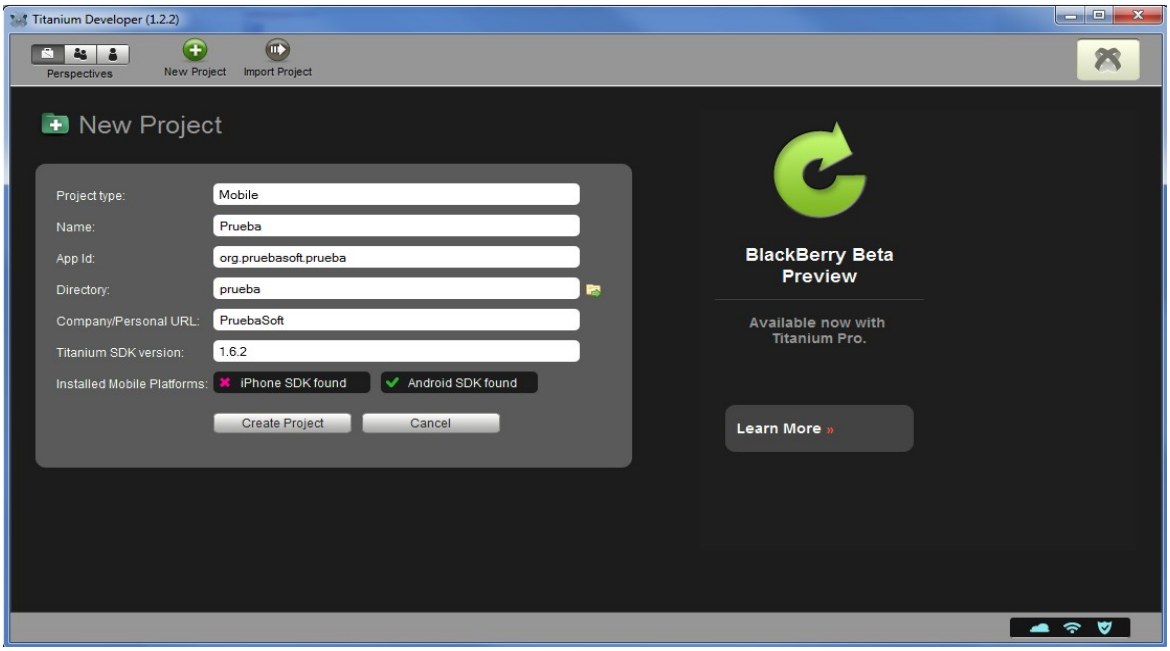
Creación de un proyecto.

Requisitos para crear una aplicación en Android.

Python 2.6
Git
Scons
Android SDK
ADT plugin
Titanium Developer

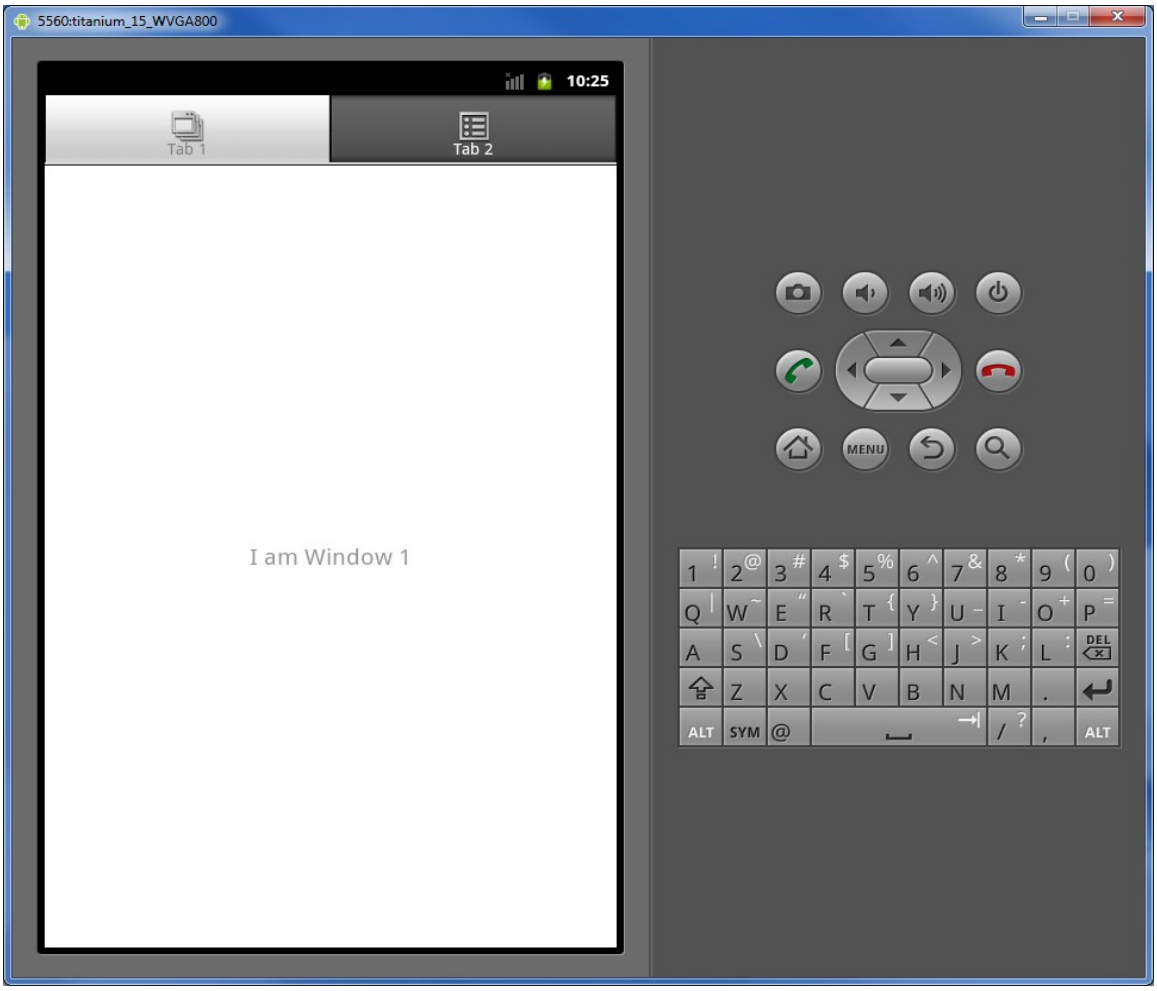
Creación de un proyecto:

Gracias a Titanium Developer podremos crear la estructura básica de un nuevo proyecto, pulsando en el botón “New project” iniciaremos el asistente de creación del mismo:



Actualmente no existe un IDE con el que poder hacer modificaciones de los componentes del proyecto, por lo que una cez creado el proyecto las modificaciones deberán hacerse utilizando un editor externo.

El proyecto generado es una demostración de como funcionan las pestañas. Una vez generado ya puede ser ejecutado en un emulador de Android:



Pasar a código nativo.

Apccelerator ha solucionado este problema permitiendo la creación y distribución de elementos independientes que se encarguen de la ejecución de las tareas que sea necesario ejecutar utilizando código nativo. Dichos elementos reciben el nombre de Módulos. [12]

Creación y funcionamiento de un módulo para Android.

Un modulo de Titanium Appcelerator puede constar de los siguientes elementos:

- Proxy: Una clase base que representa el enlace entre el código JavaScript y el código nativo.
- ViewProxy: Una versión especializada de un proxy que sirve para renderizar vistas.
- View: La representación visual de los componentes que Titanium puede renderizar.

- Modulo: Un tipo especial de proxy que describe una funcionalidad específica.

Los proxys son los elementos que exponen la API entre JavaScript y Java, por ejemplo Titanium.UI.createView es un proxy. Deben seguir los siguientes patrones:

- Las clases proxy deben extender de la clase KrollProxy.
- Los constructores de clase solo podrán recibir el parámetro TiContext.

A diferencia de los Proxys, los Módulos son enlazados estáticamente por el API y solo puede haber una sola instancia del mismo cargada en memoria, ya que utilizan el patrón de desarrollo Singleton, por ejemplo Titanium.UI y Titanium.App son dos ejemplos de módulos.

Los Módulos deben heredar de KrollModule (que a su vez hereda de KrollProy)

En un Módulo debe exister una clase Módulo, pero puede haber varios o ningún Proxies, ViewProxies o Views.

Creación de un modulo.

Para crear un nuevo modulo se utiliza el comando titanium create pasando como parámetros el nombre del módulo, la plataforma sobre la cual se ejecutará y la ruta del SDK de la plataforma, por ejemplo:

```
titanium create --platform=android --type=module --name=calc
--id=org.appcelerator.calc --android=/path/to/android-sdk
```

Si la operación se ha realizado con éxito se creará un nuevo directorio llamado calc dentro del directorio donde se ejecutó el comando.

Empaquetando el módulo.

Los módulos de appcelerator se distribuyen comprimidos en un fichero zip, por defecto este fichero se genera utilzando ant. El proceso de empaquetado se puede lanzar llamando al ejecutable ant desde el directorio del módulo. En la primera ejecución el resultado obtenido debería ser similar al siguiente:

```
$ ant
Buildfile: /Users/marshall/Code/test/test_modules/calc/build.xml
init:
  [mkdir] Created dir: /Users/marshall/Code/test/test_modules/calc/build/classes
  [mkdir] Created dir: /Users/marshall/Code/test/test_modules/calc/dist
process.annotations:
  [javac] Compiling 2 source files to/Users/marshall/Code/test/test_modules/calc/build/classes
  [javac] Note: [KrollBindingGen] Running Kroll binding generator.
  [javac] Note: [KrollBindingGen] No binding data found, creating new data file.
  [javac] Note: [KrollBindingGen] Found binding for module Calc
  [javac] Note: [KrollBindingGen] Found binding for proxy Example
compile:
  [javac] Compiling 2 source files to/Users/marshall/Code/test/test_modules/calc/build/classes
  [copy] Copying 1 file to/Users/marshall/Code/test/test_modules/calc/build/classes
dist:
```

```
  [jar] Building jar: /Users/marshall/Code/test/test_modules/calc/dist/calc.jar
  [zip] Building zip: /Users/marshall/Code/test/test_modules/calc/dist/org.appcelerator.calc-
android-0.1.zip

BUILD SUCCESSFUL
Total time: 1 second
```

Distribución del modulo.

Para utilizar el módulo en un proyecto appcelerator se deberán seguir los siguiente pasos:

- Copiar el fichero comprimido .zip a la raíz del proyecto.
- Modificar el fichero tiapp.xml y añadir su declaración dentro del tag <ti:app>:

```
<!-- $MODULE_VERSION should bethe same as "version" in the module manifest -->
<modules>
  <module version="$MODULE_VERSION">$MODULE_ID</module>
  <!-- For example, ifwewere adding the calc module: -->
  <module version="0.1">org.appcelerator.calc</module>
</modules>
```

Utilizar la función *require* para cargar el módulo en el código del programa:

```
var Calc = require('org.appcelerator.calc');
```

Creación de un módulo para Iphone en Appcelerator Titanium

De la misma manera que se ha creado el módulo para Android, la creación para Iphone se realiza uilizando el comando titanium create:

```
titanium
create --platform=iphone --type=module --dir=~/tmp --name=test --id=
com.test
```

Para crear un Proxy se debe crear una interfaz que extienda de la clase TiProxy, por ejemplo:

```
#import "TiProxy.h"
@interface FooProxy : TiProxy {
}
@end
```

Y la implementación debe ser:

```
#import "FooProxy.h"
@implementation FooProxy
@end
```

Por defecto, Titanium necesita que la clase contenga el sufijo Proxy para indicar que la clase es un proxy.

Un proxy expone los métodos y propiedades simplemente usando sintaxis Objective-C. Para exponer un método un Proxy debe tener un método firma (method signature), por ejemplo para exponer la propiedad:

```
@property(nonatomic,readwrite,assign) NSString* propertyName;
```

Se usará:

```
-(void)setPropertyName:(id)value
{
}

-(id)propertyName
{
  return @"foo";
}
```

Pruebas realizadas:

Para realizar la prueba en Appcelerator se ha desarrollado una aplicación javascript que utilizando sus librerias invoca a un script PHP que devuelve un array de objetos JSON. Los resultados obtenidos son los siguientes:

Tiempo invertido en el desarrollo de la aplicación: 6 horas.
Número de objetos JSON parseados: 10.000.
Número de atributos por cada objeto JSON: 2.
Tiempo invertido en el proceso: 16 segundos.

El código javascript de este pequeño programa es el siguiente:

```
loader.open("GET","http://boubes.dyndns.org/stockPrices.php");
loader.onload = function() {
  var data = eval('('+this.responseText+')');
  var count = 0;
  for (var i = 0; i < data.length; i++) {
        var myObject = eval('(' + data[i] + ')');
        count++;
  }
loader.send();
```

Características Importantes

Soporte multi-thread

Appcelerator soporta la ejecución de threads independientes utilizando su api Javascript. Estas librerías utilizan el concepto de contexto de ejecución de manera que cada nueva ventana se ejecuta en un contexto diferente y por lo tanto en un hilo de ejecución independiente del resto [13].

Para definir un nuevo hilo de ejecución se puede hacer del siguiente modo:

```
var networkThread = Ti.UI.createWindow
({
        url: "networkThreadStuff.js"
});
networkThread.open();
```

Rhomobile.

Rhomible ha creado una herramiento de desarrollo cross-platform llamada Rhodes. Este producto es muy similar Titanium, con la particularidad de que ha sido desarrollado en lenguaje de programación Ruby.

Las vistas, al igual que en el resto de productos, son renderizadas mediante el interpretador web de la plataforma que ejecuta la aplicación (en IPhone UIWebView). Se puede utilizar CSS i javasript para modificar el aspecto de las mismas, de hecho Rhodes incluye un robusto sistema de estilos CSS que optimizan la aparienza HTML de las vistas para cada dispositivo, al contrario que Titanium, Rhodes muestra componentes con apariencia similar a los componentes nativos de cada plataforma, no realmente los nativos.
Para la parte del controlador, al contrario que Titanium que utiliza javascript, Rhodes interpreta el código escrito en Ruby para convertirlo a código nativo de cada plataforma, objectiveC para IPhone y Java para Android o Blackberry.

Por que Ruby?

Rhodes eligió Ruby como lenguaje de desarrollo de su herramienta, sus argumentos son los siguientes:

Lenguaje de *scripting*. Incremento de la productividad debido a la utilización de expresiones regulares en la creación de clases y funciones
Menos código. Las aplicaciones Ruby suelen tener menos de una tercera parte del tamaño en c código que su equivalente en Java. Esto hace que las aplicaciones sean más fáciles de mantener.
Lenguaje de programación en expansión O'Reilly estima un crecimiento del 50% cada año desde que se creó Ruby.
Gran comunidad. En GitHub, RubyForge y en definitiva por todo Internet se han creado plugins en forma de gems para Ruby. El éxito de Ruby on Rails ha hecho que se disparé la creación de estos add-ons, plenamente compatibles con Rhodes (versión escritorio).
Diseño orientado a objetos. Gracias al cual ha sido fácil la creación de un Framework desde cero, como Rhodes o Rails. y de las librerías de utilidades existentes.

Funcionalidades y plataformas:

Capability	iPhone	Windows Mobile	BlackBerry	Android
GeoLocation	✔	✔	✔	✔
PIM Contacts	✔	✔	✔	✔
PIM Calendar	✔	✔	✔	✔
Camera	✔	✔	✔	✔
Barcode	✔	✔	✔	✔
Date/Time picker	✔	✔	✔	✔
Menu	✔	✔	✔	✔
Toolbar	✔	✔	✘	✔
Tab Bar	✔	ℹ	✘	✔
Nav Bar	✔	ℹ	✘	✔
Signature Capture	✔	ℹ	ℹ	✔
Audio/Video capture	ℹ	ℹ	ℹ	ℹ
Bluetooth	✔	✔	✔	✔
Push	ℹ	ℹ	✔	✔
Screen rotation	✔	ℹ	✔	✔
Native Maps	✔	ℹ	✔	✔
Alerts/Audio File Playback	✔	✔	✔	✔
Ringtones	✔	✔	✔	✔

✔: Funcionalidad soportada ✘: Funcionalidad no soportada ℹ: Funcionalidad en desarrollo

Pasar a código nativo.

Existen tres maneras de extender el funcionamiento de Rhodes . Se pueden añadir gems de Ruby soportados por la herramienta. Se pueden crear nuevas extensiones utilizando el SDK de la plataforma de ejecución móvil.
Es este documento únicamente se tratará la creación de extensiones utilizando código nativo del SDK de la plataforma de ejecución, en Rhodes estas modificaciones reciben el nombre de "extensiones nativas".
Para crear una extensión nativa llamada ext-name, en primer lugar se deberá crear un directorio que contendrá el contenido de la extensión (app-name/extensions/ext-name) y se crea el fichero ext.yml con el siguiente contenido:
entry: Init_ext_name
libraries: ["ext-name"]
El método Init_ext_name será el que rhodes invoque en el inicio de la aplicación, por lo que este método deberá realizar las tareas de inicialización que se consideren oportunas. Si existe mas de una libreria deben incluirse sus nombres separados por comas.
Se debe tomar el script de construcción de la aplicación y realizar una copia en el directorio ext, adaptandolo a las necesidades de la nueva extensión nativa, modificando el nombre del proyecto por el nombre de la extensión y modificando la ruta de la ubicación. Con este cambio se conseguirá que el proyecto cree el fichero libext-name.a (en Android) o ext-name.lib (en Windows) al construir el proyecto.
Notas para Android:

- Si la extensión nativa usa librerías adicionales, estas deben de ser accesibles y añadidas al script de construcción del proyecto o copiadas al directorio temporal TARGET_TEMP_DIR en la ejecución del script de construcción.
- Si la extensión nativa usa código Java en el script de construcción debe crearse un fichero llamado ext_build.files en el directorio temporal de construcción (TARGET_TEMP_DIR) que contenga los nombres completos, uno por línea, incluyendo la ruta de las clases java.
- Si la extensión nativa crea threads nativos, se deben enlazar con la máquina virtual para que sea capaz de llamar a los métodos de su contexto. Para conseguir este funcionamiento se deben usar las funciones rho_nativethread_start / rho_nativethread_end, por ejemplo:
-

```
void *thread_routine(void *arg)
{
  void *q = rho_nativethread_start();
  .....
  rho_nativethread_end(q);
  return NULL;
}
```

Ejemplos de casos de éxito:

Wikipedia Mobile: Según la web de Rhodes la aplicación oficial de Wikipedia ha sido desarrollado con rhodes y añaden que se utilizó aproximadamente el 20% del código original de la aplicación realizada previamente en Objective C para Iphone. De todas maneras, ya que este proyecto es de código abierto, se ha comprobado como actualmente el código que hay publicado en su repositorio vuelve a estar en Objective C. Parece ser que Wikipedia creó una nueva versión de la aplicación para iphone utilizando la plataforma de desarrollo nativa y desechó el proyecto previo [14]
Humama: Esta aplicación permite a los asegurados del seguro médico de la empresa Humana encontrar servicios de emergencia y revisar el balance de su cuenta de salud. Está disponible tanto en la App Store como en Android Market.

Pruebas realizadas:

Para realizar la prueba en Rhodes se ha creado un proyecto que realiza la misma tarea que en el resto de herramientas: invoca a un script PHP que devuelve un array de objetos JSON. Los resultados obtenidos son los siguientes:

Tiempo invertido en el desarrollo de la aplicación: 8 horas.
Número de objetos JSON parseados: 10.000.
Número de atributos por cada objeto JSON: 2.
Tiempo invertido en el proceso: 22 segundos.

El código del controlador de este programa es el siguiente:

```
uri = URI.parse("http://boubes.dyndns.org/")
array = Net::HTTP::Get.new("stockPrices.php", 'Accept' => 'application/json')
array.each_with_index {|x, y| puts "#{x} => #{y}" }
end
```

Características Importantes

Soporte multi-thread

Rhodes utiliza un interprete de Ruby gracias al cual soporta múltiples hilos de ejecución.

El siguiente fragmento [15] de código crea tantos hilos de ejecución como elementos tenga el array de direcciones web pages:

```
pages = %w( www.rubycentral.com
            www.awl.com
            www.pragmaticprogrammer.com
          )
threads = []
```

```
for page in pages
  threads << Thread.new(page) { |myPage|

    h = Net::HTTP.new(myPage, 80)
    puts "Fetching: #{myPage}"
    resp, data = h.get('/', nil )
    puts "Got #{myPage}:  #{resp.message}"
  }
end
threads.each { |aThread|  aThread.join }
```

Los hilos de ejecución son creados con la llamada Thread.new, al cual se le pasa el código que ejecutará cada
uno de ellos. Cada hilo hará un acceso a la dirección web recibida como parámetro. Una vez finalizado el proceso
se mostrará el siguiente resultado:

```
Fetching: www.rubycentral.com
Fetching: www.awl.com
Fetching: www.pragmaticprogrammer.com
Got www.rubycentral.com:  OK
Got www.pragmaticprogrammer.com:  OK
Got www.awl.com:  OK
```

Resultados

A continuación se muestra una tabla comparativa de los datos que se pueden extraer de todas las pruebas realizadas a las herramientas de desarrollo “Cross-platform” analizadas en este documento:

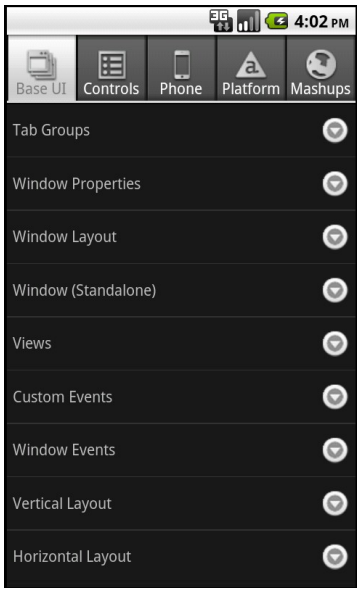
	Titanium Appcelerator	PhoneGap	RhoMobile
Lenguaje de programación	Javascript y HTML 5	Javascript y HTML 5	Ruby y HTML 5
Ide de desarrollo	Titanium Studio (actualmente en beta)	No, aunque se puede utilizar Eclipse	Eclipse
MultiThread Nativo	Si	No	Si, con Ruby
Licencia	Apache 2.0	MIT License	MIT License
Android	✔	✔	✔
Iphone	✔	✔	✔
BlackBerry	✔ (beta)	✔	✔
Symbian	✘	✔	✔
Palm	✘	✔	✘
Windows Mobile	✘	✔	✔
Segundos para parsear 10.000 objetos JSON	16	14	22
Soporte código nativo	Si	Si	Si
Soporte desarrollo GWT	Si	Si	No

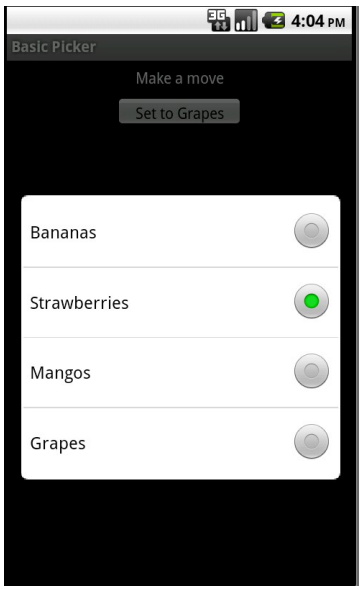
Otros comentarios

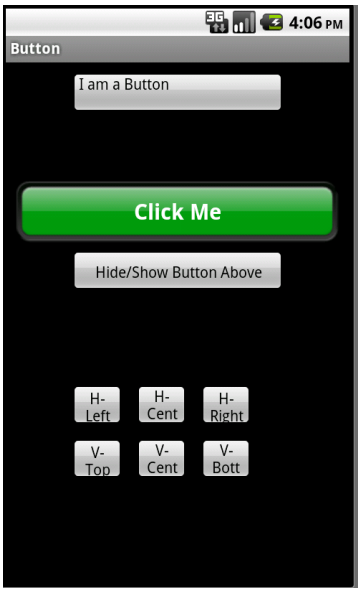
Más allá de los resultados objetivos existen otros factores para evaluar las herramientas descritas en este documento.

Aspecto final de la aplicación

El primero de estos factores es el aspecto final de la aplicación resultante. En el caso de PhoneGap no existe ningún diseño predefinido, por lo que este es creado por el programador desde cero mediante hojas de estilo (CSS). Rhomobile funciona del mismo modo, aunque al crear un nuevo proyecto ya crea sus correspondientes archivos CSS que dotan a la aplicación de una apariencia similar a una aplicación nativa. Appcelerator va un paso más allà y gracias a su API javascript incorpora a la aplicación resultante widgets nativos de la plataforma en ejecución, a continuación se muestran algunos ejemplos del aspecto de una aplicación de prueba desarrollada en Appcelerator para Android:







Coste de adaptación.

Bajo el punto de vista del esfuerzo requerido en implantar una nueva herramienta y teniendo en cuenta que WorldSensing tiene experiencia en desarrollos Java utilizando GWT, la herramienta mas adecuada sería PhoneGap. Esta herramienta cuenta con un desarrollo independiente que expone todas las funcionalidades ofrecidas en Javascript utilizando código Java, del mismo modo en que lo hace GWT. Utilizando esta tecnología un desarrollo para dispositivos móviles utilizando PhoneGap y GWT se podria hacer íntegramente en Java y HTML, sin utilizar Javascript. Teniendo en cuenta este factor, Rhomobile seria el que saldría peor parado, el hecho de utilizar Ruby supondría un tiempo de adaptación que se podría evitar utilizando otra de las herramientas estudiadas.

Documentación disponible.

Es muy importante la documentación que cada herramienta facilita a los desarrolladores, la inexistencia o escasez de la misma puede hacer que el tiempo de desarrollo se alargue considerablemente. En este aspecto Titanium Appcelerator es el que sale mejor parado, en su sitio web ha publicado ejemplos y tutoriales de prácticamente todas las funcionalidades que son soportadas por su herramienta, además han desarrollado una aplicación donde muestran con ejemplos el funcionamiento y aspecto visual de los widgets nativos. En el lado opuesto se puede encontrar RhoMobile, su documentación es escasa, faltan ejemplos, en su wiki hay enlaces a otros sitios que ya no existen o simplemente no han documentado algunas funcionalidades, como por ejemplo la utilización de Plugins en Windows Mobile.

Soporte de la comunidad.

Otra factor a tener en cuenta es el soporte de la comunidad de desarrolladores y la documentación disponible para cada herramienta "Cross-platform". En este aspecto PhoneGap [17] goza de mejor salud que sus competidores con una comunidad de usuarios de más de cinco mil miembros activos y varias decenas de proyectos asociados a su herramienta. Rhomobile [18], por su parte, dispone de una comunidad de poco más de 1000 participantes. En cambio Titanium [19] ha descuidado por completo la interacción con desarrolladores y colaboradores externos y ha limitado las aportaciones de la comunidad a traducciones de su wiki y aportaciones en los foros.

Plataformas soportadas y acceso al Hardware.

Debido a que FastPark será utilizado por el personal encargado del control de las zonas de aparcamiento debería ser posible compatible con la mayoría de las plataformas existentes, incluyendo Windows Mobile. Este sistema operativo es ampliamente utilizado por los dispositivos utilizados en las administraciones públicas [16]. Por otro lado uno de los requerimientos de FastPark para la versión del controlador de aparcamiento es que fuera capaz de tomar fotografías de los vehículos estacionados incorrectamente. La única herramienta que soporta la captura de imágenes en Windows Mobile es Rhomobile, aunque dicha funcionalidad no está documentada. Para solucionar este problema se podría utilizar código nativo, por lo que PhoneGap continuaría siendo una alternativa viable para el desarrollo. En cambio Titanium Appcelerator no está soportado para Windows Mobile, por lo que un desarrolla utilizando esta herramienta no cumpliría este requerimiento.

Soporte Multi-Thread.

De cara a aprovechar al máximo el hardware del dispositivo móvil que ejecute FastPark, las tareas que requieran de la utilización exhaustiva del procesador se ejecutarán en paralelo. Tanto Titanium Appcelerator como Rhomobile soportan el multi-thread de forma nativa, el primero a través de Javascript, el segundo utilizando Ruby. Por contra PhoneGap no soporta este tipo de tecnología, de todas maneras una alternativa viable podría ser la utilización de web workers.

Conclusiones.

En este punto se evaluará cual de las herramientas evaluadas en este documento es más adecuada según diferentes puntos de vista.

Según el aspecto final de la aplicación.

Este factor no debería ser decisivo de cara a elegir una herramienta “Cross-Platform” ya que FastPark, al menos en sus primera versión, no debería tener mucha carga de elementos gráficos. De todos modos teniendo en cuenta este punto de vista Titanium Appcelerator es la herramienta que ofrece el aspecto visual más similar a una aplicación nativa para Iphone y Android, aún así hay aspectos de la visualización que deben tenerse en cuenta por el programador, por ejemplo mostrar un botón “Volver” en Iphone y ocultarlo en Android (debido a que los móviles con Android ya incorporan un botón físico con esta funcionalidad).

Según la facilidad de uso y la documentación existente.

Bajo este punto de vista tanto Titanium Appcelerator como PhoneGap son buenas alternativas, el primero tiene una documentación más clara y completa, pero PhoneGap se integra correctamente con GWT y esto brinda una herramienta muy potente para el desarrollador, pudiendo crear una aplicación móvil sin necesidad de utilizar Javascript.

Según el rendimiento.

Tras las pruebas que se han hecho para crear este documento, la herramienta que consiguió unos mejores resultados en el test de parseo de un documento JSON fué PhoneGap. De todos modos y puesto que PhoneGap no soporta el uso de Multi-Thread, la opción más adecuada bajo este punto de vista sería Titanium Appcelerator.

Según la flexibilidad.

Teniendo en cuenta este factor PhoneGap es la alternativa. En el caso de Titanium Appcelerator y Rhomobile el desarrollador está sujeto a la utilización de los estilos y widgets ofrecidos por la herramienta. PhoneGap deja total libertad de elección.

Después de todos los datos expuestos en este documento y suponiendo que conociéndolos aún se considere una opción viable el hecho de utilizar una herramienta “Cross-platform” para el desarrollo de FastPark, se considera que la mejor opción, por su facilidad de uso y flexibilidad, es PhoneGap.

Lecciones aprendidas

Antes de iniciar esta colaboración no habia oido hablar de las herramientas “Cross-platform”, de hecho en la primera reunión que mantuve con WorldSensing mi primera reacción cuando se dijo que se iba a valorar la utilización de estas herramientas, fué desaconsejar su uso por varias experiencias previas que habia tenido con aplicaciones similares para aplicaciones de escritorio. Una vez finalizado el estudio y habiendo analizado las herramientas en profundidad puedo decir que mi punto de vista ha cambiado considerablemente, si bien es cierto que algunas de ellas, como por ejemplo Rhomobile aún no están el todo maduras, hay que reconocer que tanto Titanium Appcelerator como PhoneGap son grandes proyectos que dificilmente caerán en el olvido.

Previamente ya tenia conocimientos teoricos de como funcionaba HTML5 y como podia acceder al hardware de un dispositivo, utilizar pequeñas bases de datos, etc. Pero gracias a este trabajo de podido comprovar de primera mano como están evolucionando las tecnologias web y que HTML 5 puede ser finalmente la plataforma de desarrollo que unificará los desarrollos de aplicaciones web, Google ya ha apostado fuerte por esta tecnologia y pocas veces se equivocan.

Por último solo añadir que he tenido la oportunidad de discutir los datos y resultados con Jordi Llosa y Jon Vadillo, responsables del desarrollo en WorldSensing, a los cuales agradezco que hayan ofrecido indicaciones e inestimable ayuda para que este proyecto haya llegado a buen cabo.

Bibliografía

1: Worldsensing (sitio en internet). Disponible en www.worldsensing.com. Último acceso el 11 de junio de 2011
2: PhoneGap (sitio en internet). Disponible en http://wiki.phonegap.com/w/page/36752779/PhoneGap-Plugins#Limitations. Último acceso el 15 de junio de 2011
3: Creación de un plugin en Android con PhoneGap (sitio en internet). Disponible en http://www.google.com/url?q=http%3A%2F%2Fwiki.phonegap.com%2FHow-to-Create-a-PhoneGap-Plugin-for-Android&sa=D&sntz=1&usg=AFQjCNHfjwAUx21HSe1O90oJnSe9m1NqAQ. Último acceso el 15 de junio de 2011
4: Creación de un plugin para Iphone (sitio en internet). Disponible en http://wiki.phonegap.com/How-to-Create-a-PhoneGap-Plugin-for-IPhone. Último acceso el 15 de junio de 2011
5: Servidor web en Android (sitio en internet). Disponible en http://groups.google.com/group/phonegap/browse_thread/thread/09ae2d6b592a9bce?pli=1. Último acceso el 15 de junio de 2011
6: Soporte Multi-Thread en PhoneGap (sitio en internet). Disponible en http://www.google.com/url?q=http%3A%2F%2Fwww.rgagnon.com%2Fjavadetails%2Fjava-0170.html&sa=D&sntz=1&usg=AFQjCNEFx7raTvuEoIPsfXy2bj7bEffaUg. Último acceso el 11 de junio de 2011
7: Web Workers en HTML 5 (sitio en internet). Disponible en http://dev.w3.org/html5/workers/. Último acceso el 11 de junio de 2011
9: Ejemplo de web workers (sitio en internet). Disponible en http://www.codeproject.com/KB/scripting/parallel-js.aspx. Último acceso el 11 de junio de 2011
10: GWT (sitio en internet). Disponible en http://code.google.com/intl/es-ES/webtoolkit/. Último acceso el 11 de junio de 2011
11: GWT - Cross Site Scripting (sitio en internet). Disponible en http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/tutorial/Xsite.html. Último acceso el
12: Pasar a código nátivo (sitio en internet). Disponible en http://www.google.com/url?q=http%3A%2F%2Fwiki.appcelerator.org%2Fdisplay%2Fguides%2FModule%2BDeveloper%2BGuide%2Bfor%2BAndroid&sa=D&sntz=1&usg=AFQjCNFISxEkO8H2l3XMEhxzMrTBlvH_Og. Último acceso el
13: Soporte Multi-Thread en Appcelerator (sitio en internet). Disponible en http://www.google.com/url?q=http%3A%2F%2Fdeveloper.appcelerator.com%2Fquestion%2F117199%2Fhow-to-create-multithreading-in-mobile-sdk&sa=D&sntz=1&usg=AFQjCNG4Rq5jDC8-Yxwq-qox-26-RqZ_IQ. Último acceso el
14: Cófigo fuente de wikipedia Mobile (sitio en internet). Disponible en https://github.com/wikimedia/wikipedia-iphone. Último acceso el 5 de junio de 2011
15: Multi-Thread en Rhomobile (sitio en internet). Disponible en http://www.ruby-doc.org/docs/ProgrammingRuby/html/tut_threads.html. Último acceso el 11 de mayo de 2011

17: Comunidad PhoneGap (sitio en internet). Disponible en http://groups.google.com/group/phonegap. Último acceso el 2 de junio de 2011
18: Comunidad en Rhomobile (sitio en internet). Disponible en http://groups.google.com/group/rhomobile. Último acceso el 2 de junio de 2011
19: Comunidad en Titanium (sitio en internet). Disponible en http://titaniumcommunity.com/index.php/category/titanium-community/. Último acceso el 2 de junio de 2011
16: Microsoft y la administración pública (sitio en internet). Disponible en http://www.microsoft.com/spain/aapp/articulos/default.mspx. Último acceso el