Desarrollo de Aplicaciones Multiplataforma con HTML5 para Teléfonos Inteligentes

Luis Vivas, Mauro Cambarieri, Nicolás García Martínez,
Horacio Muñoz Abbate, Marcelo Petroff

Laboratorio de Informática Aplicada - Universidad Nacional de Río Negro
{lvivas, mcambarieri, ngarcia, hmunoz, mpetroff}@unrn.edu.ar

Resumen. Este trabajo presenta una arquitectura para el desarrollo de software multiplataforma destinado a los teléfonos inteligentes, también llamados smartphones. Expone las ventajas en el desarrollo de aplicaciones web sobre las nativas - aquellas que se instalan en dispositivos móviles y fueron desarrolladas utilizando el lenguaje de programación compatible con el sistema del dispositivo. Asimismo se describen los frameworks y los kits de desarrollo de software disponibles actualmente bajo el paradigma del software libre. Se presentan alternativas para el desarrollo de la interfaz de usuario con soporte para HTML5 y computación en las nubes mediante el cual podemos exponer el servicio de datos y aplicación. Por último, se explican los aportes de HTML5 para aplicaciones móviles y las plataformas tecnológicas que lo soportan en la actualidad, como Android, iOS y BlackBerry – el grupo destacado en cantidad de dispositivos y tráfico en la red.

Abstract. This paper presents an architecture for the development of software for smartphones. It explains the advantages of web application development with respect to native applications - those installed on mobile devices that were developed using a programming language compatible with the operating system used by the device. It also describes frameworks and software development kits currently available under the free software paradigm. Alternatives are presented for developing the user interface with support for HTML5 and cloud computing through which we can expose the data service and application. It explains the contributions of HTML5 for mobile applications and platforms that currently Android, iPhone OS and BlackBerry - as a distinguished group based on the number of devices and network traffic.

Keywords: Teléfono inteligente, multiplataforma, software libre, aplicaciones web, HTML5

1. Introducción

Los teléfonos inteligentes tienen como características fundamentales una capacidad de cómputo y conectividad mayor que los móviles convencionales [1]. Estas hacen que el teléfono en algunos casos remplace el ordenador personal y lideren el consumo de datos de Internet [2].

En la actualidad los dispositivos dominantes son: Android, iPhone y BlackBerry. Estos poseen varios rasgos funcionales como, por ejemplo: cámara de Fotos, pantallas táctiles de alta definición, también llamadas pantallas capacitivas, Wi-Fi y 3G para conectividad de largo alcance, comunicación de alcance corto (Near Field Communication - NFC) que se utiliza para pagos de servicios y se conoce con el nombre de billetera digital. Los dispositivos también cuentan con sistema de posicionamiento global (GPS) para la geolocalización; varios tipos de sensores como giroscopio, acelerómetro, sensor de luz y proximidad. El corazón del hardware está compuesto por una máquina avanzada RISC (Advanced RISC Machine - ARM) y unidad de procesamiento gráfico (Graphics Processing Unit - GPU) potentes y de bajo consumo, gracias a los avances de los procesos de fabricación actuales, mediante los cuales pueden soportar toda la gama de características mencionadas [3].

Contando con las funcionalidades descriptas, el hardware no es suficiente para la correcta administración de los recursos y funcionalidades de los teléfonos inteligentes. Se necesitan sistemas operativos especialmente diseñados. Actualmente, los sistemas operativos mas relevantes son Android de Google, iOS de Apple y BlackBerry SO de RIM [2]. Android esta basado en Linux que es software de código abierto, mientras que iOS se basa en Mac OS X que es software privativo, al igual que BlackBerry SO.

Android, iOS y BlackBerry 10 sin teclado QWERTY (distribución del teclado que hace referencia a las seis primeras letras), se basan en un concepto de manipulación directa, usando gestos multitáctiles. Los controles se basan en: componentes deslizantes, interruptores y botones. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos (en inglés pinch to zoom), los cuales tienen diferentes funciones dependiendo del contexto de la interfaz. Esta nueva forma de interactuar con el sistema operativo se debe principalmente a las pantallas capacitivas – aquellas que poseen mejor calidad de imagen, tiempo de respuesta y permiten el uso de varios dedos a la vez.

El uso creciente de estos dispositivos, sus funcionalidades y las múltiples plataformas existentes, plantea un problema: cómo construir software para teléfonos inteligentes multiplataforma? Buscando una solución, este trabajo propone una arquitectura de software basada en estándares HTML5, discute las ventajas del desarrollo de aplicaciones web, y explica frameworks para el desarrollo de aplicaciones para estos dispositivos.

El trabajo esta estructurado de la siguiente manera. La sección 2 presenta los conceptos relacionados que permiten comprender la solución propuesta. La sección 3 explica la metodología propuesta para desarrollar aplicaciones multiplataforma. Finalmente, la sección 4 resume las conclusiones, lecciones aprendidas y líneas de investigación para trabajos futuros.

2. Conceptos Generales

2.1. Aplicaciones web y nativas

El desarrollo para dispositivos móviles tiene dos paradigmas bien diferenciados: nativo y web [4]. Una aplicación móvil web es la que necesita de un navegador web o browser para ejecutarse. La aplicación como los datos pueden estar alojados o consumirse de un servidor remoto, así como también obtenerse del dispositivo móvil. Las aplicaciones nativas son aquellas que se instalan en los dispositivos móviles y se desarrollan utilizando el lenguaje de programación compatible con el sistema operativo del mismo, embebidos en kits de desarrollo de software (software development kit - SDK). Los más destacados son Android SDK, iOS SDK y BlackBerry Java SDK.

Tanto las SDK de Android y BlackBerry usan Java como lenguaje de programación, en cambio las de iOS utilizan el lenguaje Objective-C (Lenguaje de programación utilizado en Mac OS X e iOS) [5][6][7]. Cada una de estas plataformas de desarrollo tiene sus particularidades, no sólo por el lenguaje de programación sino también por el framework de desarrollo.

Básicamente, las aplicaciones web se entregan como lenguaje de marcado de hipertexto (HyperText Markup Language - HTML) interpretado por un browser, el cual brinda una serie de características que aumentan las prestaciones de las aplicaciones.

Para el desarrollo de aplicaciones móviles web existen en la actualidad una serie de frameworks que hacen uso de las ventajas de HTML5, y de la plataforma WebKit[8], incorporada en los principales navegadores, como por ejemplo: Safari, Chrome y el navegador de BlackBerry; logrando aplicaciones muy fluidas utilizando, entre otras características, las pantallas táctiles y el GPS para la georeferenciación. Las características más importantes de HTML5 se explican en la sección siguiente.

Entre los frameworks más destacados podemos mencionar: Sencha Touch[9], jQuery[10] y Gwt Mobile[11]. Sencha Touch y jQuery se componen de una serie de librerías escritas en javascripts y estilos Cascading Style Sheets (CSS). Algunas de las ventajas de Sencha Touch frente a jQuery es una nueva arquitectura Modelo Vista Controlador (MVC - Model View Controller). MVC es un patrón de diseño que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes [12]. Otra de las ventajas de Sencha Touch es la documentación muy completa con muchos ejemplos para descargar y componentes muy logrados tanto funcional como visualmente. En contraposición, JQuery tiene una curva de aprendizaje más veloz. Gwt Mobile, a diferencia de los frameworks mencionados, se programa en lenguaje Java.

2.2 HTML5 [13]

En cualquier sitio web, la interfaz de usuario y los datos residen en un servidor específico, y estos deben volver a descargar cada vez que se solicita, lo cual genera un problema de rendimiento así como también un obstáculo insalvable cuando la conexión de un usuario es débil. HTML5 se pensó para resolver este tipo de problemas, pudiendo mantener la interfaz de usuario y/o datos en línea.

HTML5 provee caché de aplicación que permite almacenar el código HTML, CSS y javascript en el dispositivo móvil en la primera carga, como si fuera una instalación, permitiéndonos acceder a la aplicación fuera de línea como si fuera una aplicación nativa.

Una de las nuevas características es la llamada: localstorage (almacenamiento local), la cual permite almacenar cualquier variable que se desee de forma persistente, similar a las cookies (datos enviados desde un sitio web y se almacenan en el navegador web de un usuario mientras el usuario está navegando por un sitio web). Esta facilidad de almacenamiento local es más fácil y flexible de administrar. HTML5 también provee la base de datos SQLite (base de datos de código abierto).

Como característica relevante, HTML5 incorpora la integración con una interfaz de programación de aplicaciones (Application Programming Interface - API) de geolocalización, permitiendo acceder a la posición del móvil a través de los datos del GPS disponible en los dispositivos. Esta herramienta permite hacer una amplia gama de aplicaciones donde se requiere saber la posición del usuario, como por ejemplo: calcular recorridos, lugares de referencia cercanos, etc.

HTML5 incorpora nuevos tipos de campos de entrada para formularios con una propiedad llamada type (tipo), la cual permite diferenciar distintos tipos de datos de ingreso como texto, email, teléfono, número, colores, rango de números y fecha. El teclado virtual cambia de comportamiento según el tipo de datos. Ejemplo: si el tipo es numérico, el teclado presenta los dígitos para ingresar.

2.3 Computación en la Nube (Cloud Computing)

La computación en la nube, del inglés Cloud (metáfora de Internet) Computing, es un paradigma que permite ofrecer servicios de computación a través de Internet.

Para el desarrollo de aplicativos en dispositivos móviles, sean web o nativos, necesitamos consumir datos que la alimentan. Podemos mencionar una aplicación que nos proporcione el clima según la ciudad donde el usuario esté ubicado actualmente. En este tipo de servicio sería imposible tener la información almacenada en el dispositivo del usuario porque dicha información se crea dinámicamente con variables climatológicas. En estos casos la computación en la nube nos permite crear servicios y aplicaciones disponibles todo el tiempo en cualquier lugar donde tengamos acceso a Internet.

Para llevar a cabo una arquitectura utilizando Cloud Computing podemos utilizar appEngine de Google, que nos brinda una SDK para desarrollo de software, pudiendo elegir algunas alternativas de lenguajes de Programación como Python, Ruby o Java.

Trabajando con esta arquitectura dejamos de tener los problemas de infraestructura habituales que tiene una aplicación o servicio de alta disponibilidad, sean estos desde la conectividad hasta el suministro eléctrico. También, es una opción para pymes que no tienen una infraestructura para brindar un servicio con la escalabilidad y rendimiento que nos proporciona appEngine.

Google nos proporciona appEngine alojado en sus miles y miles de ordenadores distribuidos por todo el mundo, con un sistema de virtualización basado en Linux el cual esta modificado con un sistema de archivo distribuido propio llamado Google File System (GFS). Para los datos tendremos acceso a BigTable, motor de bases de datos creado por ellos mismos con las características de ser: distribuido, incremental y de alta eficiencia. Con este sistema, Google nos brinda todo su conocimiento en búsquedas para acceder a los datos en forma rápida y eficiente [14][15].

Para el manejo de datos utilizamos Objectify [16], Mapeo de Objeto Relacional (Object Relational Mapping - ORM) el cual nos permite trabajar con BigTable, que nos provee de simple y elegantes operaciones de manejo de datos, tales como: get, put, delete y query.

En el siguiente código se define la entidad persistente Car, como se inicia una transacción mediante begin, se crea, obtiene y elimina un objeto Car.

```
class Car {
       @Id String vin;
       String color;}

Objectify ofy = ObjectifyService.begin();
ofy.put(new Car("206", "azul"));
Car c = ofy.get(Car.class, "206");
ofy.delete(c);
```

3. Metodología propuesta para desarrollar aplicaciones multiplataforma

Para que una aplicación nativa funcione en todas las plataformas, hay que construir una aplicación para cada una de ellas. Todas tienen su SDK de desarrollo y lenguaje de programación. Entonces podemos decir que las aplicaciones web son la alternativa para el desarrollo multiplataforma a través del estándar HTML5.

El proceso de desarrollo para la construcción de software web se basa en el patrón MVC, el cual es un patrón de diseño muy estudiado y utilizado en la ingeniería de software, el cual separa en tres capas el modelo, el controlador y la vista. Para la vista optamos por usar Sencha Touch 2 y la SDK java de appEngine con Spring (framework de

código abierto de desarrollo de aplicaciones para la plataforma Java) para el Modelo y Controlador.

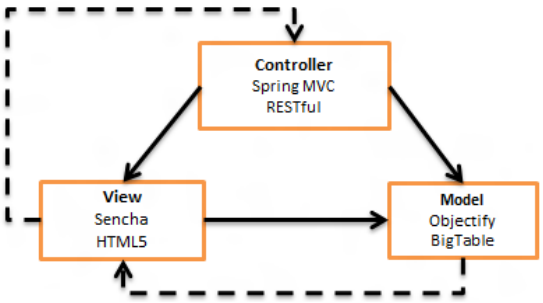

Fig. 1. Patrón Model View Controller - MVC

La comunicación entre controlador y vista es a través de objetos serializados mediante el formato json (javascript object notation, es un formato ligero para el intercambio de datos) mediante servicios de transferencia de estado representacional (representational state transfer - REST o RESTful) [16].

Esta nueva forma de comunicación ha supuesto una nueva opción a los servicios web. REST consiste en utilizar la especificación HTTP (protocolo de tranferencia de hipertexto - HyperText Transfer Protocol) [17]. Este fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP (simple object access protocol) que es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML[18] y a WSDL (web services description language o servicios web basados en el lenguaje de descripción de servicios web) que describe la interfaz pública a los servicios web [19]. Las grandes empresas del la web 2.0 (Yahoo, Google y Facebook entre otras), están usando REST, quienes marcaron como obsoletos a sus servicios SOAP y WSDL y pasaron a usar un modelo más fácil de usar, orientado a los recursos.

La última implementación de Spring y otras alternativas como Java Server Faces (java server faces, framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE - JSF2), tienen soporte completo con Restful además de una fácil y elegante implementación.

A continuación describimos como integrar Spring y Objectify [20]. El ejemplo busca obtener una lista de Objetos Car. En principio tenemos que importar la librería objectify.jar a nuestro proyecto, la cual tiene la clase com.googlecode.objectify.util.DAOBase [21], que será la base de nuestros objetos de acceso a datos (DAO - Data Access Object), que es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. Esta clase nos provee el acceso a la conexión de los datos a través de la instancia del método ofy().

```
public class ObjectifyDao<T> extends DAOBase
{ public Query<T> listAll()
        {Query<T> q = ofy().query(clazz);
          return q;
```

El segundo paso es crear el CarDao que extiende de ObjectifyDao para manejar la Clase persistente Car. Con la anotación @Repository podemos poner en el contexto de Spring los servicios que vamos a utilizar.

```
@Repository
public class CarDao extends ObjectifyDao<Car> {
       static {     ObjectifyService.register(Car.class);}
       public CarDao() {
              super(Car.class);
```

El siguiente paso es generar el servicio CarService mediante la anotación @Service. En los servicios de Spring se inyectan todos los Dao que necesitemos, en nuestro caso solo CarDao. La anotación @Autowired se encarga de hacer la inyección de controles en nuestro servicio.

```
@Service
public class CarService {
       @Autowired
       private CarDao carDao;
       public List<Car> findAll() {
              return this.carDao.listAll().list();
```

La última capa que vamos a utilizar con Spring es el Controlador, esta capa se comunica directamente con la interfaz de usuario. Con este controlador podríamos brindar servicios a tecnologías web pero también a tecnologías nativas como Android o iOS:

```
@Controller
@RequestMapping(value = "/car")
public class CarController {
       @Autowired
       private CarService carService;
       @RequestMapping(method = RequestMethod.GET,
       value = "/list", produces = "application/json")
       public @ResponseBody
       List<Car> carList() {
              return carService.findAll();
```

En este fragmento de código se destaca la anotación RequestMapping, y los valores **/car** y **/list** del método carList. Mediante el protocolo Http accediendo a la url

http://server:8888/car/list el cliente puede acceder al recurso sin tener que configurar o intercambiar un contrato de comunicación. El parámetro produces = "application/json" indica que el resultado sea transformado a formato json.
El resultado de recurso:

```
[{"vin":"206","color":"amarillo"},
{"vin":"ka","color":"rojo"},
{"vin":"mondeo","color":"blanco"}]
```

La última capa es la vista que consumirá la lista de Car en formato json. Al proyecto solo tenemos que agregar las librerías de javascripts, imágenes y estilos que están disponibles del sitio de Sencha. Este framework tiene una serie de componentes para consumir datos locales o remotos. El componente que usaremos para este caso será un Ext.data.Store [22], el cual nos permite configurar la url del recurso creado anteriormente.

```
Ext.create('Ext.data.Store', {
          fields: ['vin', 'color'],
          sorters: 'vin',
          groupField: 'color',
           autoLoad: true,
          proxy:{type: 'ajax',
                    url : '/car/list',
                    reader: { rootProperty: 'data',
                              totalProperty: 'total',
                              successProperty: 'success',
                              type: 'json'}
```

En el código se observa el tipo de almacenamiento (Store), los atributos que vamos a consumir definidos en la propiedad fields, por que atributo quiero ordenar y agrupar la información; también defino la url del recurso, en este caso como la vista y el controlador están en el mismo server el recurso se consume con un path relativo /car/list, pero sino fuera el caso, y esta aplicación estaría en otro servidor o la vista esta empaquetada en un dispositivo móvil, entonces la url podría ser : http://liaproyect.appspot.com/car/list. Por último queda explícito que nuestro store espera recibir un json con la propiedad type: 'json'. Con el store configurado solo nos queda definir una componente de lista con pueda dibujar la lista de Car. Sencha tiene el componente Ext.dataview.List [23].

```
xtype: 'list',
itemTpl: '<div class="contact">{vin}
<strong>{color}</strong></div>',
store: this.getStore(),
grouped: true,
emptyText: '<div>No tiene elementos</div>',
```

Como podemos apreciar la lista se define con xtype: 'list', los campos que se quieren mostrar los exponemos con itemTpl y la fuente de datos se declara con store.

El resultado de lo expuesto queda disponible en http://liaproyect.appspot.com y los fuentes del proyecto en https://liamobileweb.googlecode.com/svn/liaProyect.

4. Conclusión y Trabajos futuros

En este trabajo se presentó una arquitectura multiplataforma para el desarrollo de software para teléfonos inteligentes con tecnología web. Se basó en los estándares de HTML5 que son ampliamente soportados por las principales plataformas de móviles, esto fue probado en la aplicación http://liaproyect.appspot.com. Quedó explicado la diferencia entre el desarrollo de aplicaciones web y nativas, este último supone elaborar una aplicación para cada plataforma lo cual incrementa costos y tiempos, de acuerdo a los resultados obtenidos en nuestro laboratorio a través de la realización de aplicaciones.

Otro punto de estudio ha sido el consumo de datos mediantes servicios REST, alternativa que puede utilizarse para aplicaciones web y nativas, siendo una parte fundamental en la optimización del funcionamiento ágil de las aplicaciones que consumen datos de Internet, basado en los resultados obtenidos en el laboratorio.

Nos abrió distintos caminos de investigación la posibilidad futura de acceder a otros recursos de los teléfonos inteligentes, tales como: los contactos de la agenda, obtener imágenes utilizando la cámara de fotos, las alertas como el ringtone o la vibración del dispositivo, para brindar otros tipos de funcionalidades relacionadas con la informática ubicua o realidades aumentadas. Aplicaciones como estas son las denominadas híbridas [24] las que permitirían a las aplicaciones web acceder a las API de sistema y tiendas digitales.

Referencias

1. Wikipedia. 2012. Teléfono inteligente. http://es.wikipedia.org/wiki/ Teléfono_inteligente (accedido 10/07/2012).
2. Cisco Visual Networking index, Global mobile data traffic forecast update. 2011.
3. Proceso de Fabricación ARM Cortex 28nm. http://www.tsmc.com/tsmcdotcom/ PRListingNewsAction.do?action=detail&newsid=6781&language=E (accedido 12/07/2012).
4. 2012. Mobile HTML5 java developer. http://www.jboss.org/webinars (accedido 20/06/2012).
5. JBossCommunitty. 2012. SDK BlackBerry. https://developer.BlackBerry.com /java/ (accedido 10/06/2012).

6. Appe.2012. SDK iPhone. https://developer.apple.com/ (accedido 11/05/2012).
7. Google. 2012. SDK Android. http://developer.android.com/sdk/index.html (accedido 02/04/2012).
8. Apple, Kde. 2006. http://www.webkit.org/ (accedido 19/07/2012)
9. The jQuery Foundation. 2012. Framework Jquery para Móviles. http://jquerymobile.com/ (accedido 01/06/2012).
10. Gwtmobile. 2012. Framework Gwt mobile (Google web Toolkit) http://code. google. com/p/gwtmobile/ (accedido 21/02/2012)
11. Sencha Inc. 2012. Framework Sencha para móviles http://www.sencha. com/products/touch/ (accedido 10/11/2011).
12. Wikipedia. 2012. Patrón MVC. http://es.wikipedia.org/wiki/Modelo_ Vista_Controlador (accedido 10/07/2012).
13. HTML5rocks.2012. HTML5 lo nuevo y características slider. HTML5rocks. com (accedido 13/03/2012).
14. Google Developers. 2012. AppEngine. http://code.google.com/intl/es-ES/appengine/ (accedido 2/05/2012)
15. Google España. 2012. AppEngine Campus Party. http://www.youtube.com/watch?v=9Ocjqxhh3RQ. (acedido 10/05/2012).
16. Google App Engine. 2012. Objectify. http://code.google.com/p/objectify-appengine/wiki/IntroductionToObjectify (accedido 28/05/2012).
17. Navarro Marset, R. 2006. Servicios REST http://users.dsic.upv.es/~rnavarro/ NewWeb/docs/RestVsWebServices.pdf (1/07/2012).
18. W3C. 2010. Estandar HTML http://www.w3.org/MarkUp/ (accedido 10/07/2012)
19. W3C. 2007. SOAP http://www.w3.org/TR/soap/ (accedido 11/07/2012)
20. W3C. 2001 WSDL http://www.w3.org/TR/wsdl (accedido 11/07/2012)
21. Matías Molinas. 2012. Integración de Spring y Objectify http://fuse21.blogspot.com.ar/2012/04/spring-mvc-y-objectify-en-google-app.html (10/06/2012).
22. Google. 2011. Clase DAOBase http://objectify-appengine.googlecode.com/svn-history/r635/trunk/javadoc/com/googlecode/objectify/helper/DAOBase.html. (accedido 2/07/2012).
23. Sencha Inc. 2012. Ext.data.Store. http://docs.sencha.com/touch/2-0/#!/api/Ext.data.Store (accedido 10/07/2012).
24. Sencha Inc. 2012.Ext,dataview.list. http://docs.sencha.com/touch/2-0/#!/api/Ext.dataview.List (accedido 10/07/2012).
25. Kaiser, C. 2011. Aplicaciones móviles Hibridas. How to Develop Mobile Applications with Web-Technologies. Université the Fribourg Suisse. http://diuf.unifr.ch/main/is/sites/diuf.unifr.ch.main.is/files/documents/student-projects/eBiz_2011_Christian_Kaiser.pdf (accedido 21/04/2012).