Experimentando un uso no tradicional de PhoneGap

Agustina M. Zimbello[1] and Cecilia Challiol[1,2]

[1] LIFIA, Facultad de Informática, UNLP, 50 y 120 Primer Piso,
La Plata, Argentina
{azimbello,ceciliac}@lifia.info.unlp.edu.ar
http://www.lifia.info.unlp.edu.ar
[2] CONICET, Argentina

Resumen. En este trabajo presentamos un plugin de *PhoneGap* para Android que permite una comunicación genérica entre los componentes visuales de una aplicación *PhoneGap* y una clase que funciona como punto de entrada de algún modelo Java. Dicho plugin fue utilizado para la creación de una aplicación móvil basada en posicionamiento. En este trabajo se brindan detalles de nuestro plugin, como así también aspectos aprendidos durante el desarrollo y futuros usos del mismo.

Palabras Claves: PhoneGap, Aplicaciones Móviles basadas en posicionamiento, Prototipado, Computación Móvil.

1 Introducción

El avance de la tecnología ha traído en consecuencia la creciente y variada gama de dispositivos y plataformas móviles disponibles (Android, iOS, etc.). Esto genera un desafío a la hora de desarrollar aplicaciones móviles, sobre todo poder cubrir esta variada gama en corto tiempo. Varios autores (por ejemplo, [3] y [13]) coinciden en que se pueden identificar tres posibles enfoques de desarrollo: aplicaciones móviles nativas, aplicaciones web móviles o aplicaciones móviles hibridas. La elección de cada uno de ellos conlleva ventajas y desventajas [3]. Es decir, las características propias de la aplicación móvil a desarrollar determinarán cuál es el enfoque que se ajuste mejor a la solución buscada.

Las aplicaciones móviles basadas en posicionamiento, y más aún aquellas sensibles al contexto [8], poseen aspectos particulares que están asociados, en muchos casos, con los datos que proveen los sensores de los dispositivos móviles. Varios autores coinciden ([3],[11],[13]), que para lograr una solución multiplataforma y además poder acceder a los sensores, la mejor opción es optar por desarrollar aplicaciones móviles hibrida. Actualmente, existen diferentes plataformas para crear este tipo de aplicaciones (móviles hibridas). Varios autores (por ejemplo, [3],[4],[7],[9],[12] y [13]) presentan y describen comparaciones de algunas de ellas. En todos estos trabajos, los

autores coinciden en que *PhoneGap*[1] (distribución libre de *Apache Cordova*[2]) es la plataforma más popular, con más documentación y cuenta con la posibilidad de ser extendida mediante la definición de plugins. *PhoneGap* es un framework que se basa en tecnología web estándar como: HTML, JavaScript y CSS. Cabe destacar que usar esta tecnología permite realizar prototipos rápidos.

En nuestro grupo de investigación hace varios años que venimos trabajando en la temática de aplicaciones móviles sensibles al contexto ([5],[6]). En el 2014 nos invitaron a participar del evento *TEDx Diagonal73*[3], que se desarrolló en La Plata. La participación consistía en poner en práctica durante el evento una aplicación móvil basada en posicionamiento, la cual denominamos *Caminos Alternativos*. Para darle más atractivo a la aplicación la combinamos con actuaciones teatrales, las cuales estaban relacionadas con las consignas que se le presentaban al usuario desde la aplicación. En la puesta en práctica de la aplicación había actores que interpretaban escenas en diferentes lugares del edificio. Más detalles de la puesta en práctica de esta aplicación se mencionan en [1]. Es importante mencionar que esta aplicación fue desarrollada usando *PhoneGap* de una manera no tradicional. La motivación de este trabajo es precisamente esta, poder compartir el aprendizaje que experimentamos durante este desarrollo.

El objetivo del trabajo es describir un uso no tradicional de *PhoneGap*. En este trabajo presentaremos un plugin de *PhoneGap*, para Android, que permite una comunicación genérica entre los componentes visuales de una aplicación *PhoneGap* y una clase que funciona como punto de entrada de algún modelo de dominio implementado en Java. Este plugin fue desarrollado motivados por la necesidad de contar con aplicaciones puramente clientes que hicieran usos de aspectos internos del dispositivo móvil (por ejemplo, la cámara). Y, además, poder reusar modelos de dominios existentes (implementados en Java) con funcionalidad específica para aplicaciones móviles basadas en posicionamiento. La necesidad de contar con un plugin viene dada por la forma en que funciona *PhoneGap*, ya que requiere de plugins para comunicar la vista con clases Java. En nuestro caso particular, queríamos conectar las vistas con modelos de dominios que ya teníamos implementados. Y dado que estos modelos tenían diferentes puntos de entradas (clases Java específicas) queríamos encontrar una forma genérica de solución, y ahí es donde surge el plugin genérico que presentamos en este trabajo. La decisión de elegir *PhoneGap* fue aprovechar la ventaja de poder desarrollar usado las tecnologías HTML, JavaScript y CSS, sin tener que aprender aspectos particulares de la definición de las vistas, por ejemplo, en Android. En este trabajo se brindan detalles de nuestro plugin como así también aspectos aprendidos que pueden servir para otros desarrolladores.

La estructura del trabajo es la siguiente. En la Sección 2 se presentan las características de la problemática que se busca resolver. En la Sección 3 se presenta el plugin de *PhoneGap* para Android propuesto. Los resultados obtenidos y las lecciones aprendi-

[1] Página de *PhoneGap*: `http://phonegap.com` Último Acceso: 13-5-2016)
[2] Página de *Apache Cordova*: `https://cordova.apache.org` (Último Acceso: 13-5-2016)
[3] Página de *TEDx Diagonal7:* `https://www.ted.com/tedx/events/10077` (Último Acceso: 13-5-2016)

das se presentan en la Sección 4. Las conclusiones y trabajos futuros se presentan en la Sección 5.

2 Características de la Problemática a Resolver

En esta sección se describirán algunas características del contexto en el cuál trabajamos y cómo llegamos a usar *PhoneGap* de una forma no tradicional. Como mencionamos anteriormente venimos investigando temáticas afines a las aplicaciones móviles basadas en posicionamiento, en particular, distintos aspectos relacionados al diseño y modelado de las mismas. Por lo tanto, en la aplicación *Caminos Alternativos* presentada en [1] queríamos poner en práctica aspectos de modelado en los cuales veníamos trabajando, por ejemplo, la separación en capas de los contenidos de las posiciones.

Como mencionamos anteriormente *PhoneGap* se basa en HTML, JavaScript y CSS, por lo tanto, toda la lógica de control queda contenida en los JavaScript. Este es el uso tradicional de *PhoneGap*, sin embargo, esto nos generaba pasar nuestros modelos de clases implementados en Java a código JavaScript. Estos modelos de clases los venimos desarrollando como parte de un framework Java que estamos definiendo para brindar soporte en la creación de aplicaciones móviles basadas en posicionamiento [2].

Por otro lado, *PhoneGap* provee el concepto de plugin para extender funcionalidad. Estos plugins proveen una interfaz JavaScript a los componentes nativos del dispositivo (por ejemplo, para acceder a los sensores), permitiendo así que la aplicación pueda usar características nativas del dispositivo más allá de lo que está disponible para aplicaciones web puras. Cada plugin es desarrollado para una plataforma particular, por ejemplo, Android. Los plugins para Android se definen con una clase Java (que posee la funcionalidad para acceder, por ejemplo, a un sensor), un archivo JavaScript (que especifica cómo se debe llamar el plugin desde los HTMLs) y, por último, un archivo XML (el cual permite especificar la relación entre el llamado del plugin con la clase puntual a ejecutarse). Actualmente, *PhoneGap* provee plugins generales que pueden ser descargados y utilizados, por ejemplo, para acceder a la cámara del dispositivo o comunicarse con una base de datos.

Tomando este concepto de plugins, como se mencionó en la Sección 1, nos surgió la idea de usarlo para acceder a clases Java que pudieran servir como punto de entrada de nuestros modelos, y así poder reusar nuestros modelos de clases. Los plugins en *PhoneGap* están definidos para acceder a una clase puntual invocando un método puntual con sus parámetros. Esto nos generaba que por cada método que fuera punto de entrada de nuestro modelo debíamos crear un nuevo plugin, algo que claramente no escalaba. Esto nos motiva a crear una solución genérica, y la misma es presentada en este trabajo en la siguiente sección.

3 Plugin Desarrollado para PhoneGap

El plugin general que desarrollamos esta implementado para Android. Al definirlo de manera genérica, puede ser usado para comunicar cualquier JavaScript de la aplicación con alguna clase Java (que sea parte de un modelo). Esta característica nos permite que pueda ser usado con cualquier modelo de dominio. Como se mencionó en la sección anterior, cualquier plugin en *PhoneGap* para Android se debe definir con alguna clase Java, un archivo JavaScript y un archivo XML.

Para entender mejor el funcionamiento del mismo se presenta en la Figura 1 la comunicación que se establece entre la vista (HTML/JavaScript), nuestro plugin y un modelo. La característica genérica nos permite tener definida tanto la vista como el modelo acorde a los requerimientos propios de cada aplicación.

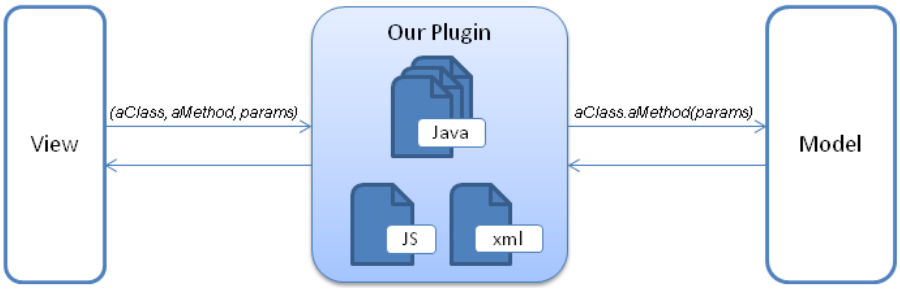

Fig. 1. Comunicación relacionada a nuestro plugin.

Se puede apreciar en la Figura 1 que nuestro plugin es invocado con: *aClass* (representa el nombre de la clase a la cual se desea invocar un método), *aMethod* (representa el nombre del método que se desea ejecutar, el mismo debe existir en el momento de invocarlo) y *params* (representa los parámetros a enviarse en el método indicado, es una colección con la forma `<tipo,valor>`). Esta información que llega a nuestro plugin es usada con reflection[4] de Java, para invocar el método de la clase con los parámetros correspondientes. La respuesta brindada por esta clase es enviada a la vista que realizó la invocación. Por ahora, el plugin está implementado para enviar y recibir información en formato JSON (*JavaScript Object Notation*), pero este aspecto está diseñado para ser extendido por cualquier otro formato.

Cabe destacar que se realizó un análisis para determinar si reflection era la mejor forma de solucionar nuestro problema. Si bien reflection puede ocasionar errores en tiempo de ejecución, que no son detectados en compilación, la flexibilidad de invocar cualquier método de cualquier clase en forma dinámica se ajustaba perfectamente a la

[4] Para más información sobre reflection se puede consultar: `https://docs.oracle.com/javase/tutorial/reflect` (Último Acceso: 13-5-2016).

problemática que teníamos que dar solución. Por ende, este fue el motivo que prevaleció en la elección del uso de reflection.

A continuación, se presenta el código relacionado a los archivos JavaScript y XML de nuestro plugin como así también la clase Java del mismo.

En la Figura 2 se puede apreciar la definición del archivo JavaScript; se puede observar que el plugin debe ser invocado usando una función que recibe una función de *success* y otra de *error* (estas funciones son invocadas para devolver el control al JavaScript que invocó nuestro plugin), como así también los parámetros ya mencionados en la Figura 1.

```
var pluginHandler = {
        pluginHandler: function(successCallback, errorCallback,theClass,theMethod,arrayParameters){
    cordova.exec(successCallback,
        errorCallback,
        "PluginHandler",
        "executePlugin",
        [{
            "theClass":theClass,
            "theMethod":theMethod,
            "arrayParameters":arrayParameters,
        }]
    );
   }
}
module.exports = pluginHandler;
```

Fig. 2. Archivo JavaScript de nuestro plugin.

En la Figura 3 se puede apreciar la clase Java que representa nuestro plugin, la cual denominamos *PluginHandler*. Esta clase extiende de *CordovaPlugin*, que es la clase que se debe extender para representar un plugin personalizado en *PhoneGap*. Se debe implementar el método `execute()` como se puede visualizar en la figura. Si bien contamos con otras clases que colaboran con la clase *PluginHandler*, a fin de entender el funcionamiento y para simplificar la descripción no entraremos en detalle.

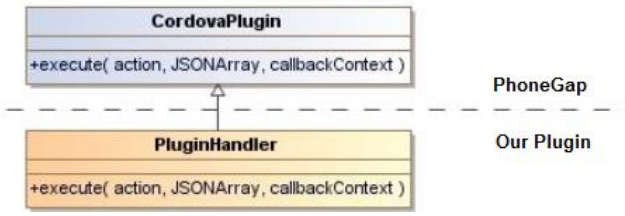

Fig. 3. Clase *PluginHandler*.

En la Figura 4 se puede apreciar el código del archivo XML de nuestro plugin, donde se establece la relación entre el archivo JavaScript y la clase *PluginHandler*.

```
<plugin xmlns="http://www.phonegap.com/ns/plugins/1.0"
    id="com.lifia.plugin.pluginHandler"
    version="0.1.0">
    <name>pluginHandler</name>
    <description> </description>
    <license>  </license>
    <keywords>pluginHandler</keywords>
    <js-module src="www/pluginHandler.js" name="pluginHandler">
    <clobbers target="pluginHandler"/>
    </js-module>
    <!--android-->
    <platform name="android">
    <config-file target="res/xml/config.xml" parent="/*">
    <feature name="PluginHandler">
        <param name="android-package"
        value="com.lifia.plugin.pluginHandler.PluginHandler" />
    </feature>
    </config-file>
    <source-file src="src/android/PluginHandler.java"
    target-dir="src/com/lifia/plugin/pluginHandler" />
    </platform>
</plugin>
```

Fig. 4. Archivo XML de nuestro plugin.

De esta manera, queda definido nuestro plugin de *PhoneGap* genérico (para Android) que puede ser usado con cualquier modelo de dominio, más allá del que probamos, en particular, para la aplicación presentada en [1].

4 Resultados Alcanzados y Lecciones Aprendidas

Para lograr que nuestro plugin interactuara con cualquier clase Java (que sea punto de entrada de un modelo), tuvimos que restringir a que los métodos invocados reciban datos simples (por ejemplo, *String*) o a lo sumo colecciones con datos simples. Como mencionamos anteriormente la variable *params* que recibe el plugin al ser invocado es una colección con la forma `<tipo,valor>`. Esto nos permite transformar el *String* que se recibe en el JSON al tipo correspondiente y así hacer la invocación del método (de la clase Java) con los parámetros con el tipo adecuado. Sin esta especificación todas las clases Java (del modelo a invocar) debían definir sus métodos recibiendo solo *String*. Con esta característica damos más flexibilidad a las clases que son invocadas usando nuestro plugin.

Una desventaja de usar plugins, es que se pierde la característica multiplataforma que propone *PhoneGap*, ya que en nuestro caso solo definimos una solución para Android. Es decir, para brindar soporte multiplataforma se debería desarrollar un plugin específico para cada una de ellas. Esto requiere incorporar conocimientos de cada una de las plataformas para las cuales se quiera desarrollar un plugin. En nuestro caso, al querer usar este desarrollo para el prototipado solo nos limitamos a desarrollarlo para Android.

Nuestro plugin se pudo probar en una aplicación específica cómo se presentó en [1]. Esto nos permitió probarlo con un modelo particular, pero además poder proyectar su uso en otro tipo de aplicaciones como son aquellas aplicaciones móviles educati-

vas[5] basadas en posicionamiento que es un área que también venimos trabajando. Cabe destacar que la aplicación presentada en [1], además de usar nuestro plugin, hace uso de un plugin provisto por *PhoneGap*, que permite la lectura de códigos de barra (llamado *BarcodeScanner*[6]). Este último plugin, abre la cámara, detecta que se está leyendo un código y devuelve el valor leído. Usamos códigos QR (Quick Response) para determinar dónde está posicionado el usuario y acorde a esto brindarle información o servicios.

Contar con el plugin para Android presentado en este trabajo nos permitirá en un futuro focalizarnos solo en la parte visual y de modelado, mientras que la comunicación entre estos estará dada por nuestro plugin. Esto nos permitirá agilizar los tiempos de prototipado, y a su vez reusar modelos que ya tenemos definidos para dominios específicos. Por otro lado, podremos trabajar con tecnologías HTML, JavaScript y CSS para definir las vistas, sin requerir aprender aspectos particulares de la definición de vistas en Android.

Varios de los prototipos que hemos desarrollamos son probados en lugares donde no hay conectividad (acceso a internet), lo cual, nos llevó a desarrollar aplicaciones clientes, en particular, en Android. Hasta el momento, las aplicaciones que venimos prototipando están haciendo uso solo del posicionamiento del usuario, en un futuro se incorporará el uso de otros sensores. Esto seguramente requerirá un análisis del plugin presentado para lograr también agilizar el uso de dichos sensores. La ventaja que tiene *PhoneGap* en este sentido es que al contar con el concepto de plugin se puede acceder a cualquier aspecto interno del dispositivo, por ejemplo, sensores. Otro aspecto a mencionar, es que las aplicaciones que hemos desarrollado no requieren intercambio de información entre usuarios; esto es un tema a explorar, y también requerirá un análisis del plugin presentado para poder incorporar esta característica.

Es recomendable antes de hacer cualquier desarrollo, evaluar cuáles de las soluciones disponibles se ajusta mejor a los requerimientos de la aplicación que se quiere desarrollar. Se recomiendan lecturas actuales como, por ejemplo, [10] para tener en cuenta otros aspectos que pueden ser críticos para algunos desarrollos como pueden ser el consumo de energía o la seguridad de la información.

6 Conclusiones y Trabajos Futuros

En este trabajo se presentó un plugin de *PhoneGap* para Android que permite una comunicación genérica entre la vista de una aplicación y una clase que funciona como

[5] Venimos trabajando en el modelado de este tipo de aplicaciones, más información se detalla en Lliteras, A.B.: "*Un enfoque de modelado de Actividades Educativas Posicionadas que contemplan elementos concretos*". Tesis de Magister en Tecnología Informática Aplicada en Educación, Facultad de Informática (UNLP), 2016 `http://sedici.unlp.edu.ar/handle/10915/50030` (Último Acceso: 13-5-2016)

[6] Más información sobre el plugin *BarcodeScanner* se detalla en: `https://build.phonegap.com/plugins/951` (Último Acceso: 13-5-2016)

punto de entrada de un modelo Java. Se describió el funcionamiento del mismo, como así también se indicó que dicho plugin se utilizó en la aplicación *Caminos Alternativos* (la puesta en práctica de la misma se presentó en [1]).

El trabajo presentado es producto de haber detectado ciertos aspectos repetitivos en el prototipado de aplicaciones móviles basadas en posicionamiento, y acorde a eso, lograr agilizar esta tarea, en pos de reducir el tiempo de desarrollo. Además, se buscaba no tener que aprender aspectos puntuales de Android, por ejemplo, cómo definir las vistas; usando *PhoneGap* podemos usar tecnologías como HTML, JavaScript y CSS. En un futuro, con la solución presentada en este trabajo, nos focalizaremos solo en la especificación de las vistas como así también del modelo de la aplicación en sí.

Todavía nos queda explorar otros aspectos como, por ejemplo, la comunicación entre usuarios. Para esto se analizará, por ejemplo, el uso de NFC para que dos usuarios puedan intercambiar información sin requerir conectividad (acorde al tipo de aplicaciones que venimos probando). Otro tema a explorar es el uso de sensores, para esto, se tomará como punto de partida los plugin que ya provee *PhoneGap* para Android. Se analizará si los mismos se necesitan generalizar y cuál es la mejor forma para combinarlos con nuestro plugin genérico. Esto nos permitirá crear aplicaciones móviles sensibles al contexto, por ejemplo, usando sensores internos del dispositivo.

Estamos trabajando para que el plugin presentado nos agilice el empaquetado de aplicaciones móviles basadas en posicionamiento definidas usando la herramienta presentada en [2]. Por ejemplo, agilizar el prototipado de aplicaciones móviles educativas basadas en posicionamiento, para poder realizar pruebas más fácilmente y con distintas características.

Referencias

1. Alconada Verzini, F.M., Tonelli, J.I., Challiol, C., Lliteras, A.B., Gordillo, S.E.: Combing Location-Aware Applications with in-situ Actors Performances. In: the 2015 Workshop on Narrative & Hypertext, pp. 27-31. ACM, New York (2015)
2. Alconada Verzini, F.M., Tonelli, J.I., Challiol, C., Lliteras, A.B., Gordillo, S.E.: Authoring Tool for Location-Aware Experiences. In: the 2015 Workshop on Narrative & Hypertext, pp. 21-25. ACM, New York (2015)
3. Brucker, A. D., Herzberg, M.: On the Static Analysis of Hybrid Mobile Apps. In: Engineering Secure Software and Systems, pp. 72-88. Springer International Publishing (2016)
4. Dalmasso, I., Datta, S. K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 9th International Wireless Communications and Mobile Computing Conference, pp. 323-328. IEEE (2013)
5. Fortier, A., Challiol, C., Fernández, J.L., Robles, S., Rossi, G., Gordillo, S.: Exploiting personal web servers for mobile context-aware applications. The Knowledge Engineering Review 29(02), 134-153 (2014)
6. Fortier, A., Rossi, G., Gordillo, S, Challiol, C.: Dealing with variability in context-aware mobile software. Journal of Systems and Software 83(6), 915-936 (2010)

7. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Web information systems and technologies, pp. 120-138. Springer-Verlag Berlin Heidelberg (2012)
8. Hong, J.Y., Suh, E.H., & Kim, S.J.: Context-aware systems: A literature review and classification. Expert Systems with Applications 36 (4), 8509-8522 (2009)
9. Jobe, W.: Native Apps vs. Mobile Web Apps. International Journal of Interactive Mobile Technologies (4), 27-32 (2013)
10. Nagappan, M., Shihab, E.: Future trends in software engineering research for mobile apps. In: 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering. (2016) http://www.se.rit.edu/~mei/publications/pdfs/Future-Trends-in-Software-Engineering-Research-for-Mobile-Apps.pdf
11. Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: 16th International Conference on Intelligence in Next Generation Networks, pp. 179-186. IEEE (2012)
12. Lim, S.: Experimental Comparison of Hybrid and Native Applications for Mobile Systems. Int. J. Multimed. Ubiquitous Eng 10(3), 1-12 (2015)
13. Xanthopoulos, S., Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In: 6th Balkan Conference in Informatics, pp. 213-220. ACM, New York (2013)