Librería Multiplataforma para Videojuegos en Dispositivos Móviles

Cristian Costa, Federico Bricker y Claudio Aciti[1]

[1] Universidad Nacional del Centro de la Provincia de Buenos Aires
Pinto 399 (7000) – Tandil – Argentina

caciti@exa.unicen.edu.ar

Abstract. This paper describes how to homogeneously use several game-engagement components included on different mobile platforms, that helps increasing gameplay on video games. Each platform provides a service API for handling leaderboards, achievements and social interaction between players. Design choices made by each platform's provider, impacts on the game's development tasks and process. Using the multi-platform language Haxe, individual libraries were built to allow easy integration of Amazon, Apple and Google's gaming services. With them an extra library that homogenizes the common services of each provider was generated, which -based on the target platform- determines the implementation to include in the final application and therefore which services to use.

Resumen. Este trabajo describe la utilización homogénea de componentes lúdicos facilitados por proveedores de distintas plataformas móviles. Tales componentes permiten incrementar la mecánica de juego de aplicaciones lúdicas en el contexto de los dispositivos móviles. Cada proveedor es el encargado de otorgar al desarrollador, una API de servicios para la manipulación de elementos como logros y tablas de clasificación. Las decisiones políticas y tecnológicas tomadas por cada uno impacta en la tarea del desarrollador de aplicaciones móviles. A partir del lenguaje multiplataforma Haxe, se construyeron librerías individuales que permitan usar los servicios de las implementaciones de Amazon, Apple y Google. Con ellas se generó una librería extra que homogeniza los servicios comunes de cada proveedor y, a partir de la plataforma objetivo, determina cual implementación del proveedor incluir en la aplicación final y por ende, cuáles servicios utilizar.

Keywords: Componentes lúdicos, API homogénea, multiplataforma, Haxe, OpenFL, videojuegos, dispositivos móviles.

1 Introducción

En el último año se observó un crecimiento en el mercado de los teléfonos inteligentes con 1200 millones de unidades vendidas. Actualmente, estos dispositivos

cuentan con grandes capacidades de procesamiento, almacenamiento y conectividad, siendo su plataforma móvil, la encargada de administrar tales recursos. Estas plataformas son proporcionadas por proveedores tales como: Apple, Google, Windows, etc. Aquí, cada uno determina el lenguaje de programación en el que está concebida su plataforma, restringiendo el desarrollo y funcionamiento de aplicaciones. En cada una de estas plataformas, se encuentra un gestor de aplicaciones que permite a los usuarios instalar aplicaciones de acuerdo a sus necesidades [1].

No hay restricciones sobre quien puede crear aplicaciones, pudiendo ser: el proveedor de la plataforma, el proveedor del dispositivo, una empresa de desarrollo móvil o un desarrollador particular. Independientemente de quien sea el autor, las aplicaciones pueden seguir estrategias de monetización [2] para generar ganancias. Para insertar aplicaciones al gestor, el proveedor de la plataforma móvil ofrece un portal web llamado consola de desarrollador.

A partir de estas condiciones, es válido pensar que cualquier desarrollador desee cubrir la mayor cantidad de plataformas con la misma aplicación. De esta forma abarcar una mayor cantidad de público y, a partir de las estrategias de monetización, mayor ganancia. Pero, como cada proveedor utiliza su lenguaje de programación predilecto para concebir su plataforma móvil, el desarrollador está obligado a generar la misma aplicación en cada lenguaje [3].

Ningún tipo de aplicación está exento a la problemática de múltiples plataformas, ni siquiera los videojuegos. Adicionalmente, las aplicaciones de este estilo conviven con el desafío de cautivar o "enganchar" a los usuarios que las utilicen. En los videojuegos, el número de participantes a atraer está determinado por las mecánicas del juego [4]. Estos son los componentes base del juego: reglas o acciones que puede realizar un jugador en el mismo. Para crear mecánicas de juego existe un conjunto de herramientas o componentes lúdicos tales como: logros, niveles, tablas de clasificación, sistemas de puntajes, etc..

En el ámbito de los dispositivos móviles, los proveedores permiten la generación de logros y tablas de clasificación desde la consola de desarrollador. Estos mismos buscan, mediante la difusión del progreso de cada jugador a los restantes, conformar un estado de competitividad entre los participantes. Para manipular estos componentes (por ejemplo: guardar un puntaje en una tabla dada) cada proveedor ofrece una API de servicios para utilizar desde el código fuente de la aplicación. Ahora bien, en caso de que el desarrollador desee simultáneamente:

- Cubrir con el mismo videojuego varias plataformas.
- Utilizar los componentes lúdicos de los proveedores de cada plataforma.

Se encontrará con desventajas impuestas por las API's que manipulen tales componentes:

- El proceso de integración de cada API es distinto para con el mismo proyecto.
- Existe una falta de estandarización entre API's. Por ejemplo, ciertos servicios que otorgue la implementación de un proveedor, pueden no estar en la de otro.
- Cada API se encuentra implementada en el mismo lenguaje de programación que la plataforma móvil del proveedor. Por ende, la lógica del videojuego también debe ser desarrollada en tal lenguaje.

1.1 Motivación

La situación que motiva a afrontar un problema de esta índole se hace presente en el contexto de una empresa de desarrollo de videojuegos para dispositivos móviles. Aquí, sus objetivos constan en:

1. Comercializar sus productos en las distintas tiendas.
2. Abarcar la mayor cantidad de usuarios a partir del nivel de "enganche" que generen sus aplicaciones.

Para lograr el primer objetivo la empresa utiliza un lenguaje multiplataforma, mientras que para el segundo, la misma hace uso de componentes lúdicos propios y facilitados por los proveedores. Para manipular a estos últimos, es necesario contar con la API de cada proveedor e integrarla al código fuente. Esta situación genera conflictos con el lenguaje multiplataforma:

- Incompatibilidad entre los lenguajes de programación utilizados. Esto dificulta la utilización directa de los servicios desde el código fuente de la aplicación.
- Cada API está restringida a la plataforma móvil donde funciona correctamente, anulando la capacidad multiplataforma del lenguaje para crear la aplicación final (condicionando la salida para la cual debe generarse la aplicación final).

Todas estos condicionamientos se traducen en incrementos de tiempo y esfuerzo invertido que serían deseables evitar.

1.2 Objetivos

Se propone la creación de una API que permita utilizar, de forma homogénea, los servicios de las implementaciones de cada proveedor. Así mismo se establecerán como objetivos parciales la generación de módulos específicos para cada proveedor.

1.3 Alcances y Limitaciones

La solución se realiza en el lenguaje multiplataforma Haxe utilizando las implementaciones de los proveedores Amazon, Apple y Google. Las limitaciones existentes son causadas por la falta de estandarización a partir de las políticas adoptadas por cada proveedor para materializar su API de servicios.

1.4 Organización del Trabajo

En este primer capítulo se introduce la problemática abordada. En el segundo, se realiza una reseña del estado del arte. En el tercer capítulo se establecen los requerimientos y el punto de partida del proyecto. En el capítulo cuatro se describe la solución y se narran las estrategias utilizadas para su producción. Posteriormente, en el quinto capítulo se muestran análisis que permiten conocer las ventajas que ofrece el uso de la solución. En el sexto se exponen las conclusiones obtenidas de este trabajo. En el séptimo capítulo son enumerados una serie de trabajos futuros propuestos como

continuación de este estudio. Finalmente se listan las referencias bibliográficas utilizadas.

2 Estado del Arte

2.1 Componentes Lúdicos de los Proveedores

Actualmente Apple, Amazon y Google ofrecen sus servicios de juegos [5] que permiten a los desarrolladores el agregado de componentes lúdicos como son los logros y las tablas de clasificación. Así mismo, estos proveedores ofrecen sus API's de servicios para manipular los atributos de cada elemento. Por otro lado, la ejecución de los servicios de cualquiera de las API's se realiza de manera asíncrona. En términos de diferencias conceptuales, estas se evidencian en los Logros: Google utiliza la noción de pasos, mientras que Amazon y Apple utilizan la noción de porcentaje de avance. Por último, en el aspecto tecnológico, Amazon y Google ofrecen el servicio de sincronización de partidas entre distintos dispositivos dentro de su implementación, mientras que Apple lo ofrece con otra API dedicada exclusivamente a dicha tarea.

2.2 Haxe/OpenFL

Haxe [6] es un lenguaje de programación de código abierto multiplataforma que permite generar código para lenguajes como JavaScript, Flash, PHP, C++, C# y Java. La ideología que sigue es, permitir al desarrollador seleccionar la plataforma objetivo más acorde al problema a resolver aumentando la flexibilidad en el desarrollo a la vez que se maximiza el retorno de una inversión. Una de las características más importantes de Haxe es que permite, a partir de macros de compilación condicional, ejecutar código especifico para una determinada plataforma.

OpenFL [7] es un framework de código abierto multiplataforma que, haciendo uso de Haxe permite la generación de aplicaciones o videojuegos ejecutables, pudiendo correr sobre plataformas como: Windows, GNU/Linux, navegadores web con soporte de HTML5 o Flash, teléfonos inteligentes y tablets con Android o iOS. OpenFL permite materializar librerías a partir de implementaciones no existentes en Haxe ni en dicho framework, vinculando y haciendo uso de servicios provistos por terceros.

3 Descripción del Problema

A partir de las implementaciones de los proveedores en conjunción con Haxe/OpenFL se pretende generar una API que permita hacer uso de los servicios de cada uno de las implementaciones de los proveedores que manipulan los componentes lúdicos de forma homogénea. Para ello, la solución debe satisfacer los siguientes requerimientos:

- Pertenecer al lenguaje Haxe.

- Facilitar la actividad de integración al desarrollador. Transitivamente, este aspecto reducirá el tiempo invertido en realizar tal actividad.
- A partir de la plataforma objetivo para la cual sea generada la aplicación final, la solución debe determinar cual implementación del proveedor incluir en la misma.
- Por último, la solución, al tratarse de una API, debe ofrecer a los desarrolladores de videojuegos en Haxe los servicios presentados en la Tabla 1.

Tabla 1. Requerimientos solicitados.

Componente	Descripción del servicio
Tabla de clasificación	Visualizar una tabla en particular.
	Visualizar todas las tablas del juego.
	Almacenar un puntaje en una tabla dada.
	Obtener el puntaje de una tabla dada.
Logro	Visualizar todos los logros del juego.
	Desbloquear un logro en particular.
	Establecer el progreso de un logro dado.
	Revelar un logro en particular.
	Incrementar el progreso de un logro dado.
	Obtener el progreso de un logro dado.
	Obtener el estado de un logro dado.
Almacenamiento	Almacenar partida.
	Recuperar partida.

Por otro lado, se debe realizar una "línea base" del trabajo, presentada en la Tabla 2. Esto se debe a que inicialmente, la comunidad de Haxe/OpenFL ya contaba con dos complementos individuales de los proveedores de Apple y Google. A partir de los existentes es necesario completarlos y generar los complementos faltantes (Amazon y API Homogénea).

Tabla 2. Línea base del proyecto.

Servicio	Apple	Google	Amazon	API Homogénea
Visualizar tabla.	Si	Si	Posible	Posible
Visualizar tablas.	No	Si	Posible	Posible
Almacenar puntaje.	Si	Si	Posible	Posible
Obtener puntaje.	Posible	Posible	Posible	Posible
Visualizar logros.	Si	Si	Posible	Posible
Desbloquear logro.	No	Si	No	Posible
Establecer progreso.	Si	Si	Posible	Posible
Revelar logro.	No	Si	No	Posible
Incrementar progreso.	No	Si	No	Posible
Obtener progreso.	Posible	Posible	Posible	Posible
Obtener estado.	Posible	Posible	Posible	Posible
Almacenar partida.	No	Si	Posible	Posible

Recuperar partida.	No	Si	Posible	Posible

Las referencias para cada valor que puede contener una celda se aclara a continuación:

- Las celdas que indican "Si", denotan que la funcionalidad se encuentra implementada.
- Las celdas con el estado "Posible", indican que la funcionalidad no se encuentra implementada pero, a partir de la documentación de la API del proveedor, es posible implementarla.
- Las celdas con "No" establecen que no es posible generar la funcionalidad ya que la implementación del proveedor no la ofrece.

4 Solución Propuesta

Se propone brindar al desarrollador, una API que permite utilizar servicios para videojuegos de diversos proveedores, de forma homogénea y en lenguaje Haxe. En la Figura 1 se presenta en alto nivel el rol que cumple la solución.

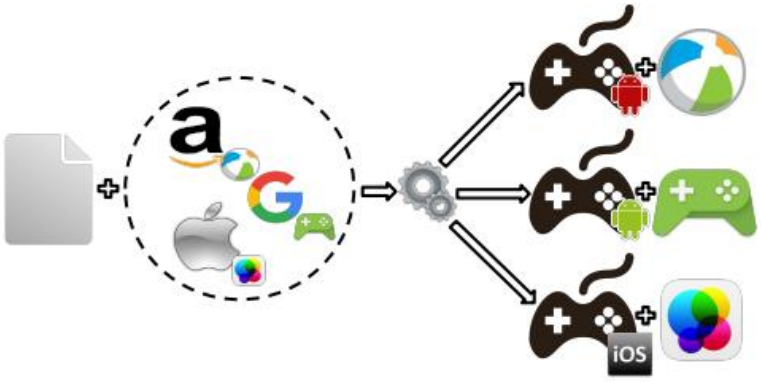

Figura 1. Esquema de la solución en alto nivel.

Dado que cada proveedor utiliza aspectos tecnológicos diferentes para concebir su API, la solución está compuesta por módulos específicos que permitan la utilización de cada implementación desde el lenguaje Haxe (estos módulos se llamarán Módulos Puente).

Esta API, entiende cómo utilizar los componentes lúdicos disponibles en cada proveedor; y dependiendo de la plataforma para la cual sea compilado el videojuego, utiliza la implementación correspondiente sin que esto requiera adaptación alguna por parte del desarrollador. Así mismo facilita el proceso de integración para cualquier proyecto reduciendo tiempo y esfuerzo insumido para tal actividad.

Para los casos en los que el juego sea generado hacia una plataforma para la cual no hay un módulo puente integrado con la solución, la API resulta inocua, permitiendo la compilación y ejecución del juego sin errores ni condicionamientos.

En la Figura 2 se muestra a más bajo nivel la predisposición de módulos puentes que conforman a la API Homogénea solución. Cada uno de los módulos fue concebido a partir de la herramienta de creación de extensiones de Haxe/OpenFL. Los módulos puentes utilizan las implementaciones de los proveedores en conjunción con la capacidad de vincular e invocar servicios desde el lenguaje Haxe hacia el lenguaje de programación utilizado en la implementación del proveedor. Por último, la API Homogénea, hace uso de la compilación condicional de Haxe para agrupar en un único servicio los servicios comunes de cada módulo puente. A lo largo del desarrollo de cada módulo puente, se buscó ofrecer la funcionalidad lo más completa posible que brinda cada proveedor como así también mantener un enfoque que respete los estándares de desarrollo en Haxe/OpenFL.

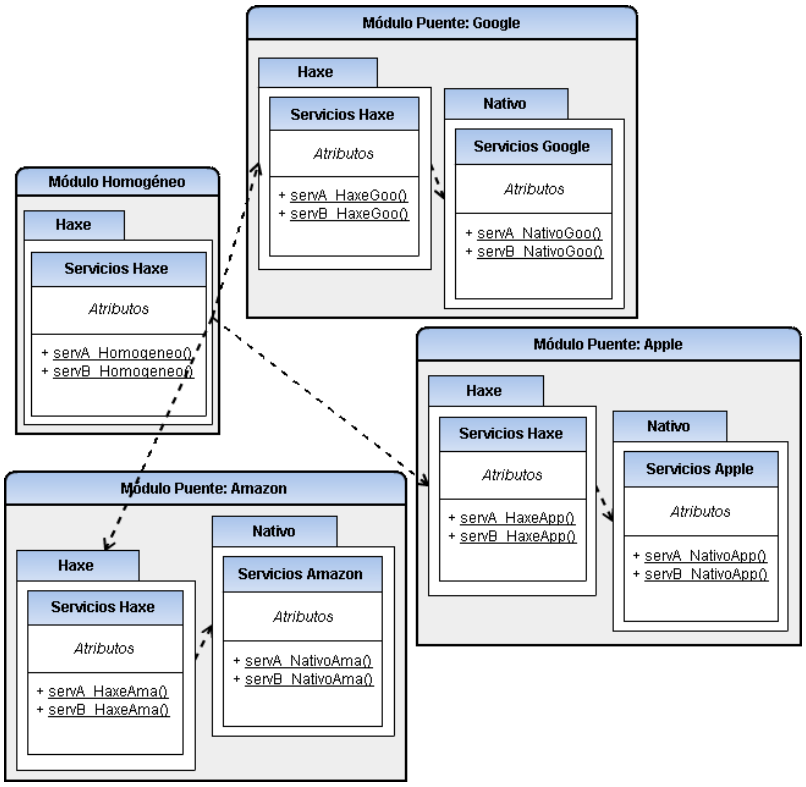

Figura 2. Predisposición de módulos puentes que conforman a la solución.

Desde el punto de vista del desarrollador, la solución es relativamente simple de utilizar, aunque se destaca la utilización de un sistema de retrollamadas (callbacks) que aumenta la complejidad de uso para todas las funciones de las cuales se espera una respuesta o valor de retorno. Esto se debe a que los servicios de los proveedores trabajan de manera asíncrona, con lo cual se separa la forma de utilizar las funciones que no entregan información al desarrollador (almacenar un puntaje, mostrar todas las

tablas, mostrar logros, etc.); de aquellas que sí lo hacen (obtener el puntaje de una tabla, obtener el progreso de un logro, etc.). Para el segundo tipo de funciones será necesario procesar la respuesta una vez que esté disponible. Las retrollamadas establecen que: en vez de esperar el resultado de una función (blocking), o de consultar esporádicamente por el estado de retorno (polling), el desarrollador debe informar qué función debe ser invocada cuando se obtenga el resultado.

5 Pruebas de Campo

Para hacer uso de la solución es necesario incluir la misma a cada proyecto Haxe. Esto se realiza utilizando dos directivas, una en el archivo de definición de proyecto y otra en el código fuente de la aplicación. Una vez hecho esto, se está en condiciones de poder realizar invocaciones de los servicios de la solución.

Para el análisis, se utilizaron los proyectos de la empresa como modelos de pruebas. El estudio consistió en comparar cada aplicación consigo misma, con la diferencia de que:

- El primer caso, supone la integración de las implementaciones de los tres proveedores sin la intervención de la solución.
- El segundo, supone el uso de la API homogénea.

De la tarea de integración se contemplaron las actividades que influyen en aspectos como: Tiempo insumido y Cantidad de sentencias necesarias a agregar en cada proyecto. En la Tabla 3 se presentan las condiciones a considerar de ambos casos.

Tabla 3. Condiciones previas al estudio de ambos casos.

Sin API Homogénea	Con API Homogénea
• Módulos existentes e incompletos (Apple y Google). • Modulo inexistente (Amazon). • Interfaz homogénea inexistente.	• Módulos de los proveedores completos y validados. • Interfaz homogénea existente.

El primer caso se corresponde a la "línea base" establecida en la sección 3:

- Por un lado, los módulos puente de Apple y Google ya existen, pero ambos se encuentran incompletos.
- Otro aspecto a tener en cuenta es que: el único módulo que debe ser creado de cero es el de Amazon.
- Por último, en el código fuente de cada proyecto se debe encontrar la misma lógica que presenta la API Homogénea, agrupando los servicios comunes de cada módulo puente en un único servicio a partir de la compilación condicional de Haxe.

De estas condiciones se concluye que: los puntos 1 y 2 (creación y completitud de cada módulo) se deben realizar una única vez para el primer proyecto, mientras que el punto 3 debe repetirse en todos los proyectos. Por último, se deben tener en cuenta las

sesiones de pruebas a los que debe someterse cada servicio de juegos y eventuales correcciones que deban realizarse en caso que se presenten errores. Estos dos últimos aspectos insumirán tiempo extra a la tarea de integración.

En cambio, con el segundo caso ya nos encontramos con una API que presenta una interfaz homogénea y módulos puentes completados, verificados y validados. Con lo cual es intuitivo pensar que la cantidad de sentencias necesarias para utilizarla y el tiempo de integración disminuyan con relación al primer caso.

Entonces, se tomó un proyecto de la empresa y, a partir de ambos supuestos, se estimaron los siguientes datos mostrados en las Tablas 4 y 5:

Tabla 4. Estimación del caso Sin API Homogénea.

	#Días inclusión	#Sentencias
Proyecto	34	320

Tabla 5. Estimación del caso Con API Homogénea.

	#Días inclusión	#Sentencias
Proyecto	2	2

El valor de 34 días, es la cantidad de tiempo aproximado que demandó generar el modulo de Amazon, adicionado al tiempo necesario para completar los módulos de Google y Apple con sus correspondientes sesiones de pruebas para cada uno de los tres.

Las 320 sentencias necesarias para agrupar los servicios comunes de cada módulo son la cantidad de sentencias que conforman a la lógica de la API Homogénea. La diferencia radica en que, para el primer caso, la lógica forma parte del código fuente de cada videojuego, mientras que para el segundo caso dicha lógica es ajena.

En conclusión, hacer uso de la API Homogénea para el desarrollo de videojuegos en el lenguaje Haxe significa un ahorro en el tiempo de lanzamiento del mismo a un mes.

En aspectos cualitativos se destaca:

- La mantenibilidad de la API Homogénea a partir de su diseño centralizado es una ventaja frente al caso donde no se utiliza la misma, reduciendo el costo en términos de tiempo y esfuerzo. Realizar los cambios sobre un único lugar impacta sobre todas las aplicaciones que lo utilicen.
- La escalabilidad en el caso de que surjan nuevos servicios de juegos, insumiendo menor esfuerzo crear el módulo puente e integrarlo a la API Homogénea que integrarlo en cada proyecto.

6 Conclusiones

Se implementó una librería que permite utilizar, de forma homogénea, servicios para manipular los componentes lúdicos que ofrecen los proveedores desde el lenguaje

Haxe. La solución es capaz de, a partir de un único código fuente y la plataforma objetivo deseada, determinar cual implementación del proveedor incluir en la aplicación final sin perjudicar al lenguaje multiplataforma. En base a lo observado durante las pruebas se apreció el nivel de impacto y las ventajas que ofrece utilizar este complemento en relación a no hacerlo. Adicionalmente, la solución posee flexibilidad ante eventuales expansiones, permitiendo agregar nuevos servicios para los proveedores considerados o permitiendo agregar implementaciones de proveedores no contemplados en el presente trabajo.

7 Trabajos Futuros

A partir de lo realizado, se proponen como trabajos futuros las siguientes consideraciones:

- Expandir la API Homogénea tanto en número de servicios como en número de proveedores.
- Otorgar la posibilidad a los jugadores de plasmar su progreso desde el mismo videojuego en los servicios de juegos de cada proveedor.
- Agregar el soporte necesario para la sincronización de partidas en dispositivos que contengan la plataforma móvil de Apple.
- Permitir que la API Homogénea mantenga el estado de ciertos atributos de interés para los componentes lúdicos utilizados. Almacenar ciertos valores en estructuras pertenecientes a la solución evitarían utilizar los servicios asíncronos para consultarlos.

8 Referencias

1. Jansen, S., Bloemendal E.: Defining App Stores: The Role of Curated Marketplaces in Software Ecosystems. Springer Berlin Heidelberg (2013).
2. Cravens, A.: A demographic and business model analysis of today's app developer. Application Developers Alliance (2012).
3. Joorabchi, M.E., Mesbah, A., Kruchten, P.: Real Challenges in Mobile App Development. IEEE International Symposium on Empirical Software Engineering and Measurement (2013).
4. Zichermann, G., Cunningham C.: Gamification by Design Implementing Game Mechanics in Web and Mobile Apps. O'Reilly Media, Inc. (2011).
5. de Prato, G., Feijoo, C., Simon, J-P.: Innovations in the Video Game Industry: Changing Global Markets. Digiworld Economic Journal (2014).
6. Ponticelli, F., McColl-Sylvester, L.: Professional haXe and Neko. Wiley Publishing, Inc. (2008).
7. Vallejo Fernández, D., González Morcillo, C., Frutos Talavera, D.: Videojuegos Multiplataforma con OpenFL. Edlibrix. (2014).