Desarrollo multiplataforma de Aplicaciones Móviles combinadas con el uso de Beacons

Franco M. Borrelli[1], Pedro Brost[1], Cecilia Challiol[1,2], Diego H. Orellano[1], Facundo I. Mendibru[1], Matías A. Santoleri[1] and Federico M. Alconada Verzini[1]

[1] UNLP, Facultad de Informática, LIFIA. Calle 50 y 120, La Plata, Argentina.
[2] CONICET, Argentina
{fborrelli, pbrost, ceciliac, dorellano, fmendiburu, msantoleri, falconada}@lifia.info.unlp.edu.ar

Resumen. En este trabajo se presenta la exploración de dos APIs de *beacons* existentes, una para *PhoneGap* y otra para *React Native*. Para esta exploración, se desarrollaron dos aplicaciones móviles, una en *PhoneGap* y otra en *React Native*; las cuales usan estas APIs para detectar la proximidad a los *beacons*, al entrar en la proximidad de un *beacon*, se brinda en pantalla la información del mismo. Se detallan distintas características de estas aplicaciones desarrolladas, haciendo hincapié en el análisis comparativo de estos desarrollos respecto a lo que provee cada API. Además, el trabajo presenta un espacio de discusión para analizar las lecciones aprendidas en la exploración realizada, en pos de contribuir a la temática relacionada al desarrollo de tipo de aplicaciones.

Palabras Claves: Mecanismos de Sensado de Posicionamiento Indoor; Desarrollo Multiplataforma de Aplicaciones Móviles; Beacons; PhoneGap; React Native.

1 Introducción

Los avances tecnológicos de los últimos años han permitido que emerjan diferentes mecanismos de sensado de posicionamiento para espacios indoor [1], [2]. Cada uno de estos tiene diferentes niveles de precisión [2]. Es decir, brindan la posición actual del usuario con mayor o menor nivel de exactitud respecto de la posición real del mismo. En algunos casos, estos mecanismos se pueden combinan para estimar con mayor precisión la posición actual del usuario [1]. Actualmente, uno de los mecanismos de sensado que se perfila como confiables y eficientes, acorde a [1], son los *beacons*; los cuales emiten datos por radiofrecuencia utilizando tecnología BLE (*Bluetooth Low Energy*) [1]. Estos se caracterizan por tener bajo costo y además su uso conlleva bajo consumo de energía.

Las aplicaciones móviles que usan mecanismos de sensado de posicionamiento para brindar servicios o información acorde a la posición actual del usuario, se denominan *sensibles a la posición*, y son un subconjunto de las aplicaciones sensibles al contexto [3], [4]. Este tipo de aplicaciones todavía no cuenta con una solución unificada, es decir, su abordaje sigue siendo un tema abierto de investigación [3]. Más

aún, en [4] se plantea la importancia de crear aplicaciones sensibles al contexto realmente usables, ya que muchas de las aplicaciones existentes no están focalizadas en las necesidades reales de los usuarios. La mayoría de los framewoks o modelos de contexto consideran la adquisición del valor sensado como algo que "*llega de alguna manera*" [3], pero no profundizan en cómo dicho valor se sensa. Por otro lado, los mecanismos de sensado de posicionamiento son analizados desde la perspectiva de cómo funcionan (como se detalla en [1] y [2]), sin hacer hincapié en qué servicios se brindan con ese valor sensado [1]. En [2] se presenta un ranking de cuáles son los mecanismos de sensado de posicionamiento más apropiados para distintos tipos de aplicaciones. Por ejemplo, los *beacons* se clasifican como el mejor mecanismo para aplicaciones de navegación indoor y tracking de usuarios. Sin embargo, en [2] no se detalla cómo se deberían desarrollar estas aplicaciones acorde a cada tipo de sensado.

El desarrollo de aplicaciones móviles, como se menciona en [5] y [6], también viene evolucionando en los últimos años debido a la constante demanda del mercado. Para cubrir la mayor cantidad de dispositivos móviles, y llegar así a más usuarios, han surgido los desarrollos multiplataforma; los cuales pueden ser, por ejemplo, híbridos (como *PhoneGap* [7]) o interpretados (como *React Native* [8]). Cada uno de estos tiene un flujo de ejecución distinto generando así ventajas y desventajas [6]. Los desarrollos multiplataforma permiten acceder a características propias de los dispositivos móviles como en un desarrollo nativo, por ejemplo, acceder a los sensores de los dispositivos. En [5] se plantea que el ciclo de vida de las aplicaciones creadas con desarrollos multiplataforma es un tema poco explorado, en particular, cómo abordar el mantenimiento de las mismas. Esto podría complejizarse aún más, si este tipo de aplicaciones se combina con mecanismos de sensado de posicionamiento.

El uso de *beacons* se viene explorando en diferentes dominios tales como publicidad [9], sistemas de salud [10], guías móviles dentro de museos [11], etc. Esta exploración está en una etapa inicial, principalmente se viene analizando para cada dominio cómo se comporta este mecanismo de sensado o qué servicios se pueden brindar a partir del mismo. Este análisis se realiza generalmente mediante aplicaciones móviles ad-hoc prototípicas (por ejemplo, [9], [10] y [11]). Sin embargo, hay poco explorado respecto a la comparación de diferentes tipos de desarrollos multiplataforma para aplicaciones móviles sensibles a la posición que usan *beacons*, esto es la principal motivación del presente trabajo.

El objetivo principal del trabajo es presentar la exploración relacionada al uso de dos APIs de *beacons* existentes, una para *PhoneGap* (*híbrido*) [12] y otra para *React Native* (*interpretado*) [13]. Para esta exploración, se desarrollaron dos aplicaciones móviles, una en *PhoneGap* y otra en *React Native*; estas aplicaciones usan las APIs mencionadas para detectar la proximidad a los *beacons*. De esta forma, al entrar en la proximidad de un *beacon*, se brinda en pantalla la información del mismo. Se realiza un análisis comparativo de estos desarrollos respecto a lo que provee cada API. Además, el trabajo presenta un espacio de discusión para analizar las lecciones aprendidas en la exploración realizada, en pos de contribuir a la temática de contar con una solución unificada para abordar el desarrollo de las aplicaciones móviles sensibles al contexto.

El trabajo se estructura de la siguiente manera. En la Sección 2 se describe el estado del arte acorde al objetivo del trabajo. En la Sección 3 se presenta la exploración realizada sobre el uso de dos APIs de *beacons* existentes. Algunas

características relacionadas a la temática son discutidas en la Sección 4. Las conclusiones y los trabajos futuros son presentados en la Sección 5.

2 Estado del Arte

En el último año han surgidos distintos tipos de *beacons* poco documentados en la bibliografía académica, por esta razón estos se detallan ampliamente. El desarrollo de aplicaciones móviles se describirá brevemente, ya que esta temática está más explorada y analizada en la bibliografía existente.

2.1 Beacons

Un *beacon* es un dispositivo que emite datos por radiofrecuencia utilizando tecnología BLE (*Bluetooth Low Energy*) [1]. Existen distintos tipos de *beacons*, los tres más conocidos se describen a continuación:

- *Beacons de Proximidad*, fueron los primeros en surgir y son al día de hoy los que más se utilizan. Estos plantean "*etiquetar*" ciertas áreas o *regiones* específicas, las cuales pueden resultar de interés para reaccionar ante la proximidad a las mismas. Estas áreas son circulares y el *beacon* se ubica en el centro de estas. Una aplicación móvil que usa estos *beacons* podría detectar cuando el usuario ingresa a unas de estas áreas, y brindarle alguna información o servicio. Por ejemplo, un área de interés podría ser un local de un shopping [9]; cuando el usuario ingresa a esta área (porque está en la proximidad de un *beacon*), se le brindan ofertas de ese local. En la Figura 1.a se puede apreciar a un usuario dentro de la región de un *beacon*.
- *Beacons de Posicionamiento* (*Location UWB Beacons*), han sido lanzados por la empresa *Estimote* [14]; permiten obtener la posición del usuario en un formato (x,y) en relación a una zona de sensado. Estos *beacons* requieren primero establecer la zona de sensado, para lo cual se podría usar una aplicación brindada por *Estimote*, la cual crea un mapa de la zona automáticamente, para luego poder posicionar al usuario sobre el mismo. En una zona al menos debe haber cuatro *beacons* de este estilo. Cabe mencionar que estos *beacons* también pueden funcionar como de *proximidad* en el caso de no usar el concepto de zona de sensado. En [10] se utilizan estos *beacons* para saber la posición de una enfermera en un centro asistencial para adultos mayores. En la Figura 1.b se puede observar un usuario dentro de la zona de sensado delimitada por cuatro *beacons*.
- *Beacons Stickers (o Nearables)*, estos también han sido creados por la empresa *Estimote* [14]; sirven para posicionar objetos que pueden moverse en una zona de sensado, es decir, se combinan con los *beacons de posicionamiento* mencionados anteriormente. Un aplicación móvil podría reaccionar al detectar el movimiento o cercanía de este tipo de *beacon*, además de poder determinar en qué posición (x,y) se encuentran los mismos. En [10] se utilizan estos *beacons* para saber la posición de una paciente con capacidad de movilidad reducida en un centro asistencial para adultos mayores. En la Figura 1.c se puede observar un *sticker* dentro de la zona de sensado delimitada por cuatro *beacons de posicionamiento*.

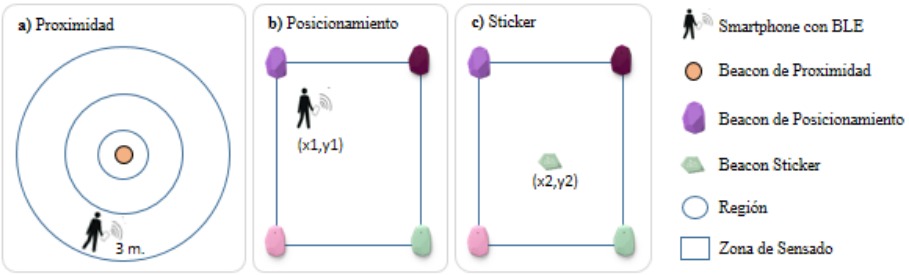

Fig. 1. Ejemplos de tipos de *beacons*. 1.a) *beacon de proximidad*. 1.b) *beacons de posicionamiento*. 1.c) *beacon sticker delimitado por beacons de posicionamiento*.

Cabe mencionar que los *beacons de proximidad* pueden combinarse bajo alguna heurística para funcionar en forma conjunta como los *beacons de posicionamiento*, es decir, brindar una posición (x,y). Esto requiere desarrollar el algoritmo de análisis de señales, algo que es trasparente cuando se usan *beacons de posicionamiento*. En [15] se explora el uso de seis *beacons de proximidad* delimitando una área cuadrada para analizar la precisión de la posición en términos de (x,y); los autores identifican un margen de error de entre treinta y setenta centímetros en las pruebas realizadas. Por otro lado, en [11] se exploran cómo se comportan nueve *beacons de proximidad* formando un área circular, una línea o un cuadrado. Algo similar ocurre con la exploración realizada en [16] donde se prueban distintas distribuciones de los *beacons*. Los autores mencionan en [11] y [16] que en las áreas donde hay muchos *beacons* para determinar la posición se logra mayor precisión, pero esto requiere colocar en pocos metros muchos *beacons*, elevando así el costo implicado.

2.1.1 Protocolos asociados a los distintos tipos de beacons

Cada uno de los tres tipos de *beacons* mencionados anteriormente (*proximidad*, *posicionamiento* y *sticker*) tiene protocolos para transmitir datos. Para los *beacons de proximidad* existen distintos protocolos que pueden ser usados por las aplicaciones móviles que hacen uso de estos *beacons*. Por ejemplo, *iBeacon* [17] es un protocolo desarrollado por *Apple* (en el año 2013). Si bien este protocolo brinda soporte para los dispositivos móviles con iOS, existe una versión open source compatible, denominada *AltBeacon* [18], que da soporte a otras plataformas como *Android*. Otros protocolos son *Estimote Monitoring* [19] implementado por *Estimote* y *Eddystone* [20] desarrollado por *Google*. Todos estos protocolos de proximidad definen mínimamente dos eventos: entrada y salida a una región. Estos eventos son abordados por los diferentes protocolos bajo distintos nombres: *Monitoring* tanto en *iBeacon*, *AltBeacon* y *Estimote*, mientras que *Eddystone* lo llama *UID* (y *EID* a su versión segura, que garantiza que solo las aplicaciones y servicios autorizados pueden comunicarse con los *beacons*).

En la Tabla 1 se puede apreciar el tiempo de demora (*delay*) que tiene cada uno de los protocolos mencionados para detectar la entrada y salida a las regiones. Se puede observar en dicha tabla que todos los protocolos permiten funcionamiento en

background. En la misma también se analizan otras características, las cuales se describen a continuación. La *Configuración de Rango de Acción* se refiere a la capacidad de poder definir distintos rangos de proximidad sobre los cuales actuar. En relación a esto se puede apreciar que solo el protocolo *Estimote Monitoring* brinda soporte a esta característica. La *Cantidad Máxima de Regiones*, define el número máximo de regiones detectables simultáneamente; en el caso de *iBeacon*/*AltBeacon* esto se limita a veinte regiones simultáneas, mientras que en *Estimote Monitoring* es ilimitado. El *Efecto Rebote* es ese "*parpadeo*" que se puede visualizar en una aplicación al estar cerca de un determinado *beacon*; esto sucede porque se acciona falsamente un evento de salida y luego de unos pocos segundos se acciona nuevamente un evento de entrada. Este tipo de comportamiento puede afectar la experiencia del usuario que usa la aplicación. En la Tabla 1 se puede apreciar que solo el protocolo *Estimote Monitoring* brinda soporte para evitar este efecto.

Tabla 1. Comparación entre los distintos protocolos de proximidad.

	iBeacon/AltBeacon	Estimote Monitoring	Eddystone UID/EID
Tiempo de demora del evento de entrada	< 3 segundos	< 3 segundos	< 3 segundos
Tiempo de demora de evento de salida	20-30 segundos	< 5 segundos	20-30 segundos
Funcionamiento en background	si	si	si
Configuración de Rango de Acción	no	si	no
Cantidad Máxima de Regiones	20	ilimitado	-
Efecto Rebote ("*parpadeo*")	si	no	si

Además, el protocolo *iBeacon*/*AltBeacon* provee la funcionalidad de *ranging*, la cual para una determinada región retorna una lista con los datos básicos de los *beacons* detectados, así como también información adicional como la proximidad estimada entre el dispositivo y cada uno de los *beacons*. Esta información podría ser útil para poder realizar heurísticas para que estos *beacons* se comporten como de *posicionamiento* (como se realiza, por ejemplo, en [11], [15] y [16]). *Eddystone* por su parte provee además el protocolo *Eddystone URL*, el cual resulta útil para transmitir direcciones URL. Supongamos que un cine tiene *beacons* cerca de los banners de presentación de las películas en cartelera; cuando el usuario está cerca de uno de los *beacons*, le llega una notificación al celular con el link para ver el tráiler de la película asociada al *beacon*.

Para los *beacons de posicionamiento*, *Estimote* provee el protocolo denominado *Estimote Location Protocol* [21], el cual permite recibir información respecto a la posición (x,y) de un dispositivo móvil o sticker, facilitando que la misma sea mostrada en un mapa. Este protocolo está pensado para agilizar la construcción de aplicaciones móviles basadas en posicionamiento indoor. Para este tipo de *beacons* se

pueden recibir, mediante otro protocolo, los datos telemétricos, que incluyen, por ejemplo, temperatura, luminosidad, etc.

En el caso de los *beacons Stickers*, *Estimote* provee el protocolo denominado *Nearable* [22], el cual permite enviar información contextual en un solo paquete, como por ejemplo, temperatura, nivel de batería, movimiento sobre un (x,y,z), orientación en el espacio, etc.

2.2 Desarrollo de Aplicaciones Móviles

En esta sección se hará hincapié solo en aquellas características que pueden ayudar al lector a comprender mejor la exploración que se presenta en este trabajo. Dada las características de las aplicaciones móviles sensibles a la posición [3], es de interés cubrir la mayor cantidad de dispositivos móviles, para llegar así a más usuarios. Acorde a esto, el desarrollo multiplataforma ([5], [6]) de este tipo de aplicaciones agiliza tiempos y cubre mayor cantidad de dispositivos. En [6] se clasifican estos desarrollos como: híbrido (por ejemplo, *PhoneGap*), interpretado (como es *React Native*), complicación cruzada, basado en componentes y desarrollo dirigido por modelos. Acorde a [6], estos dos últimos son los menos explorados en la bibliografía. Cabe mencionar que los enfoques *híbridos* muestran su funcionalidad sobre un *WebView*, mientras que los *interpretados* compilan a nativo esta funcionalidad. En [6] se presenta y detalla cómo es el flujo de ejecución para el acceso a características internas del dispositivo móvil, tanto en *PhoneGap* como en *React Native*, analizando ventajas y desventajas de cada uno.

Considerando la popularidad que tienen *PhoneGap* y *React Native*, estos fueron seleccionados para hacer la exploración presentada en este trabajo, sumado a que existen APIs disponibles ([12] y [13] respectivamente) para acceder a los datos de los *beacons*. En particular, estas APIs utilizan el protocolo de proximidad de *iBeacon/AltBeacon* que se describió en la Sección 2.1.1.

Un tema abierto es cómo abordar el mantenimiento de las aplicaciones móviles generadas a partir de desarrollos multiplataforma [5]. Esta problemática impacta en forma directa cuando este tipo de aplicaciones tiene que usar además sensores, por ejemplo, *beacons*; ya que se le suma la complejidad del mantenimiento de los mismos. Supongamos que un *beacon* deja de funcionar, el mismo debe reponerse en la misma posición; caso contrario, las aplicaciones móviles podrían brindar servicios inconsistentes, por ejemplo, al posicionar al usuario en un lugar distinto a la posición real del mismo. Esto guarda relación con lo que se plantea en [4], sobre la importancia de crear aplicaciones sensibles al contexto realmente usables.

3 Exploración de algunas APIs de proximidad

En esta sección se presenta la exploración realizada sobre el uso de dos APIs de *beacons* existentes, una para *PhoneGap* (desarrollo multiplataforma híbrido) [12] y otra para *React Native* (desarrollo multiplataforma interpretado) [13]. Para la exploración, se desarrollaron dos aplicaciones móviles, una en *PhoneGap* y otra en *React Native*; cabe mencionar que el foco está puesto en explorar el uso de las APIs y

cómo estas se comportan, y no se hace hincapié en el desarrollo de una aplicación a gran escala.

Para la exploración se usaron cuatro *beacons de posicionamiento* (*Location UWB Beacons*) de *Estimote* [14], los cuales se describieron en la Sección 2.1. Estos *beacons* pueden transmitir datos usando tanto protocolos de proximidad como de posicionamiento, ambos fueron descriptos en la Sección 2.1.1. *Estimote* provee una página de configuración (*Estimote Cloud)*, para setear diferentes valores a los *beacons* utilizados. Adicionalmente, *Estimote* brinda algunas aplicaciones *templates* que permiten probar algunas funcionalidades de los *beacons de posicionamiento*. Estas aplicaciones *templates* fueron usadas principalmente para poder comprender cómo funcionan estos *beacons*.

Cabe mencionar que las APIs de *beacons* usadas ([12] y [13]) funcionan sólo para el protocolo de proximidad de *iBeacon/AltBeacon*. Acorde a esto, se desarrolló la funcionalidad de las dos aplicaciones móviles para la exploración; las cuales monitorean la proximidad a los *beacons*; y al entrar en la proximidad de alguno de ellos, se brinda en la interfaz la información del mismo. Para esto, se aprovechó las características de *ranging* que provee el protocolo de proximidad de *iBeacon/AltBeacon* (como se describió en la Sección 2.1.1). Más allá de las diferencias que propone el desarrollo de aplicaciones *PhoneGap* y *React Native,* ambas APIs ([12] y [13]) no difieren en la interfaz de acceso a aquellos datos comunes.

La restricción de las APIs respecto al protocolo que usa (de proximidad) también impactó en cómo los cuatro *beacons de posicionamiento* fueron usados, es decir, solo se accedió a los datos brindados por el protocolo de proximidad de los mismos.

A continuación se presentan las principales características relacionadas al desarrollo de las dos aplicaciones móviles que usan APIs de *beacons* ([12] y [13]). Para más detalle de cómo funcionan en general *PhoneGap* y *React Native* se puede consultar [5], [6], [7] y [8]. En la Figura 2 se puede apreciar el código implementado en cada aplicación para dar comienzo al *ranging*.

a) PhoneGap

```
const uuid =
cordova.plugins.locationManager.BeaconRegion.
WILDCARD_UUID;
const major = undefined;
const minor = undefined;
const identifier = 'REGION1';
const beaconRegion = new
cordova.plugins.locationManager.BeaconRegion
(identifier, uuid, major, minor);
startRanging = () => {
  cordova.plugins.locationManager
    .startRangingBeaconsInRegion
    (beaconRegion)
    .fail((e) => {
      console.error('Error while starting
      ranging');
    })
    .done();
}
```

b) React Native

```
Beacons.detectIBeacons();
try {
  Beacons.setForegroundScanPeriod(2000);
  await Beacons.startRangingBeaconsInRegion
  ('REGION1');
  console.log(`Beacons ranging started
  succesfully!`);
} catch (error) {
  console.log(`Beacons ranging not started,
  error: ${error}`);
}
```

Fig. 2. Código para dar comienzo al *ranging*. 2.a) *PhoneGap*. 2.b) *React Native.*

En la Tabla 2 se puede observar la comparación de los datos obtenidos usando *ranging* con las dos APIs usadas. Por simplicidad, no se especifican en esta tabla los datos generales respecto a los *beacons*, ya que ambas APIs los brindan; estos datos son el UUID (*Universally Unique Identifier*) que identifica a un *beacon* y el *Major-Minor* que son valores numéricos que oscilan entre 0 y 65535, los cuales sirven para identificar a un *beacon* con mayor precisión.

Tabla 2. Comparación de los datos obtenidos usando *ranging* con las dos APIs de *beacons*.

Dato	Tipo	API para PhoneGap	API para React Native
Proximity (posibles valores: "*inmediate*", "*near*", "*far*")	String	si	si
RSSI (*fuerza de recepción de la señal*)	Double	si	si
TX *(fuerza de la señal de transmisión del beacon)*	Double	si	no
Distance *(en metros)*	Double	Calculable	si
Accuracy *(respecto de RSSI)*	Double	si	no

Cada una de las APIs usadas en los desarrollos solo permiten mostrar aquellos datos que se especifican en la Tabla 2. Por ejemplo, la aplicación desarrollada para *React Native* no puede acceder al TX o Accuracy; mientras que la aplicación *PhoneGap* puede mostrar la distancia siempre y cuando se implemente una función para tal fin, ya que esta es calculable. En la Figura 3 se puede observar el código de ambas aplicaciones para mostrar los datos obtenidos del *ranging*. Cabe destacar que solo se hace hincapié en el código relevante para mostrar las características mencionadas.

a) PhoneGap

```
function toString(beacon) {
  return (
    'Color: <span class="dot ' +
    this.getName(beacon.uuid) +
    '"></span> (' + this.getName(beacon.uuid) +
    ') <br> UUID: ' + beacon.uuid +
    '<br> major: ' + beacon.major +
    '<br> minor: ' + beacon.minor +
    '<br> Proximity: ' + beacon.proximity +
    '<br> rssi: ' + beacon.rssi +
    '<br> TX: ' + beacon.tx +
    '<br> Distance: ' +
    this.calculateDistance(beacon.rssi, beacon.tx) +
    ' meters <br> Accuracy: ' + beacon.accuracy +
    '<hr>'
  );
}
```

b) React Native

```
const beacon = ({ data }) => {
  return (
    <Text style={styles.text}>
      <ColorLabel uuid={data.uuid} />{'\n'}
      UUID: {data.uuid}{'\n'}
      major: {data.major}{'\n'}
      minor: {data.minor}{'\n'}
      Proximity: {data.proximity}{'\n'}
      rssi: {data.rssi}{'\n'}
      Distance: {data.distance}{'\n'}
    </Text>
  );
};
```

Fig. 3. Código para mostrar los datos del *ranging*. 3.a) *PhoneGap*. 3.b) *React Native*.

4 Discusión

En esta sección se presentan algunas lecciones aprendidas durante la exploración realizada. Se espera que estas contribuyan en la temática de los mecanismos de sensado indoor [1], [2], la cual sigue siendo un área abierta de investigación.

Como se mencionó en la Sección 3, se usaron las aplicaciones *templates* brindadas por *Estimote* para comprender como funcionan los *beacons*. Se detectó que estas aplicaciones no tenían un comportamiento estable, lo cual impactó en el tiempo invertido en comprender cómo funcionan los *Location UWB Beacons*.

En la Tabla 2 de la Sección 3 se pudo apreciar como dos APIs que usan el mismo protocolo de proximidad, difieren en algunos datos que proveen. Esto puede impactar a la hora de tener que elegir un desarrollo usando *PhoneGap* o *React Native*. Es decir, estas APIs no tienen un comportamiento unificado aun usando el mismo protocolo.

La curva de aprendizaje del desarrollo de aplicaciones *PhoneGap* y *React Native*, es diferente. *PhoneGap* requiere conocimiento Web estándar, mientras que *React Native* requiere además aprender, por ejemplo, *React* (librería para interfaces de usuarios). Cabe mencionar que la experticia que tenga el desarrollador en el uso de buenas prácticas impacta en el mantenimiento del código, esto está más guiado en *React Native* ya que al desarrollar se debe seguir cierta estructura. En el caso de contar con experiencia previa en el desarrollo de *PhoneGap* o *React Native*, solo se deberá aprender cómo funcionan las APIs de *beacons*, las cuales están bien documentadas. Sin embargo, lograr aplicaciones móviles sensibles al contexto realmente usables [4] dependerá de la experticia de los desarrolladores en la temática.

Se deja planteada la siguiente pregunta abierta: *¿es el desarrollo multiplataforma la mejor estrategia?*, por ejemplo, *Airbnb* [5] venía usando *React Native*, sin embargo anunció que migrará al desarrollo nativo. Acorde a lo planteado en [5], *¿tendrá que ver esta decisión de Airbnb con la incertidumbre que hay sobre cómo mantener aplicaciones móviles generadas a partir de desarrollos multiplataforma?*.

5 Conclusiones y Trabajos Futuros

En este trabajo se presentó la exploración de dos APIs de *beacons* existentes. Se desarrollaron dos aplicaciones móviles, una en *PhoneGap* y otra en *React Native*; las cuales usan estas APIs para detectar la proximidad a los *beacons*. Estas aplicaciones al entrar en la proximidad de un *beacon*, brindan la información del mismo. Este funcionamiento simple permitió poder explorar la información brindada por cada una de estas APIs, detectando que aun usando ambas el mismo protocolo de proximidad, las APIs difieren en algunos datos que proveen. Por ejemplo, un dato que difiere es la distancia, la cual solo es provista por la API de *React Native*. Sin embargo, la distancia es calculable por la API de *PhoneGap*. Además, el trabajo presentó un espacio de discusión en el cual se detallaron las lecciones aprendidas en la exploración realizada, en pos de contribuir a la temática relacionada al desarrollo de este tipo de aplicaciones.

Como trabajo futuro se espera poder armar guías que asistan a los desarrolladores tanto en el diseño como en el desarrollo de este tipo de aplicaciones. Para esto se

considerarán las lecciones aprendidas tanto en el área de aplicaciones móviles sensibles al contexto [3], [4] como de los mecanismos de sensado [1], [2].

Referencias

1. Davidson, P., Piché, R.: A survey of selected indoor positioning methods for smartphones. IEEE Communications Surveys & Tutorials 9 (2), 1347-1370 (2017)
2. Basiri, A., Lohan, E.S., Moore, T., Winstanley, A., Peltola, P., Hill, C., Silva P.F.: Indoor Location Based Services Challenges, Requirements and Usability of Current Solutions. Computer Science Review 24, 1-12 (2017)
3. Alegre, U., Augusto, J., Clark, T.: Engineering context-aware systems and applications: A survey. Journal of Systems and Software 117, 55-83 (2016)
4. Alegre-Ibarra, U. Augusto, J.C., Carl, E.: Perspectives on engineering more usable context aware systems. Journal of Ambient Intelligence and Humanized Computing, 1-17 (2018)
5. Martinez, M., Lecomte, S.: Towards the quality improvement of cross-platform mobile applications. In: IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, pp. 184-188. IEEE Press, New York (2017)
6. Biørn-Hansen, A., Ghinea, G.: Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. In: 51st Hawaii International Conference on System Sciences, pp. 5717- 5724. HICSS, Hawaii, Manoa (2018)
7. Página de PhoneGap, https://phonegap.com (último acceso: 06/07/2018)
8. Página de React Native, http://www.reactnative.com (último acceso: 06/07/2018)
9. Parikh, M., Shetty, M., Divita Vora, P., Koshy, R.: Smart Shopping Using Beacons. International Journal of Current Engineering an Scientific Research 5 (4), 1-6 (2018)
10. Klakegg, S., van Berkel, N., Visuri, A., Huttunen, H.L., Hosio, S., Luo, C., Ferreira, D.: Designing a context-aware assistive infrastructure for elderly care. In: UbiComp '17, pp. 563-568. ACM, New York (2017)
11. Kaulich, T., Heine, T., Kirsch, A.: Indoor Localisation with Beacons for a User-Friendly Mobile Tour Guide. KI-Künstliche Intelligenz 31(3), 239-248 (2017)
12. Repositorio del plugin iBeacon para PhoneGap, https://github.com/petermetz/cordova-plugin-ibeacon (último acceso: 06/07/2018)
13. Repositorio del plugin iBeacon para React Native, https://github.com/MacKentoch/react-native-beacons-manager (último acceso: 06/07/2018)
14. Página de Estimote, https://estimote.com (último acceso: 06/07/2018)
15. Shen, H., Cheng, P.H.: Using area judgment to adjust Indoor Positioning. In: 2017 International Conference on Applied System Innovation, pp. 272-275. IEEE Press, New York (2017)
16. Mohebbi, P., Stroulia, E., Nikolaidis, I.: Indoor Localization: A Cost-Effectiveness vs. Accuracy Study. In: IEEE 30th International Symposium on Computer-Based Medical Systems, pp. 552-557. IEEE Press, New York (2017)
17. Página de iBeacon, https://developer.apple.com/ibeacon (último acceso: 06/07/2018)
18. Página de AltBeacon, https://altbeacon.org (último acceso: 06/07/2018)
19. Página Estimote Monitoring, https://community.estimote.com/hc/en-us/articles/360003252832-What-is-Estimote-Monitoring (último acceso: 06/07/2018)
20. Página de Eddystone, https://developers.google.com/beacons (último acceso: 06/07/2018)
21. Página Estimote Location Protocol, https://community.estimote.com/hc/en-us/articles/208546097-What-is-a-beacon-protocol-Can-beacons-broadcast-multiple-packets-simultaneously (último acceso: 06/07/2018)
22. Página Estimote Nearable Protocol, https://community.estimote.com/hc/en-us/articles/203323543-What-are-Estimote-Stickers- (último acceso: 06/07/2018)