UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Framework para brindar soporte a la integración de diferentes mecanismos de sensado de posicionamiento en Aplicaciones Móviles
Autores: Ramiro Ongaro
Director: Dra. Cecilia Challiol
Codirector: Dra. Silvia Gordillo
Carrera: Licenciatura en Sistemas

Resumen

En los últimos años uno de los campos que más ha crecido y evolucionado es el de la computación móvil El amplio uso de los dispositivos móviles como herramienta personal se ha vuelto habitual en nuestro día a día. Acorde a ésto han surgido variadas aplicaciones para diferentes dominios. Un tipo particular de aplicación móvil, son aquellas basadas en posicionamiento. Este tipo de aplicaciones brinda información o servicios acordes a la posición del usuario. Pero no sólo han evolucionado los dispositivos, sino también los mecanismos para obtener la posición del usuario, como puede ser el uso de los Beacons.

La complejidad para desarrollar Aplicaciones Móviles basadas en Posicionamiento aumenta a medida que convive más de un mecanismo de posicionamiento para brindar información. Actualmente, no se cuenta con una solución estandarizada para el desarrollo de este tipo de aplicaciones. Esta es la motivación de la tesina, brindar un framework de solución que integre distintos mecanismos de sensado de posicionamiento permitiendo que la creación de este tipo de aplicaciones sea sencilla para el desarrollador.

Palabras Claves

Mecanismos de Sensado, Posicionamiento, Aplicaciones Móviles basadas en Posicionamiento, GPS, Beacons, códigos QR, PhoneGap

Conclusiones

El framework propuesto brinda soporte a la creación de Aplicaciones Móviles que requieren tener más de un mecanismo de sensado de posición funcionado simultáneamente. El framework provee distintos puntos de extensión, algunos de estos fueron apreciados al presentar los dos prototipos desarrollados. Ambos prototipos combinan el uso de GPS, beacons y códigos QR. Ésto permitió analizar cómo los mismos se comportan y cómo establecer prioridades cuando se combinan para brindar la posición actual al usuario.

Trabajos Realizados

- *Análisis de diferentes mecanismos de sensado de posición, para poder comprender su funcionamiento. También se analizaron distintos frameworks y arquitecturas de solución existentes para aplicaciones móviles sensibles al contexto.*
- *Análisis y planteo de un framework que permite integrar diferentes mecanismos de sensado de posición.*
- *Se realizaron dos prototipos funcionales, los cuales instancian el framework propuesto, como diferentes mecanismos de sensado.* En particular, se usaron *GPS, beacons y códigos QR*).

Trabajos Futuros

- *Realizar pruebas del framework con desarrolladores, para poder determinar la utilidad del mismo y así ir enriqueciéndolo con más funcionalidad reusable.*
- *Probar dominios particulares de aplicaciones para poder determinar requerimientos específicos de cada uno de estos. En base a ésto, tener extensiones del framework para cada domino particular.*
- *Analizar cómo se podría desacoplar la heurística de combinación de sensores para que los desarrolladores puedan indicar de manera sencilla qué mecanismo de sensado es prioritario en cada situación.*

Fecha de la presentación: Diciembre 2017

Índice

1 Introducción

1.1 Motivación

En los últimos años uno de los campos que más ha crecido y evolucionado es el de la computación móvil [Emmanouilidis et al., 2013]. El amplio uso del celular como herramienta personal se ha vuelto habitual en nuestro día a día. Acorde a ésto han surgido variadas aplicaciones para diferentes dominios. Un tipo particular de aplicación móvil, son aquellas basadas en posicionamiento. Este tipo de aplicaciones brinda información o servicios acorde a la posición del usuario. Es decir, a medida que el usuario se mueve por un espacio físico va recibiendo información (o servicios) acorde a la naturaleza de la aplicación. No sólo han evolucionado los celulares, sino también los mecanismos para obtener la posición del usuario, como puede ser el uso de los *Beacons*[1]. Al contar con diferentes mecanismos de posicionamiento conviviendo simultáneamente se genera más variabilidad en el tipo de servicios que pueden brindar las aplicaciones móviles basadas en posicionamiento.

Existen múltiples aplicaciones que hacen uso de diversos mecanismos de sensado de posicionamiento, incluso en algunos casos usando varios de ellos simultáneamente. Por ejemplo en [MuseoDeBrooklyn] se presenta una aplicación para el Museo de Brooklyn que le brinda información al usuario sobre las obras que están cercanas y de esa manera ayudar al staff del museo, haciendo uso de *Beacons*. Otro ejemplo puede ser la aplicación *Waze*[2] que brinda información del tráfico en tiempo real, esta aplicación usa el GPS para informar el estado del camino que está realizando el usuario hacia un destino que éste eligió. En el área de juegos, un ejemplo que tuvo un gran impacto a nivel mundial, en este último tiempo, es *Pokémon Go*[3] donde también se hace uso del GPS para poder ubicar a los jugadores y así proponerle diferentes acciones que forman parte del juego. En el ámbito educativo, en [Hui-Chun et al., 2010] se presenta una aplicación móvil para alumnos de primaria, esta aplicación propone diferentes actividades relacionadas con conceptos de Ciencias Naturales usando RFID (*Radio Frequency IDentification*). La idea es que el alumno conteste preguntas al mismo tiempo que puede observar el objeto (en este caso plantas).

Los ejemplos mencionados usan un único mecanismo de sensado de posicionamiento. También hay aplicaciones móviles que combinan varios mecanismos de sensado de posicionamiento. Por ejemplo, en [Hansen et al., 2012] los autores proponen una herramienta para la realización de dramas urbanos (*Urban Dramas*) con fines artísticos, turísticos y educativos. En el trabajo afirman que se pueden usar varias tecnologías como

[1] *Ibeacons* es un protocolo creado por Apple que fue la primera empresa que hizo conocida esta tecnología a nivel mundial, pero no fabricaba los beacons. Éstos son fabricados por empresas de terceros, por ejemplo, Estimote, Kontakt, Swirl. [Abdullah, 2016]
Página de Ibeacons de Apple: https://developer.apple.com/ibeacon (Último acceso: 18/04/17).
Página de uno de los fabricantes de Beacons: http://estimote.com (Último acceso: 18/04/17).

[2] Página de *Waze*: https://www.waze.com (Último acceso 05/04/17)

[3] Página de *Pokémon Go*: http://www.pokemongo.com (Último acceso 04/04/17)

por ejemplo, QR[4], RFID y GPS y dan ejemplos de aplicaciones usando estos mecanismos. Otro trabajo que usa varios mecanismos de posicionamiento, es descripto en [Cheng et al., 2016], en este trabajo se presenta una aplicación móvil que combina el uso de varios mecanismos de posicionamiento como: GPS, *Beacons* y NFC (*Near field communication*) para brindarle ayuda orientativa a los estudiantes de un campus universitario en Taiwán.

Cabe mencionar que en los trabajos previamente mencionados, no se brindan detalles del modelado de los aspectos relacionados con los mecanismos de sensado.

La complejidad de este tipo de aplicaciones aumenta a medida que convive más de un mecanismo de posicionamiento para brindar información. Acorde a ésto, existen soluciones que facilitan desarrollar este tipo de aplicaciones. Uno de ellos es [Dey et al., 1997], donde se presenta un framework basado en componentes que hace uso del contexto del usuario, en particular la posición del mismo. Dicha posición se puede obtener de forma explícita (indicado por el usuario, mediante un menú de acciones) o implícita (mediante el uso del GPS). Si bien estos conceptos son interesantes para tener en cuenta en una solución de modelado, dicho framework está discontinuado. Es decir, no se ha ido ajustando a las nuevas tecnologías que han ido surgiendo, sin embargo algunos conceptos generales son interesantes porque fueron las bases de muchos trabajos en el área de aplicaciones sensibles al contexto.

En [Fahy and Clarke, 2004] también se brinda una solución cuando hay en uso múltiples sensores. Los autores presentan CASS (*Context-awareness sub-structure*) un middleweare basado en servidor. Un aspecto a destacar de esta solución es que hace uso de múltiples contextos distintos, definiendo abstracciones y generalizaciones para este tipo de aplicaciones. El middleweare propuesto no tiene soporte en la actualidad. Sin embargo, los conceptos planteados por los autores son interesantes para tener en cuenta como una posible forma de generalización de conceptos para aplicaciones sensibles al contexto (como por ejemplo, la capacidad de que el usuario pueda cambiar en tiempo de ejecución la forma de tratar los sensores o manejar varios sensores simultáneamente).

En [Fortier et al, 2010] se presenta una arquitectura para la construcción de aplicaciones sensibles al contexto. Como parte de esta arquitectura, se provee un framework orientado a objetos, el cual brinda soporte de sensado. Los autores hacen foco en el soporte de la viabilidad y evolución de este tipo de aplicaciones. Esto es un aspecto a destacar de este trabajo. Sin embargo, el framework está desactualizado respecto de los sensores que usa.

Un framework para posicionamiento es propuesto en [Dongsoo et al, 2014], el cual se denomina *KAILOS*. Este framework está pensado para usarse de manera global, es decir, en edificios de todo el mundo. Para brindar el servicio de posicionamiento *KAILOS* describe tres componentes: mapas indoor y outdoor (o de radio), un sistema de posicionamiento híbrido y un sistema integrado de navegación outdoor/indoor. Los autores siguieron evolucionado este trabajo y en [Suk-Hoon et al, 2017], donde describen ejemplos que usan como mecanismo de sensado de posicionamiento outdoor, el GPS,

[4] Página con información de códigos QR: http://www.codigo-qr.es (Último acceso: 03/04/17)

mientras que para el sensado indoor usan WI-FI. Esta solución esta focalizada en la implementación y no se describe a nivel de modelado la solución implementada, sólo se menciona los componentes generales del framework. Lo interesante de este trabajo es que los autores continúan trabajando en el área.

Otra solución focalizada en la implementación es presentada en [Sneps-Sneppe and Namiot, 2016], donde se hace referencia al uso de los sensores, sin embargo estos autores no brindan un modelo de solución.

Actualmente todavía no se cuenta con una solución estandarizada para aplicaciones sensibles al contexto como se menciona en [Rivero-Rodriguez et. al, 2016]. Se pudo apreciar en los ejemplos mencionados anteriormente, que los dominios de las aplicaciones basadas en posicionamiento son muy variados. Por otro lado, varios de los framework mencionados no cuentan con soporte en la actualidad. O los trabajos actuales están focalizados sólo en la implementación sin brindar una solución de modelo. Por otro lado, tanto los frameworks como las implementaciones existentes, se focalizan principalmente en brindar soluciones para sensores físicos [Rivero-Rodriguez et. al, 2016] sin focalizarse en tomar los datos de sensores virtuales5, ingreso[6] de datos por parte del usuario o por una combinación de los mismos. Esta es la motivación de la tesina, brindar un modelo de solución que integre distintos mecanismos de sensado de posicionamiento (físicos, virtuales, ingreso por parte del usuario o combinados) permitiendo que la creación de este tipo de aplicaciones sea sencilla para el desarrollador. Se buscará contar, por ejemplo, con distintos mecanismos de sensores para modificar simultáneamente un mismo contexto, en este caso, el posicionamiento.

1.2 Objetivo

El objetivo de esta tesina es proponer una solución de modelado que permita brindar soporte a aquellas aplicaciones móviles basadas en posicionamiento, en las cuales se tiene más de un mecanismo de sensado de posición funcionado simultáneamente. Es decir, la posición del usuario es obtenida usando, por ejemplo, GPS y códigos QR.

El modelo de solución es genérico e independiente del dominio de la aplicación, esto permite que funcione como un framework para la construcción de este tipo de aplicaciones. Es decir, la solución propuesta brinda las bases para contar con la integración y combinación de mecanismos de sensado de posicionamiento. Esto permite que el desarrollador de este tipo de aplicaciones se focalice en los aspectos de dominio, heredando de nuestro framework aquellos aspectos relacionados con el sensado de posicionamiento. Al plantear la solución como un framework, este tiene puntos de extensión que podrán ser usados para la construcción de aplicaciones móviles basadas en posicionamiento orientadas a dominios específicos, por ejemplo turismo o educación.

[5] Los sensores virtuales toman información de aplicaciones o servicios existentes, por ejemplo de redes sociales. [Rivero-Rodriguez et al., 2016]

[6] Son ingresos de datos que sirven como valores para datos de contexto. El usuario indica explícitamente un determinado valor. Por ejemplo, "*estoy en el edificio de la facultad*". [Rivero-Rodriguez et al., 2016]

Usando como base el framework de modelado propuesto se desarrollaron dos prototipos de aplicación móvil basada en posicionamiento que muestra cómo combinar diferentes tipos de sensado de posicionamiento. Como parte de esta solución funcional se buscó una forma fácil de configurar nuestro framework para indicar cuáles son los mecanismos de posicionamiento que se desean usar, en este caso, se decidió realizarlo mediante la configuración de archivos XML.

Los prototipos se desarrollaron usando *PhoneGap* [PhoneGap] y combinan simultáneamente el uso de GPS, *Beacons* y códigos QR para determinar la posición del usuario. Esto permite explorar cómo se comportan estos mecanismos funcionando simultáneamente, y cómo se pueden crear este tipo de aplicaciones de manera sencilla para el desarrollador, sin tener que indagar en aspectos técnicos de bajo nivel de cada mecanismo de sensado.

1.3 Estructura de la tesina

A continuación se describe cada uno de los capítulos que integran la tesina.

En el Capítulo 2, se definirán y clasificarán algunos mecanismos de sensado, en particular aquellos que son usados actualmente para el posicionamiento del usuario. Se mostrará cómo el contexto puede ser usado de diversas maneras, dando ejemplo de múltiples aplicaciones móviles. Luego, se analizarán distintos enfoques de modelado y frameworks existentes para aplicaciones sensibles al contexto.

En el Capítulo 3, se describirá de manera más detallada la problemática que se busca resolver. Se introducirá el modelo para aplicaciones sensibles al contexto tomado como base. Usando este modelo como base, se explicará detalladamente los cambios realizados para poder obtener el objetivo propuesto.

En el Capítulo 4, se detalla cómo el modelo propuesto fue llevado a la práctica como un framework Java en particular implementado en *Android*. Este framework utiliza una librería XML desarrollada para poder instanciar de manera fácil y ágil los aspectos de los mecanismos de sensado que provee el framework. Además, se presentará el prototipo de aplicación desarrollado usando *PhoneGap* (en particular para ser empaquetado en *Android*). Dicho prototipo, permite el sensado simultáneo del usuario mediante el uso de GPS y *Beacons*. Este prototipo servirá para mostrar la puesta en práctica del modelo propuesto en el capítulo anterior.

En el Capítulo 5, se presentará un ejemplo particular de cómo funciona el prototipo desarrollado, permitiendo apreciar cómo funciona cada uno de los mecanismos de sensados involucrados en el mismo.

En el Capítulo 6, se presentarán conclusiones sobre el trabajo realizado y se mencionarán posibles trabajos futuros.

2 Background

En este capítulo se presenta una descripción de distintos mecanismos de sensado y cómo algunos autores los clasifican. Luego, se describen algunas aplicaciones basadas en posicionamiento, y se las analiza desde las perspectivas de los mecanismos de sensado. Finalmente se describen algunos frameworks o soluciones para este tipo de aplicaciones.

2.1 Mecanismos de sensado

Los mecanismos de sensado pueden ser clasificados de diferente manera, actualmente no hay una estandarización de criterio. Cada una de las clasificaciones existentes depende del foco dado por los distintos autores. Para esta tesina se decidió describir la clasificación presentada en [Rivero-Rodriguez et al., 2016] dado que la misma no sólo abarca sensores físicos sino también otras formas de tomar valores de contexto. Ésto nos permite tener un panorama más abarcativo, a la hora de las consideraciones que se deberían hacer al plantear una solución de modelado.

En [Rivero-Rodriguez et al., 2016] se identifican cuatro tipos distintos de formas de tomar valores de contexto, los cuales se presentan a continuación:

- *Sensores físicos*: Son sensores capaces de tomar datos del ambiente (por ejemplo: dentro de una casa) o de los dispositivos móviles. Este tipo de sensores son los que mayormente se usan en la actualidad en las aplicaciones móviles.

  Algunos ejemplos de sensores de este tipo pueden ser, por ejemplo, para tomar la posición del usuario: GPS o *Beacons*. Para tener más precisión, por ejemplo, para determinar hacia dónde está mirando el usuario se podría tener datos del acelerómetro del dispositivo móvil.

- *Sensores virtuales*: Son aquellos que tienen acceso a información virtual como, por ejemplo, información de aplicaciones y servicios. Muchas aplicaciones y servicios pueden ser considerados sensores virtuales, tales como el calendario, e-mail y navegadores. El uso de información virtual se ha visto incrementado en los últimos años en aplicaciones móviles, aunque dos factores dificultan su desarrollo:
  - Por un lado, es complejo tratar información virtual ya que está definida en formatos menos estructurados que la información que brindan los sensores físicos.
  - Además, sólo algunos proveedores de información virtual tienen métodos para que terceros puedan acceder a esa información, lo que complica su inclusión en aplicaciones sensibles al contexto. Sin embargo, cada vez se van creando más API para brindar a terceros datos de información existente.

  Un ejemplo de este tipo de sensor son las redes sociales, las cuales pueden proveer información virtual que sirva como un valor de contexto relevante, por ejemplo, la red de amigos.

- *Sensores combinados*: Proveen información combinando información obtenida de dos o más sensores. En este caso, necesitamos por lo tanto una forma de procesamiento para inferir nueva información, a partir de la información que proveen los sensores existentes.

  Es importante saber diferenciar entre los métodos asociados y los sensores en sí. En este caso, los métodos infieren información y se la provee a los sensores, los cuales registran la información que será usada posteriormente. Por ejemplo, información sobre la actividad física realizada por el usuario (inmóvil, caminado, andando en bicicleta, etc.), es información extraída de un sensor combinado. Para obtener esta información, se usan distintos métodos de inferencia.

- *Entrada directa del usuario*: Es una alternativa a la inferencia del contexto en base a datos obtenidos por sensores, ya sea físicos o virtuales (o una combinación de los mismos). En este caso el usuario provee explícitamente la información contextual, ya sea, por ejemplo, eligiendo desde un menú brindado por la aplicación móvil o escribiendo explícitamente la opción (generalmente con alguna nomenclatura definida). Un ejemplo de este tipo podría ser el usuario indica "*En casa*" o "*En la Facultad*", y estos datos son usados como valores de contexto para determinar dónde está el usuario.

A continuación se detallarán algunos sensores físicos existentes para proveer información relacionada al posicionamiento del usuario.

- *GPS*

  La técnica más popular es la llamada *Global Navigation Satellite Systems* (GNSS) [Abdullah, 2016]. GNSS es un sistema de satélites que son usados para precisar la ubicación geográfica del usuario en cualquier lugar del mundo. Actualmente hay dos GNSS operando: *Global Positioning System* (GPS) perteneciente a *Estados Unidos* y el *Global Orbiting Navigation Satellite System* (GLONASS) de origen *Ruso*. También hay dos GNSS en desarrollo [Abdullah, 2016]: *BeiDou Navigation Satellite System* (de *China*) y *Galileo* (de la *Unión Europea*).

  En la Figura 2.2.1 se puede apreciar un ejemplo simplificado de triangulación con GPS.

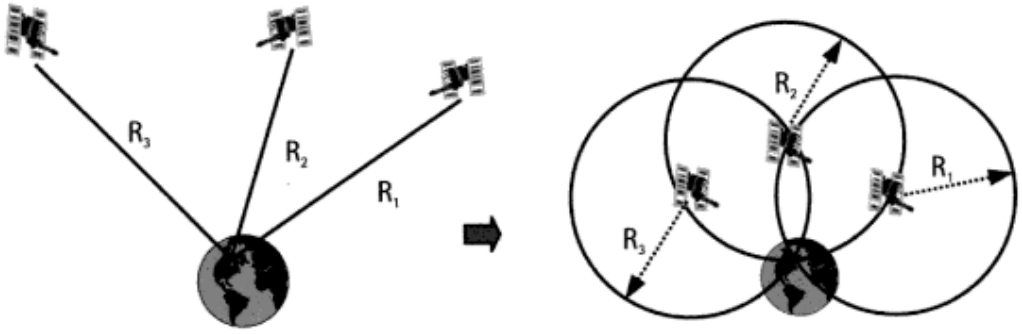

Figura 2.1.1: Imagen que representa la triangulación GPS [El-Rabbany, 2002]

A continuación se pueden apreciar algunas ventajas y desventajas [Abdullah, 2016].

- *Ventajas del GPS:*
  - Es una tecnología usada hace varios años, al punto tal que hoy en día es el sistema de posicionamiento más usado. Actualmente cualquier aplicación que quiera usar posicionamiento outdoor usar al menos el GPS. También vinculado a su madurez viene el fácil uso por parte del usuario gracias a las facilidades provistas por los distintos sistemas operativos.
  - Tecnología conocida no solo por usuarios expertos, sino también por usuarios sin experiencia, los cuales relacionan el GPS con posicionamiento outdoor (sin ser conscientes de los detalles técnicos), incluso saben que tienen que habilitar el GPS, por ejemplo en el dispositivo móvil, para poder hacer uso del mismo.

- *Desventajas del GPS:*
  - El GPS tiene algunas limitaciones cuando se lo usa para posicionar:
    - Tiene un margen de cinco metros de error, por lo que puede posicionar una persona en el patio de un vecino.
    - La precisión puede verse afectada por: obstáculos, condiciones atmosféricas, edificios, árboles, entre otros. Por lo que el GPS no funciona de manera precisa al usarse para posicionar en sistemas indoors o lugares subterráneos.
    - Dado que el GPS requiere comunicarse con varios satélites, puede agotar la batería de un celular rápidamente.
  - No hay un único estándar (está el GPS estadounidense, el GLONASS ruso, el *Galileo* europeo y el *BeiDou Navigation Satellite System* chino.

- *Beacon*

Desde la creación del *Bluetooth* en 1994 el potencial de comunicación que brinda se ha visto drásticamente incrementado [Abdullah, 2016]. *Bluetooth* es una tecnología estándar inalámbrica de intercambio de datos a corta distancia usando ondas de radio de corta longitud. Actualmente el *Bluetooth* puede encontrarse en casi todos los celulares, computadoras, televisión e incluso autos. En 2010 se lanzaron los *Bluetooth Low Energy* (BLE), poniendo a la tecnología *Bluetooth* en la vanguardia de las tecnologías para obtener contexto. Durante la *World Wide Developers Conference* del 2013, Apple presentó el protocolo *IBeacon* que usa *BLE* para transmitir paquetes de propaganda [Abdullah, 2016].

Estimote [Estimote] posiblemente sea la empresa más grande de *Beacons* en la actualidad, trabajando con empresas de la importancia de *Google*, *Apple*, *FedEx*, *Amazon*, *Toyota*, *Ford*, *Tesla*, *Porsche*, *Warner Bros*, *Barcelona FC*, *Tesco*, entre

otros. Teniendo ésto en cuenta, es claro que es un referente en lo que a *Beacons* respecta.

No sólo las empresas están interesadas en este mecanismo de sensado, sino también los gobiernos. La ciudad *Amsterdam* (*Holanda*) fue dotada de beacons, los cuales están accesibles a los programadores [BeaconsEnAmsterdam]. Otra ciudad que evoluciona usando *Beacons* es *Bucarest* (*Rumania*) donde tienen un proyecto de transporte público usando *Beacons*. La idea es dotar a los micros de la ciudad de *Beacons* para de esa manera, saber de ante mano cuándo y qué micro viene, facilitándole la información a personas con disminución visual. Más ejemplos de aplicaciones reales de beacons son nombrados en [Abdullah, 2016]. En la Figura 2.1.2 se puede apreciar un micro del proyecto de *Bucarest* mencionado anteriormente.

Figura 2.1.2: Un *Beacon* instalado en un autobus en [ExperienciaAutobusesBucarest].

A continuación se pueden apreciar algunas ventajas y desventajas [Abdullah, 2016].

- *Ventajas del uso de Beacons:*
  - La precisión es un punto a favor, aunque no hay unanimidad en cuanto a valores. Por ejemplo, en [Abdullah, 2016] se menciona que tiene una precisión de centímetros, mientras que en [Ning, 2015] se menciona que después de los dos metros de distancia la precisión empieza a decaer.
  - Fácil instalación indoor y outdoor.

- ❖ Consumo de batería bajo: del aparato en sí mismo y al trabajar con *BLE* consume mucho menos que otras tecnologías más populares y maduras, como por ejemplo el GPS.
- ❖ Tecnología relativamente nueva, en evolución y expansión. Empresas de renombre mundial están actualmente dedicando recursos a esta tecnología (por ejemplo, *Google*[7]).

➢ *Desventajas del uso de Beacons:*

- ❖ No hay homogeneidad: de protocolos *IBeacon* (Apple), *Eddystone* (Google) y otros fabricantes.
- ❖ Para espacios muy grandes se necesitan muchos *beacons*, pero para espacios pequeños la señal se puede solapar o generar problemas al posicionar: por ejemplo: si se tiene dos *beacons* (uno en cada habitación) y están pegados a la misma pared, puede ser que en algún momento la señal del *beacon* de la habitación contigua sea más fuerte y el sistema termine posicionando en la otra habitación [Bekkelien, 2012]. Por ésto, el posicionamiento de los *beacons* es algo crucial y delicado [Xin-Yu et al., 2015].
- ❖ En el caso de estar pegados a la pared pueden despegarse o ser robados.
- ❖ Tiene problemas de seguridad, ya que emiten señales que pueden ser tomadas por cualquier dispositivo.
- ❖ Control de las baterías, recambio y verificación de funcionamiento, como se describe en [MuseoDeBrooklyn].
- ❖ Los beneficios del bajo consumo de batería, pueden verse "neutralizados" si la aplicación tiene un servidor y hace un consumo intensivo del WI-FI o de los datos móviles.

Cabe destacar que a diferencia de otras compañías, *Estimote* [Estimote] provee distintos tipos de *Beacons*. A continuación se describen cada uno de ellos.

- *Location UWB Beacon*: *UWB* hace referencia a *Ultra Wide Band* o en español banda ultra ancha. Son *Beacons* que gracias al uso del *UWB*, pueden transmitir mucho mayor volumen de datos, sin ver afectado el consumo energético. Otra característica que los distingue de los demás *Beacons* es que pueden comunicarse entre sí. Gracias a estas dos particularidades, estos *Beacons* son usados para en poco tiempo, mapear un espacio físico con gran precisión.

  En la Figura 2.1.3 puede verse cómo estos *Beacons* se comunican y forman el esqueleto de una habitación. Las líneas punteadas de la figura establece la

[7] Página de *Beacon de Google*: https://developers.google.com/beacons (Último acceso: 25/08/17).

comunicación entre los *Beacons,* esta comunicación permite establecer distancias y así poder reconstruir el espacio físico, en este caso de una habitación.

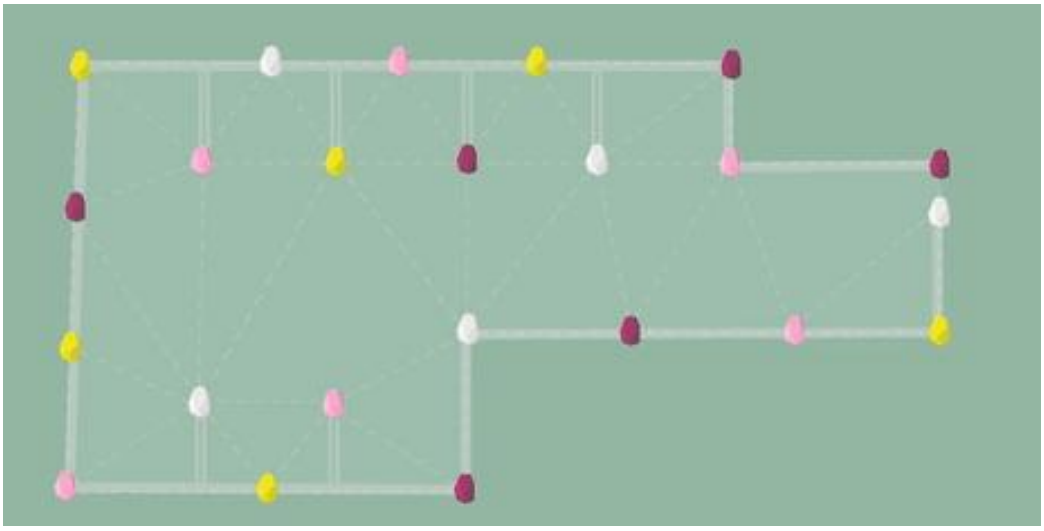

Figura 2.1.3: *Beacons* de posicionamiento UWB comunicándose (líneas punteadas) y formando el contorno de una habitación [Estimote]

- *Location Beacon*: Son *Beacons* que notifican en caso de que el usuario se encuentre en un área determinada. El uso más común es en museos (por ejemplo, se pone uno de estos *Beacons* en cada obra).

  En la Figura 2.1.4 se puede apreciar como usando distintos *Beacons* se puede posicionar a una persona.

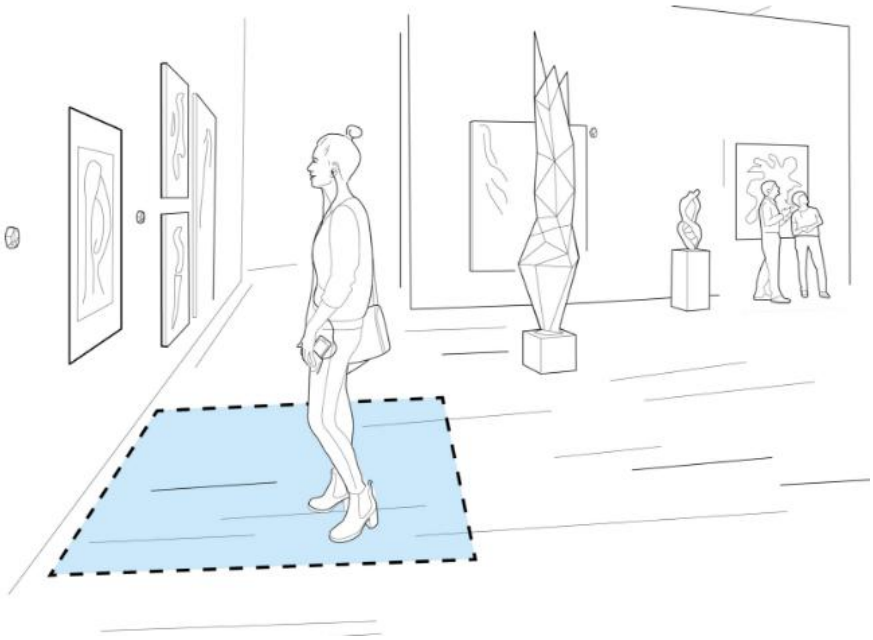

Figura 2.1.4: *Beacons* de posicionamiento instalados en un museo [Estimote]

- *Proximity Beacon*: Éste es el tipo de *Beacon* más común en el mercado y fue el primero en implementarse. Este tipo de *Beacon* emite señal en un radio, y se posiciona a la persona en base a la proximidad con el *Beacon*. Su uso más común es la propaganda. En las Figuras 2.1.5 y 2.1.6 puede observarse el funcionamiento de la proximidad de los *beacons*, en el dispositivo móvil se muestra información del *beacon* cercano.

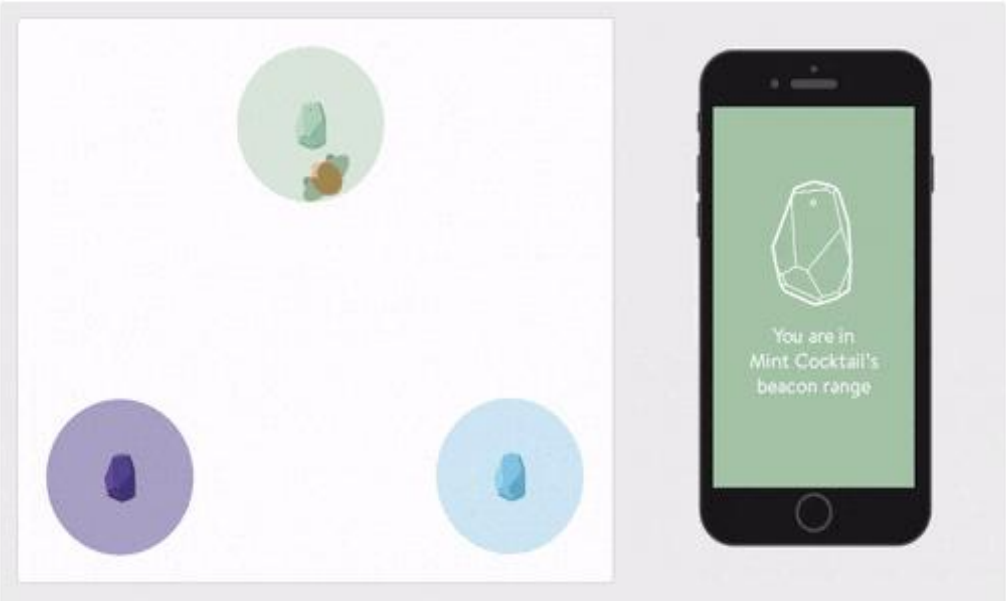

Figura 2.1.5: *Beacon* verde indicando proximidad a una aplicación. [Estimote]

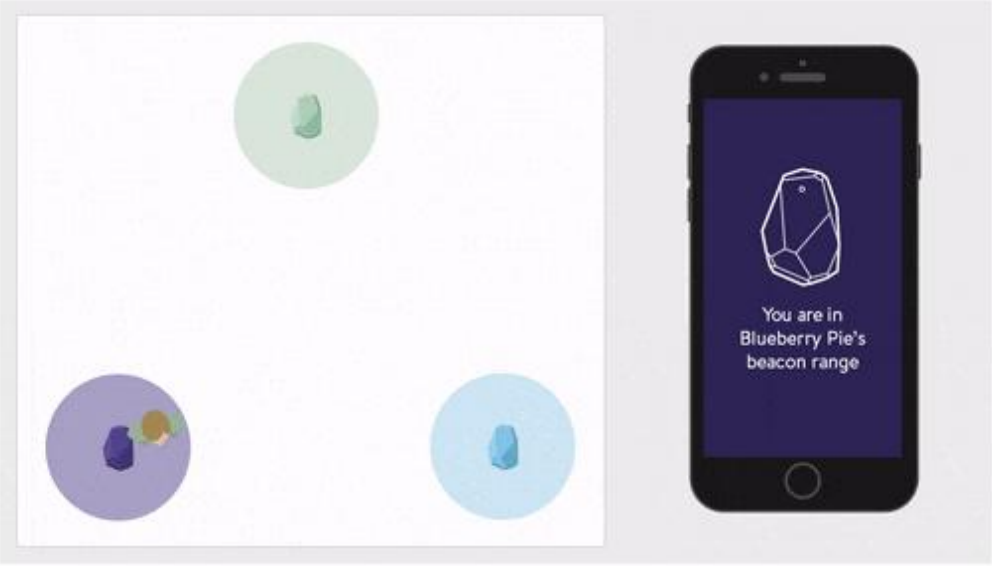

Figura 2.1.6: *Beacon* purpura indicando proximidad a una aplicación [Estimote]

- *Sticker Beacon*: En los casos anteriores, los *Beacons* se usan para identificar cosas estáticas, en el caso de los *Beacons Stickers* los objetos movibles. Ésto genera que puedan ser usados para diferentes aplicaciones.

En la Figura 2.1.7 se pueden apreciar los tamaños de cada uno de los *Beacons* descriptos de *Estimote*.

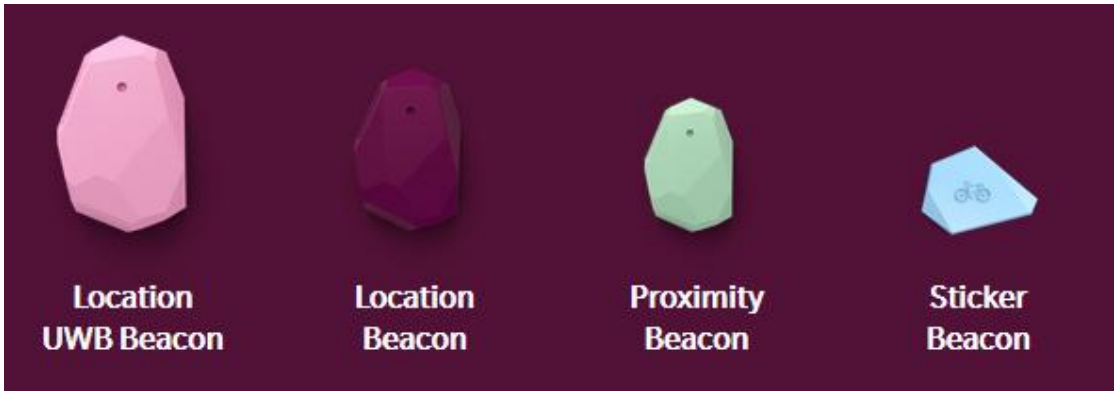

Figura 2.1.7: Variedad de *Beacons* de *Estimote* [Estimote]

- *NFC (Near Field Communication)*

  Es una tecnología inalámbrica de comunicación ultra corta basada en identificación de radiofrecuencia (RFID) [Cheng et al., 2016]. NFC actualmente es usado para aplicaciones de pago automático, venta de boletos, esquemas de lealtad, entre otros. Si bien posee un gran ancho de banda, tiene un bajo consumo energético, por lo que tiene un gran potencial para proveer posicionamiento indoor de gran precisión. Al igual que los *Beacons*, cada chip NFC tiene un único número de identificación (ID) asignado por el fabricante. De esta manera, al asociar un ID con un lugar físico y guardando esta información en una base de datos, la posición del usuario puede ser inferida con un grado de precisión extremadamente alto, cada vez que un evento de sensado ocurre.

  NFC puede pensarse como una extensión de RFID. Los intercambios NFC también involucran un iniciador y un objetivo como RFID. Los dispositivos NFC tienen dos modos de comunicación: Si el iniciador suministra la energía RF y el objetivo obtiene la energía del campo del iniciador, estos dispositivos están comunicando a través del modo pasivo. En cambio, si el objetivo y el iniciador tienen su propia fuente de energía, se comunicarían de manera activa. Estos modos también aplican a RFID [Jepson et al., 2012].

  Los dispositivos NFC tienen tres modos de operación [Jepson et al., 2012]. Los dispositivos pueden ser de lectura/escritura, es decir leen datos de un objetivo y lo escriben. Pueden ser emuladores de tarjetas, actuando como un tag RFID cuando están en el campo de otro NFC. O pueden usar el modo peer-to-peer mediante el cual hay un intercambio de datos en ambas direcciones.

El intercambio de datos entre dispositivos NFC y tags usa el formato de intercambio de datos NFC (*NFC Data Exchange Format*, NDEF). Los tipos bien conocidos de registros que maneja NDEF son:

- *Texto plano*
- *URIs*
- Smart *posters (posters inteligentes)*: contienen datos que pueden ser adjuntados para brindar más información. Los *smart posters* pueden incluir URIs, pero también podrían contener otros tipos de datos como podrían ser mensajes de texto sobre el poster, listos para ser enviados a tus amigos para avisarles sobre el poster. Un dispositivo que recibe un registro de poster inteligente, podría abrir un navegador, SMS, aplicación mail dependiendo del contenido del mensaje.
- *Firmas*: proveen un modo confiable de brindar información sobre los orígenes de la información contenida en un registro NDEF.

Se podrían mezclar y juntar registros en un mensaje NDEF o podría mandarse un mensaje por registro [Jepson et al., 2012].

En la Figura 2.1.8 se puede apreciar un ejemplo de uso de NFC para realizar pagos.

Figura 2.1.8: Pago usando NFC [NFCEjemplos]

A continuación se pueden apreciar algunas ventajas y desventajas [Jepson et al., 2012].

- *Ventajas de NFC:*
  - Gran precisión.
  - Bajo consumo energético.
  - Bueno para tareas que requieran transmisión de poca información. Por ejemplo: pasaje de passwords, realizar pagos, etc.
- *Desventajas de NFC:*
  - Corto rango (10 centímetros o menos).
  - Es relativamente lento comparado con WI-FI, Bluethoot y otros protocolos de comunicación.
  - No hay un único protocolo RFID, ni una única tecnología que haga uso del mismo.

- *Código QR (código de barra bidimensional):*

  Los códigos de barra se volvieron una tecnología popular debido a su rápida decodificación, gran precisión y bajo costo [Lai et al., 2010]. Primero surgen los códigos de barras de una dimensión que almacenan datos pequeños. Luego, surgen los códigos de barra bidimensionales, un tipo particular de éstos son los llamados código QR, los cuales fueron presentados bajo la ISO/IEC 18004. Un usuario puede usar un celular inteligente con una cámara para leer los códigos QR. Una vez escaneado, el código QR es decodificado usando un programa, para de esa manera obtener la información embebida.

  En la Figura 2.1.9 se puede apreciar como los códigos de barras fueron evolucionando hasta llegar a los códigos QR.

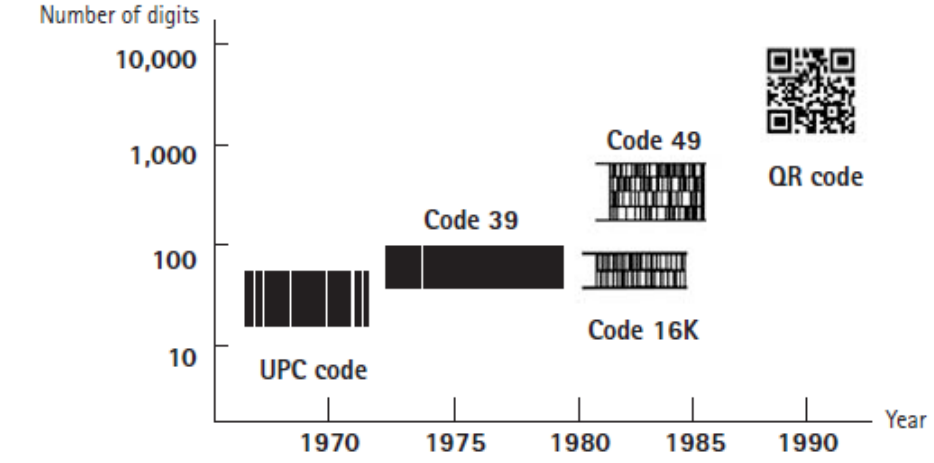

Figura 2.1.9: Evolución de los códigos de barra hasta llegar al código QR [Soon, 2008].

En la Figura 2.1.10 se puede apreciar la forma que tienen los códigos QR. Éstos tienen tres esquinas bien destacas que son las usadas por los programas de lecturas de códigos para decodificar la información.

Figura 2.1.10: Ejemplo de código QR.

Existen distintos programas para codificar este tipo de códigos. Los códigos QR pueden guardar distintos datos como por ejemplo: texto plano (caracteres alfanuméricos), imágenes, URLs, contactos de teléfono, enviar mensajes, entre otros.

A continuación se pueden apreciar algunas ventajas y desventajas [Lai et al., 2010] y [Soon, 2008].

- *Ventajas de los códigos QR:*
  - Baratos y fáciles de generar.
  - Alto margen de error y corrección de errores.
  - Rápida lectura.
  - No usa batería por tanto su duración es "ilimitada". Salvo que se deteriore.
  - Se pueden leer desde cualquier ángulo.
  - Muy usado en Asia.
- *Desventajas códigos QR:*
  - Información estática (una vez impreso el código QR no se puede cambiar).
  - Se necesita leer explícitamente.

Una vez descriptos los distintos mecanismos de sensado más comunes para el posicionamiento, a continuación se realizará una comparación entre los mismos, de las características más relevantes. En la Tabla 2.1.1 se puede apreciar la comparación respecto a diferentes características. Para posicionamiento en exteriores (outdoor) el GPS es de los mecanismos más usados y con mayor alcance. Cuando se requiere más

precisión otros mecanismos son utilizados. En el caso de requerir un sensado más preciso implícito, es decir que el usuario no tenga que realizar ninguna acción, en ese caso es mejor usar *Beacons*. Los códigos QR son los más baratos y fáciles de crear y muchas veces son elegidos por esta razón.

Tabla 2.1.1: Comparación de los mecanismos de sensado.

Tecnología	Consumo de Batería	Uso predominante	Obtención de la posición	Alcance	Precisión
GPS	Alto	Exteriores	Implícita	Casi todo el mundo	5 metros de error
Beacon	Bajo	Interiores	Implícita	Medio	Depende del tipo
NFC	Bajo	Interiores	Explícita	10 centímetros o menos	Exacto
Códigos QR	Bajo	Interiores	Lectura explícita	Corto	Muy preciso

2.2 Análisis de Aplicaciones Móviles basadas en Posicionamiento

En esta sección se presentarán primero algunas aplicaciones que usan un único mecanismo de sensado, para luego describir otras que hacen uso de varios mecanismos de sensado combinándolos para brindar determinados servicios.

- **Aplicaciones móviles que utilizan un único mecanismo de sensado de posición**

  Existen algunas aplicaciones móviles que hacen uso de un único mecanismo de sensado, a continuación se presentan algunas de ellas mostrando que las mismas pueden estar orientadas a diferentes dominios.

  - *Museo de Brooklyn [MuseoDeBrooklyn] [AskApp]*

    Basándose en experiencia anteriores, el Museo de Brooklyn [MuseoDeBrooklyn] decidió crear una aplicación móvil para lograr atrapar al visitante y brindarle un mejor feedback dentro del museo. La parte del *frontend* (interfaz) de la aplicación es simplemente un chat para enviarle texto y fotos a la gente del staff del Museo. De esta manera, los visitantes pueden hacer preguntas puntuales de distintas piezas de arte expuestas, adjuntando fotos asociadas a estas preguntas. En el *backend* de la aplicación se encuentra toda la complejidad, usa *beacons* posicionados en diferentes lugares del Museo. Estos *beacons* permiten que cada vez que un visitante realiza una pregunta, además se manda la posición del visitante acorde a los *beacons* cercanos. Esto permite al staff del Museo brindar respuestas con más precisión, ya que cuentan con expertos en áreas específicas. En este caso, el posicionamiento con *beacons* se usa para ver que personal del

staff debe responder a cada pregunta de los visitantes. En la Figura 2.2.1 se puede apreciar como el staff del museo coloca un *beacon* cerca de una obra de arte.

Figura 2.2.1: Staff del museo de Brooklyn instalando *Beacons* [MuseoDeBrooklyn]

A continuación se mencionan algunos problemas que tuvo el Museo de *Brooklyn* [MuseoDeBrooklyn] [AskApp] en relación a los *beacons*:

- Uno de los problemas que tuvieron fue la elección de los colores de los *beacons* y las pocas opciones que hay, inicialmente eran celestes y luego salieron los blancos, esto les generó no sólo tener que volver a comprar nuevos *beacons* sino además el cambio de los mismos. Ésto en un lugar como el Museo es muy importante ya que muchas veces podía afectar la estética de las obras de arte.
- En cuanto al pegamento de los *beacons* no era suficiente para mantenerlos pegados a las paredes de distintos materiales (yeso, cemento, vidrio, cartón yeso) y distintas pinturas (pintura brillante, plástica, semi), la empresa *Estimote* los asistió con más adhesivos lo que alivió un poco este problema.
    - Como consecuencia de las caídas, tenían que limpiarlos (debido a la textura ruborizada). Pero el mayor problema era que tenían que volver a poner el mismo *beacon* en el mismo lugar (no podían intercambiarse) y para ello tenían que usar una app para escanear el beacon y obtener su *major* y *minor number*, donde el *major*

indicaba la habitación y el *minor* el lugar en sí. Esto llevó a que tengan que tener un mapa de la distribución de los beacons.

- Otro problema es que no tenían forma de saber remotamente el estado de los *beacons* (batería, si estaban funcionales o no, etc), para averiguarlo tenían que usar una aplicación desde un celular, en el lugar donde estaba pegado el *beacon*.
    - Si bien el Museo podría usar la SDK de *Estimote* para tener la habilidad de leer el nivel de batería de cada *beacon*. Por un lado ésto les generaba estar atados a un único proveedor y el Museo quería mantenerse lo más independiente posible del vendedor de *beacons*. Por otro lado, usar la SDK generaba un desgaste extra en la batería de los beacons, con lo cual se volvía una solución inviable. Al decidir no usar la SDK de *Estimote* varias funcionalidades no van a ser explotadas [MuseoDeBrooklyn].
- La señal de los *beacons* fue otro problema, ya que la misma se ve interrumpida por cualquier cosa a excepción del aire, por ejemplo, las interferencias más comunes son: paredes, vitrinas, objetos, personas, entre otros.
    - El Museo comenta que han hecho pruebas en donde, estando al lado de un *beacon*, recibían una señal más fuerte de otro *beacon* que estaba del otro lado de la habitación. Ésto los llevo a implementar, por ejemplo, algoritmos para detectar el *beacon* más cercano [MuseoDeBrooklyn].
- También realizaron software que les solucionaba algunos problemas, por ejemplo una herramienta para agregar un *beacon* y emparejarlo con las posiciones de la galería. Esta herramienta les permite crear grupos de *beacons* y asignar más de un *beacon* a una posición. También brinda una vista preliminar de los objetos asignados a la posición, para de forma visual asegurarse que el *beacon* que se está instalando corresponde a ese conjunto de obras de arte.
- El staff del Museo uso latas de galletas para bloquear la señal de *beacons* que no estaban funcionales aún [MuseoDeBrooklyn].
- Cabe mencionar que el Museo tiene un edificio de aproximadamente 464.515,12 metros cuadrados y usaron alrededor de 150 *beacons*. Algo que mencionan [MuseoDeBrooklyn] es que no imaginan que esta solución escale sin realizar una gran cantidad de software a medida y teniendo *overhead* que contrarresta la idea de tener una solución liviana. Si bien

mencionan que están conformes con el resultado obtenido, también manejan la posibilidad de migrar a *Apple's Indoor Positioning*[8].

- *Waze [Waze]*

  La aplicación *Waze* brinda información del tráfico en tiempo real, esta aplicación usa el GPS para tomar la posición del usuario, y acorde a ésto informar el estado del camino que está realizando dicho usuario hacia un destino que éste eligió. También los usuarios pueden tomar un rol más activo e informar sobre distintos acontecimientos por ejemplo accidentes, controles, o cualquier obstáculo que se encuentren durante camino.

  Cabe mencionar que la aplicación *Waze* brinda información del tráfico en tiempo real y genera varias formas de llegar a un destino. Para hacer eso, la aplicación usa el GPS del celular (sensor físico) para posicionar al usuario. También tiene en cuenta información brindada por otros usuarios de la aplicación (sensado virtual) para elegir la mejor ruta, mientras que el destino es ingresado manualmente por parte del usuario.

  La aplicación tiene un gran factor social, no sólo porque se puede sincronizar con *Facebook*, *Twitter* o *Spotify*, sino porque permite crearte un perfil y desde el momento en que se usa la aplicación para hacer viajes se suman "*puntos*". También se suman puntos colaborando con la gran comunidad de conductores, por ejemplo indicando el estado de congestión de la ruta, si hay controles policiales, cámaras, algún accidente, precio del combustible, entre otros. Los puntos sirven para determinar el rango y obtener logros dentro de la aplicación. Este factor social es la "base" sobre la cual el sistema calcula los caminos. También sirve para hacer "*car pooling*", es decir, compartir el vehículo con otras personas que viajan al mismo destino.

  Waze también usa el calendario (sensor virtual) para brindar notificaciones sobre el momento de partir (dependiendo del estado del tránsito) para llegar a tiempo.

  En la Figura 2.2.2, se puede apreciar información contextual (posicionada) que se actualiza en tiempo real. En este caso, se muestra que la ruta está muy congestionada y se sugiere una opción para evitar el tránsito.

[8] Página de Apple's Indoor Positioning: https://developer.apple.com/maps (Último acceso: 25/08/17).

Figura 2.2.2: Ejemplo de funcionalidad de *Waze*[9]

- *Pokémon Go [Pokémon Go]*

  En el área de juegos, un ejemplo que tuvo un gran impacto a nivel mundial, en este último tiempo, es *Pokémon Go* donde también se hace uso del GPS para poder ubicar a los jugadores y así proponerle diferentes acciones que forman parte del juego. Otros juegos móviles basados en posicionamiento son presentados en [Xanthopoulos and Xinogalos, 2016].

  Un sensor físico diferente utilizado en *Pokémon Go* es el acelerómetro, gracias a éste detectan, por ejemplo, si el usuario va en un auto y advierten sobre los peligros del uso del juego al conducir. Un sensor que no es fundamental pero del cual se vale la aplicación es el giroscopio. Hace uso del mismo (cuando se está en modo realidad aumentada) permitiendo girar y ver todo el entorno, sólo para ver el pokémon en un punto particular. La hora del día también es algo que afecta el

[9] Imagen extraída del video: https://www.youtube.com/watch?v=PPpZNzXqId0 (Último acceso: 25/08/17).

juego y por tanto la aplicación aprovecha el uso del mismo. El último sensor físico para nombrar es la brújula, la cual también es usada por la aplicación. En la Figura 2.2.3 se puede ver en la esquina superior derecha el uso de la brújula. Indirectamente se puede ver que el celular hace uso del GPS (en la barra superior).

Figura 2.2.3: Una pantalla en vivo de alguien jugando *Pokémon Go*[10]

- *Aplicación Educativa [Hui-Chun et al., 2010]*

  En el ámbito educativo, en [Hui-Chun et al., 2010] se presenta una aplicación móvil para alumnos de primaria, esta aplicación propone diferentes actividades relacionadas con conceptos de ciencias naturales usando RFID (*Radio Frequency*

10 Imagen extraída de: https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=es_419 (Último acceso: 29/08/17).

IDentification). La idea es que el alumno conteste preguntas al mismo tiempo que puede observar el objeto (en este caso plantas).

La intención del trabajo [Hui-Chun et al., 2010] es ver si los niños aprendieron más usando la tecnología o por medio de un tour guiado normal. La prueba se realizó en el jardín de una escuela primaria de *Taiwán*, constando de trece áreas donde cada área tenía plantas para ser estudiadas. Las plantas tenían un tag RFID que las identificaba. El alumno usando una PDA equipada con un lector RFID y conexión inalámbrica, tiene que recorrer el jardín (ayudado por un sistema que lo guía) contestando las diferentes preguntas que se le presenten. En este caso, el RFID sirve para determinar la posición del alumno.

La aplicación también consta de un servidor que es el encargado de monitorear a los alumnos de manera individual y ejecutar el sistema de aprendizaje. Este sistema brinda pistas y guía de manera personalizada al alumno comunicándose a través del PDA. En la Figura 2.2.4 se puede apreciar el flujo de la aplicación.

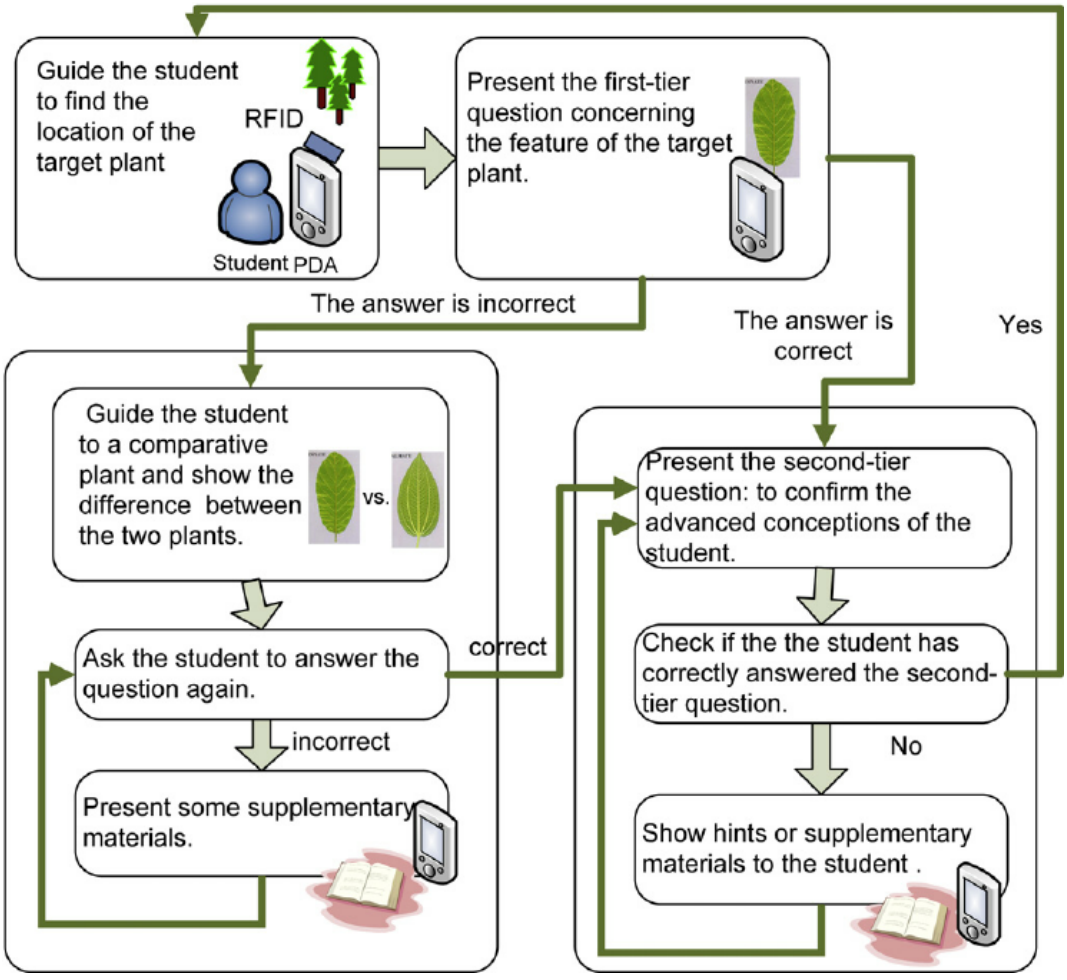

Figura 2.2.4: Flujo del sistema de aprendizaje personalizado para un caso particular [Hui-Chun et al., 2010].

- **Aplicaciones móviles que utilizan varios mecanismos de sensado de posición**

  Los ejemplos mencionados anteriormente usan un único mecanismo de sensado de posicionamiento. Sin embargo, existen algunas aplicaciones móviles que hacen de varios mecanismos de sensado, a continuación se presentan algunas de ellas.

  - *Dramas Urbanos [Hansen et al., 2012]*

    En [Hansen et al., 2012] los autores proponen la realización de dramas urbanos (Urban Dramas) con fines artísticos, turísticos y educativos. Los autores usan un XML para definir estos dramas, y mediante un framework que tienen definido, lo transforman en un modelo de objetos que posteriormente se ejecutará. Esto les permite que les sea más fácil soportar varias plataformas (navegadores o sistemas operativos como *Android*, IOS o *Symbian*).

    En [Hansen et al., 2012] se indica que se pueden usar varias tecnologías como QR, RFID y GPS, para el sensado del posicionamiento pero no brindan detalles de cómo llevan a cabo la instanciación de los mismos. Sin embargo, es interesante destacar que para desencadenar los eventos de la historia se usan entradas. Estas entradas pueden ser cualquier cosa (entrada manual, lectura de un código QR o GPS), ya que en términos del sistema no diferencian el input, sólo necesitan que dicha entrada pueda ser mapeada a un único evento de la historia. En las Figura 2.2.5 se puede apreciar distintas pantallas descriptas en [Hansen et al., 2012].

Figura 2.2.5: Pantallas de un *Drama Urbano* [Hansen et al., 2012]

- *Sistema de Guía Móvil [Cheng et al., 2016]*

  Otro trabajo que usa varios mecanismos de posicionamiento, es descripto en [Cheng et al., 2016]. En este trabajo se presenta una aplicación móvil que combina el uso de varios mecanismos de posicionamiento como: GPS, *Beacons* y NFC para brindarle ayuda orientativa a los estudiantes de un campus universitario en *Taiwán*. También presentan un framework que usaron para la realización de esta aplicación. Este framework presentado en [Cheng et al., 2016] consta de cuatro componentes:

  - *Diseño de almacenamiento de datos*

    Este componente es el encargado de guardar la información de los lugares. Para simplificar el almacenamiento y el manejo de tareas, los autores guardaron cuatro estructuras distintas, estas son:

    - *Estructura de Mapa Outdoor:* consta de seis campos id, nombre, dirección, latitud, longitud e info (breve descripción del edificio).
    - *Estructura de Mapa Indoors*: contiene cinco campos id, map-name (usado para almacenar el ccódigo correspondiente al mapa indoor), nodo (usado para indicar el nombre del nodo, posición, en el mapa indoor), x, y.
    - *Estructura de datos de Beacon*: consta de id, nodo (nodo que lo identifica en el mapa indoor), fid (el ID único asignado por el fabricante), nombre (nombre de la localización), middle (nombre del edificio en el que está ubicado) y far (el nombre del área general del campus donde está ubicado el edificio).
    - *Estructura de datos de* NFC: igual al de los *beacons* con la diferencia de que no tienen fid, tiene uid que es el tag uid.

  - *Posicionamiento*

    Es el módulo encargado de devolver un JSON con el mapa correspondiente ante la petición del usuario. Por default la aplicación arranca en modo "*outdoor*", así que lo primero que se hace es obtener la posición GPS y la manda al servidor junto con el ID del dispositivo, en consecuencia la posición del usuario es actualizada y el servidor busca por esas coordenadas y retorna el JSON con el mapa correspondiente.

    Cuando un usuario se acerca a un ambiente *indoor*, la aplicación automáticamente cambia a modo "*indoor*" y lanza una rutina de posicionamiento *indoor*. Si se detecta un *Beacon*, ocurre un evento y se actualiza el ID en el dispositivo y en el servidor. Usando el ID como clave, el servidor recupera la posición aproximada del usuario y retorna la información al celular. Una petición similar ocurre cuando se lee explícitamente un tag NFC.

- *Algoritmo de camino más corto*

  Después de obtener la posición del usuario, la aplicación descarga el mapa al celular desde el servidor. La aplicación informa al servidor del ID del dispositivo y pide por un ID de mapa particular. El servidor busca en la base de datos el correspondiente mapa y lo retorna. Básicamente guardan datos de los nodos y por medio del algoritmo de *Dijkstra* obtienen el camino más corto.

- *Mapas*

  En general el éxito de una aplicación se debe en gran medida a la apariencia y lo intuitivo de la interfaz gráfica de usuario (GUI). Para sistemas guías como el propuesto en [Cheng et al., 2016], un mapa pictórico con mucha información detallada puede simplemente ayudar a confundir al usuario. Consecuentemente en la aplicación de este trabajo los mapas *indoor* y *outdoor* son presentados en la forma de "*mapas subterráneos*", en la cual las posiciones claves (edificios, salones, oficinas y baños) son representados como nodos y la distancia es indicada como un número ubicado en el camino.

En [Cheng et al., 2016] se menciona que tuvieron complicaciones al trabajar con los *Beacons* para posicionar. Realizaron un estudio con *Beacons* a distancias de 1, 2 y 4 y metros, usando distintas marcas de celulares (*HTC One*, *Samsung Galaxy S4* y *Sony Xperia Z1 Compact*). Se tomaron cien *mediciones de muestra* en un lapso de 30 minutos en cada caso. Para las distancias de 1 y 2 metros, el teléfono *Sony* fue el más estable en cuanto a señal y se acercó mucho al valor real, mientras que el HTC y *Samsung* tuvieron variaciones significativas. Para la distancia de 4 metros los tres celulares tuvieron una performance pobre para posicionar.

La prueba de performance fue realizada en un campus en *Taiwán*, en total usaron 7 *Beacons* y 31 NFC tags distribuidos como marca la Figura 2.2.6. Fueron puestos en puertas de las habitaciones principales, entradas de escaleras, bifurcaciones o esquinas de los pasillos. Los autores comentan que la razón por la cual los NFC superan enormemente en número a los *Beacons* es el precio y que posicionaron tags cada aproximadamente entre 2 y 15 metros.

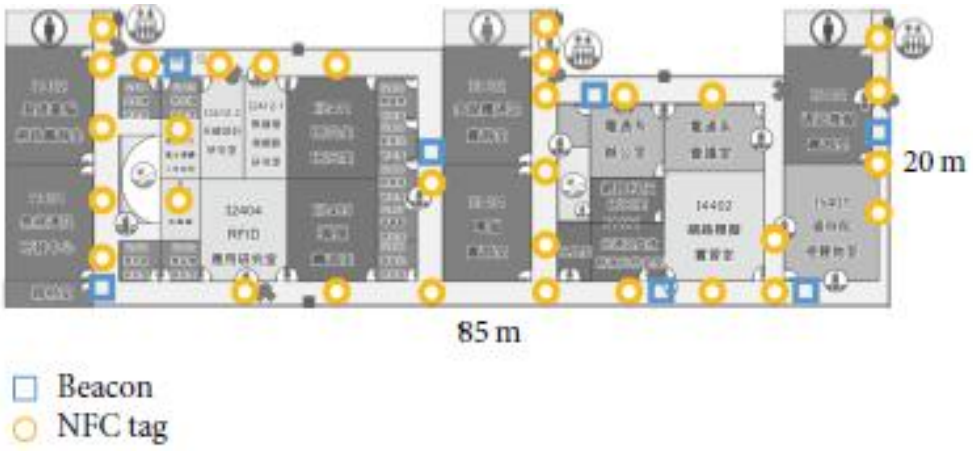

Figura 2.2.6: Distribución de *Beacons* y tags NFC [Cheng et al., 2016].

En [Cheng et al., 2016] como se mencionó anteriormente se combina tecnologías de posicionamiento, logrando aprovechar la cobertura, precisión y detalle que provee cada una. En la Figura 2.2.7 se muestra lasún grafico esquemático de los tres mecanismos utilizados. Puede verse que el GPS es el de mayor cobertura y se usa para "*distancias grandes*", los *Beacons* para "*distancias medias*" y los tags NFC para distancias "*cortas*".

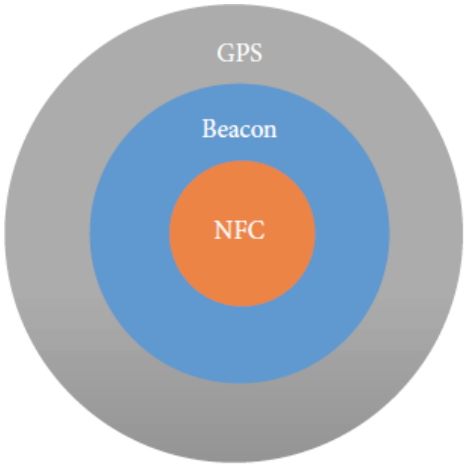

Figura 2.2.7: Cobertura de las tres tecnologías usadas en [Cheng et al., 2016].

2.3 Aplicaciones Sensibles al Contexto

Cabe mencionar que los trabajos descriptos en la Sección 2.2, no brindan detalles del modelado de los aspectos relacionados con los mecanismos de sensores. En esta sección se describirán algunas soluciones que facilitan desarrollar este tipo de aplicaciones, haciendo foco en aquellas características arquitecturales o de modelado de las mismas.

No hay que perder de vista que las aplicaciones sensibles al contexto han sido investigadas desde hace ya más de 20 años, sin embargo todavía no se ha logrado contar con una solución unificada, como se menciona en [Bauer and Dey, 2016] y [Alegre et al., 2016].

Cabe mencionar que desde los inicios de las investigaciones en esta temática, no se contaba con el avance tecnológico actual, con lo cual muchas de las primeras investigaciones eran conceptuales o han quedado desactualizadas. A continuación se mencionan diferentes trabajos que son interesantes para el foco de esta tesina.

- **Framework conceptual basado en componentes [Dey et al., 1997]**

  CyberDesk es un framework conceptual basado en componentes escrito en Java presentado en [Dey et al., 1997]. Este framework actualmente está discontinuado. Sin embargo, dado que el mismo provee conceptos interesantes para esta tesina, fue incluido en este análisis.

  Uno de las características a destacar de *CyberDesk* es que provee pequeños módulos de software que servirán como "*ladrillos*" de un edificio para crear una aplicación más grande. Ésto facilita pensar el software para que el mismo pueda evolucionar en el tiempo como se menciona en [Weyns et al., 2015].

  En [Dey et al., 1997] se hace referencia a un prototipo, para moverse por el campus de *Georgia Tech*, donde el contexto del usuario se puede obtener de forma explícita (indicado por el usuario, mediante un menú de acciones) o implícita (mediante el uso del GPS).

  En la Figura 2.3.1 se pueden apreciar los principales componentes de ejecución de *CyberDesk*, conceptos que se pueden apreciar, el contexto del usuario (*user context*) y la necesidad de muchas veces tener convertidores (*converter*). Estos son conceptos que se van a ver presentes en la mayoría de las arquitecturas o modelos existentes para este tipo de aplicaciones. Otro concepto identificado son los *Servicios*.

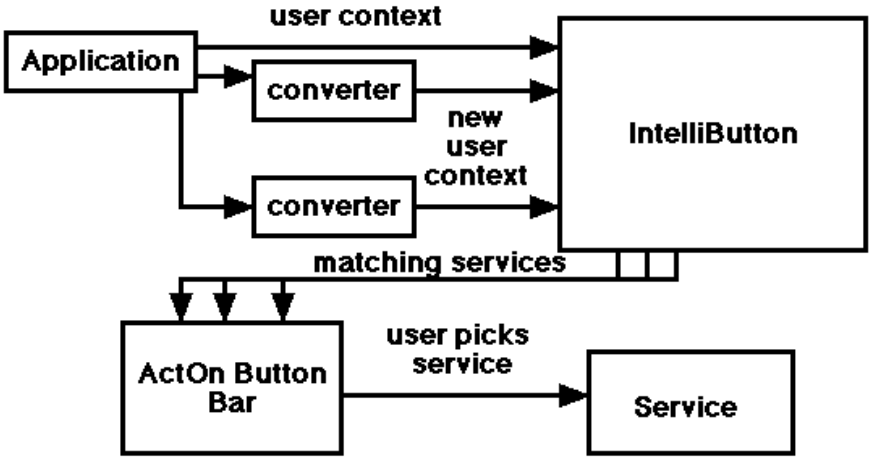

Figura 2.3.1: Arquitectura de ejecución de *CyberDesk* [Dey et al., 1997].

- **CASS (*Context-awareness sub-structure*) [Fahy and Clarke, 2004]**

  En [Fahy and Clarke, 2004] también se brinda una solución cuando hay en uso múltiples sensores, esta solución los autores la denominan CASS (*Context-awareness sub-structure*). Esta solución es un middleweare basado en servidor, para brindar soporte a un gran número de contextos y proveer abstracciones de alto nivel de esos contextos sin que implique un procesamiento o uso de memoria adicional. Ésto es una característica a destacar de esta solución.

  Si bien la idea propuesta presentada en [Fahy and Clarke, 2004] es atractiva, la solución no tiene soporte en la actualidad. Sin embargo, los conceptos planteados por los autores son interesantes para tener en cuenta como una posible forma de generalización de conceptos para aplicaciones sensibles al contexto (como por ejemplo, la capacidad que el usuario pueda cambiar en tiempo de ejecución la forma de tratar los sensores o manejar varios sensores simultáneamente).

  En la Figura 2.3.2 se pueden apreciar los conceptos abordados en [Fahy and Clarke, 2004]. Se representa la abstracción del concepto del *Sensor*, los cuales pueden ser de varios tipos concretos, como el sensor de luz, temperatura, etc. Se identifica el concepto de estar escuchando por eventos (*SensorListener*) del s*ensor* como así también la identificación del cambio (*ChangeListener*). El concepto de interpretar datos del sensor (*Interpreter*). Otra característica que modelan tiene relación con las reglas para poder recuperar valores (*RuleEngine*).

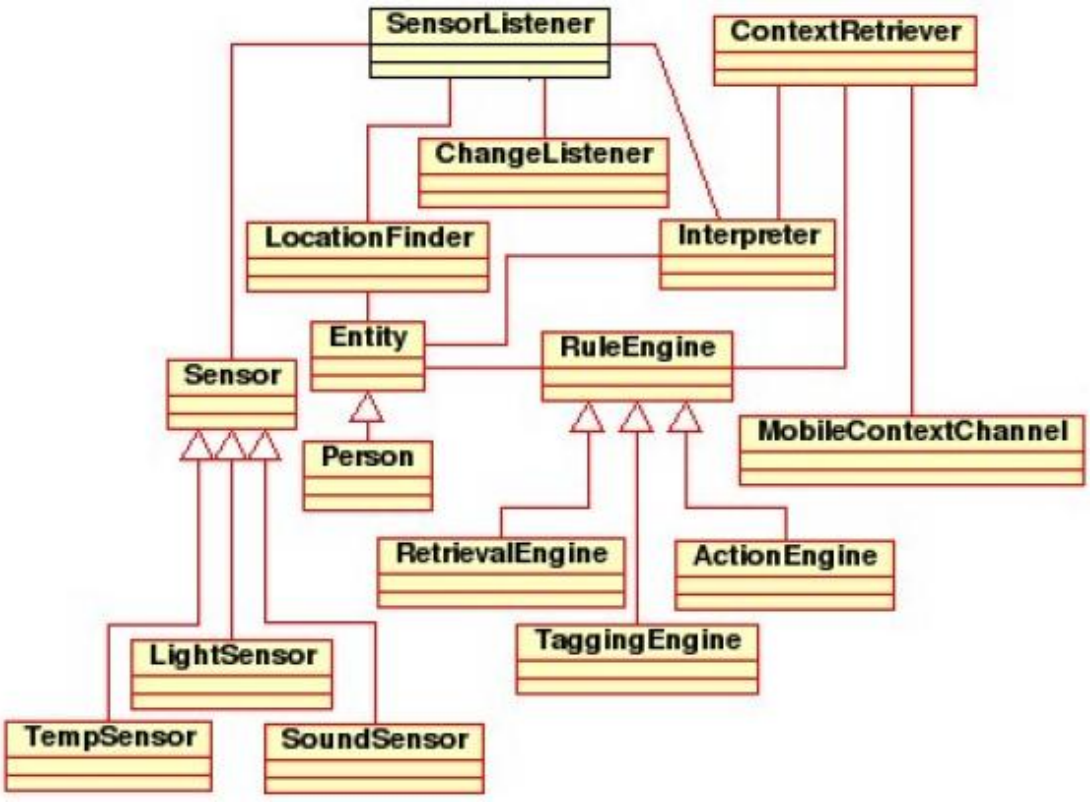

Figura 2.3.2: Conceptos considerados en CASS [Fahy and Clarke, 2004].

Si bien en la Figura 2.3.2 se pueden apreciar algunas abstracciones interesantes de conceptos, en [Fahy and Clarke, 2004] no se especifican detalles de cómo estos conceptos interactúan o la funcionalidad concreta de los mismos. Son de interés de esta tesina presentar esta figura para poder ir identificando características en común entre los distintos enfoques a lo largo de los años.

- **Arquitectura para la construcción de aplicaciones sensibles al contexto considerando variabilidad en tiempo de ejecución [Fortier et al, 2007]. [Fortier et al, 2010], [Musi, 2016]**

En [Fortier et al, 2007] y [Fortier et al, 2010] se presenta una arquitectura para la construcción de aplicaciones sensibles al contexto, en la Figura 2.3.3 se puede observar cómo está compuesta dicha arquitectura.

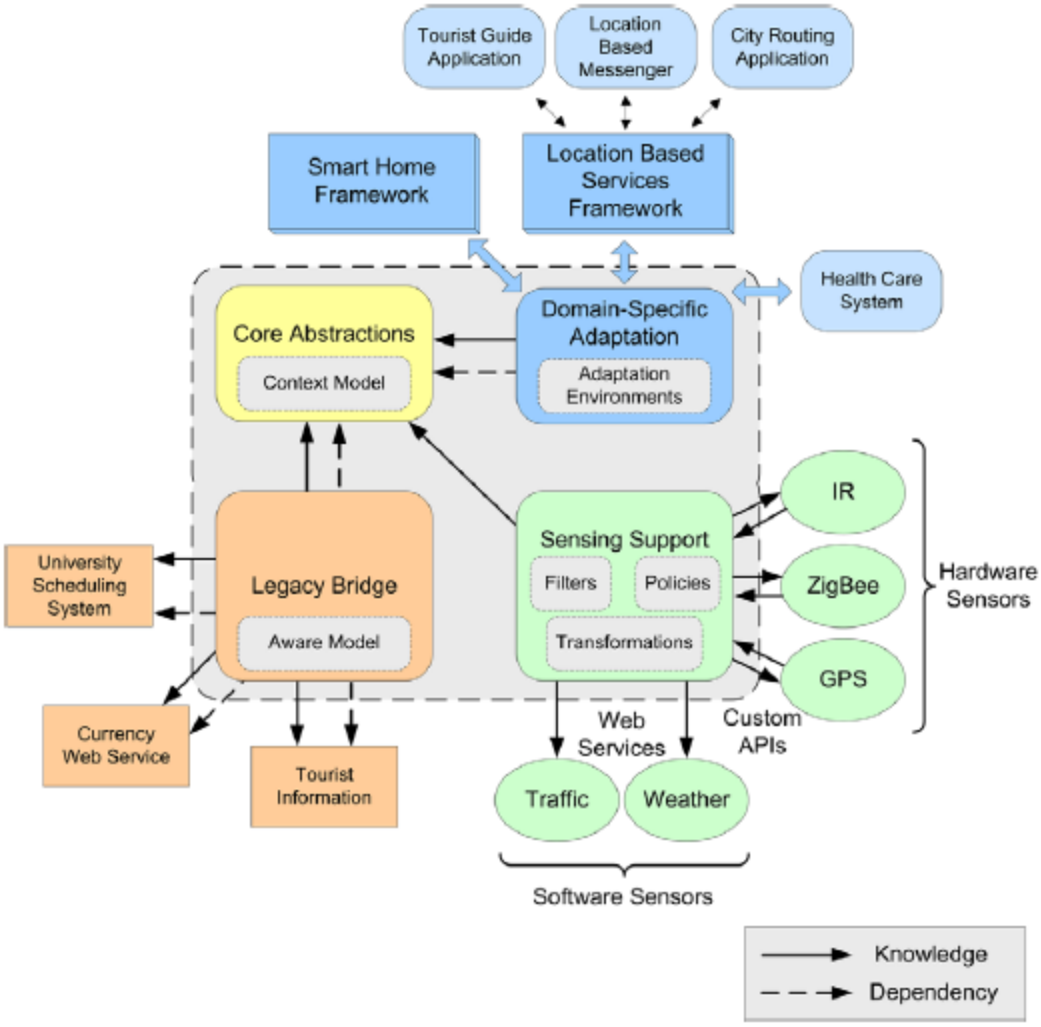

Figura 2.3.3: Arquitectura para aplicaciones sensibles al contexto [Fortier et al, 2010].

Se puede observar en la Figura 2.3.3 cuatro componentes centrales: *Core Abstractions, Domain-Sapecific Apactations, Legacy Bridge* y *Sensing Support.* Los

autores proponen un framewrok para brindar soporte a la arquitectura propuesta focalizándose en estos cuatro componentes centrales.

Cabe mencionar que en [Fortier et al, 2010] los autores hacen foco en brindar soporte para la variabilidad y evolución de este tipo de aplicaciones, y esto es considerado en el framework que proponen. Sin embargo, el framework está desactualizado respecto de los sensores que usan.

En la Figura 2.3.4 se pueden apreciar los principales conceptos relacionados con contexto que modelan. Por un lado, los ambientes de adaptación (*AdaptationEnviromentent*) que cuentan con objetos que tienen características contextuales, estos objetos se denominan *AwareObject*, y cada una de sus características contextuales se representa con la clase *ContextFeature*. De esta forma, los autores proponen un framework genérico que puede ser usado para cualquier dominio donde sea necesario representar contexto.

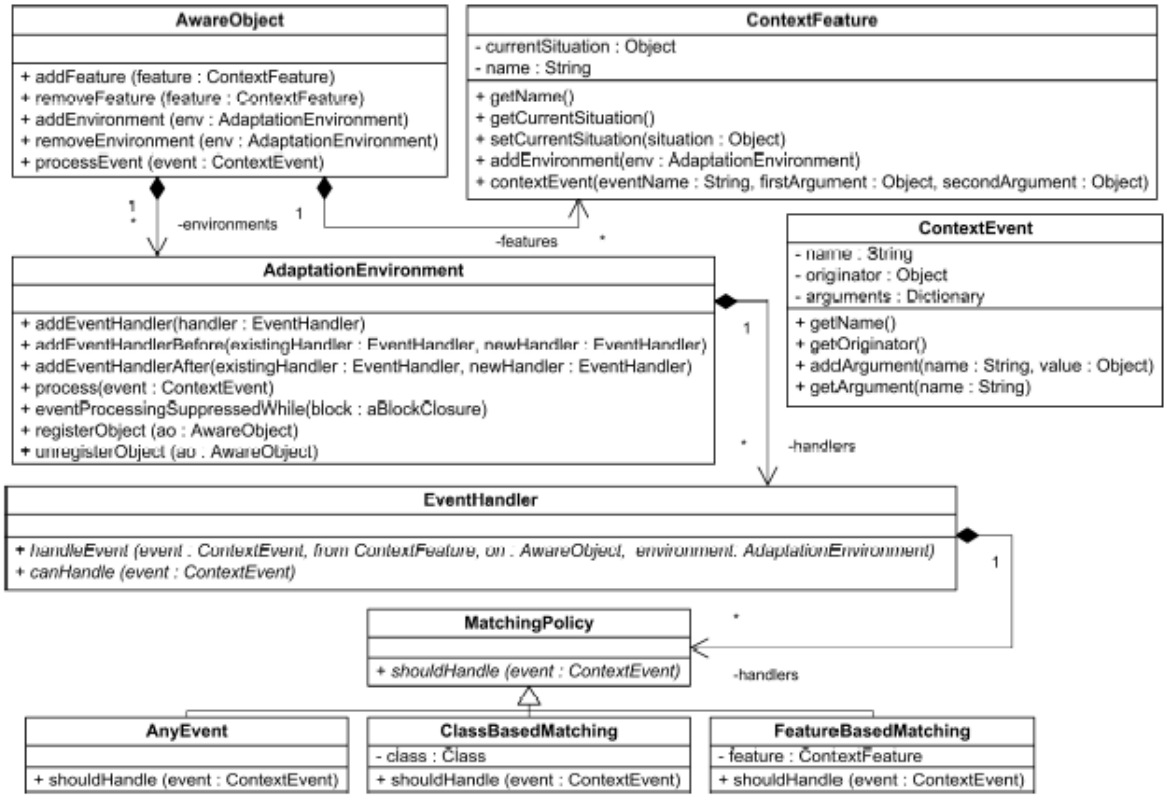

Figura 2.3.4: Modelado del ambiente de adaptación [Fortier et al, 2007].

Todo el comportamiento relacionado con el funcionamiento que se activa cada vez que cambia algún contexto es representado por la clase *EvenHandler*, que trabaja en conjunto con la política de matcheo para determinar si ese evento es relevante o no para desencadenar cierta funcionalidad.

Los autores presentan en [Fortier et al, 2010] toda una solución de diseño para poder representar los conceptos relacionados a los sensores y cómo estos operan. El modelo propuesto se puede observar en la Figura 2.3.5.

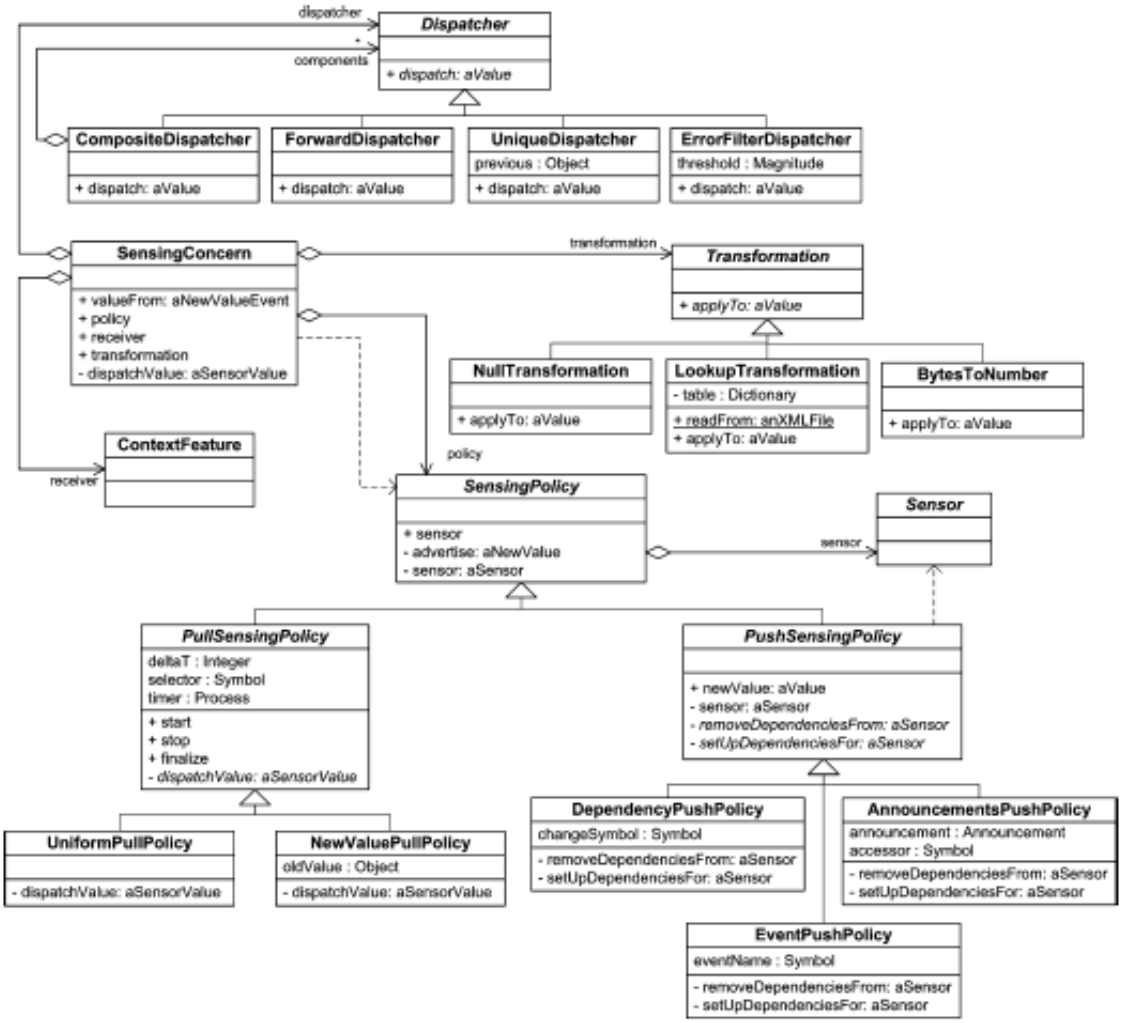

Figura 2.3.5: Clases relacionadas al sensado [Fortier et al, 2010].

A continuación se brindará más nivel de detalles de los conceptos presentados en la Figura 2.3.5.

- *SensingConcern*: es la clase encargada de coordinar desde que se detecta un cambio en algún sensor hasta que esto se propaga a la característica de contexto concreta para actualizar su valor. Esta clase conoce otras clases que colaboran con ella, las cuales se detallan a continuación:
    - *SensingPolicy*: determina la política de sensado, la cual puede ser push (*PushSensingPolicy*) es decir, que está constantemente avisando cada vez que cambia el dato de sensor. O sino la política puede ser pull,(*PullSensingPolicy*) da aviso cuando se lo solicitan.

      Además, la clase *SensingPolicy* conoce al Sensor concreto, esto es una abstracción tanto de los sensores virtuales o físicos.
    - *Transformation*: cada clase que hereda de *Transformation* define como transformar el valor sensado, a un valor compatible con el

funcionamiento de la aplicación. Por ejemplo, si el valor sensado como *string* debe convertirse en un valor numérico.

- *Dispatcher*: cada clase que hereda de *Dispatcher* define cual es el criterio para determinar si ese valor debe propagarse y avisar del cambio. Por ejemplo, si un valor es igual que el valor que se venía sensado no se propaga, ya que se determina que no hay ningún cambio.
- *ContextFeature*: esta clase es la misma que la presentada en la Figura 2.3.4, y cuando un *dispatcher* determina que el valor sensado debe propagarse, este actualiza el valor de la *ContextFeature* concreta.

La secuencia de ejecución comienza con un cambio en el sensor, en el caso de ser una política *push* se le da aviso al *SensingConcern,* el cual ejecuta su *Transformation,* para determinar si ese valor necesita de alguna transformación o ajuste; luego se le pasa el control a su *Dispatcher* y este determina si ese valor debe ser actualizado en la *ContextFeature* o no. En el caso de tener una política *pull*, sólo cambia en que todo el proceso empieza a ejecutarse explícitamente cuando se le pide el valor del sensor.

De esta manera, se puede apreciar como hay diferentes conceptos involucrados en el modelo de este tipo de aplicaciones profundizando en particular en las características de sensado.

Los conceptos presentados en [Fortier et al, 2007] y [Fortier et al, 2010] son usados de base por [Musi, 2016] para proponer un modelo general de gestos desacoplado del mecanismo de sensado, permitiendo detectar gestos y por otro lado dándoles significado.

En [Musi, 2016] el "*contexto*" sobre el cual hace foco es el movimiento/gesticulación del usuario, mientras que en esta tesina se focaliza en la posición del mismo. Sin embargo, es interesante mostrar otras características de modelados relacionadas al contexto, porque a futuro los conceptos presentados en esta tesina se podrían ampliar para incorporar otros sensores además de aquellos para determinar la posición.

En la Figura 2.3.6 se puede ver el modelo propuesto en [Musi, 2016], se puede observar que dicha figura está dividida en dos, aquellas características relacionas al contexto, donde muchas clases usadas de base ya se mostraron en las Figuras 2.3.4 y 2.3.5; y por otro lado todo lo relacionado con los gestos y la interpretación que se les da a cada uno de ellos.

Cabe mencionar que en [Musi, 2016] sólo se explora el uso del Kinect como mecanismo de sensado de movimientos.

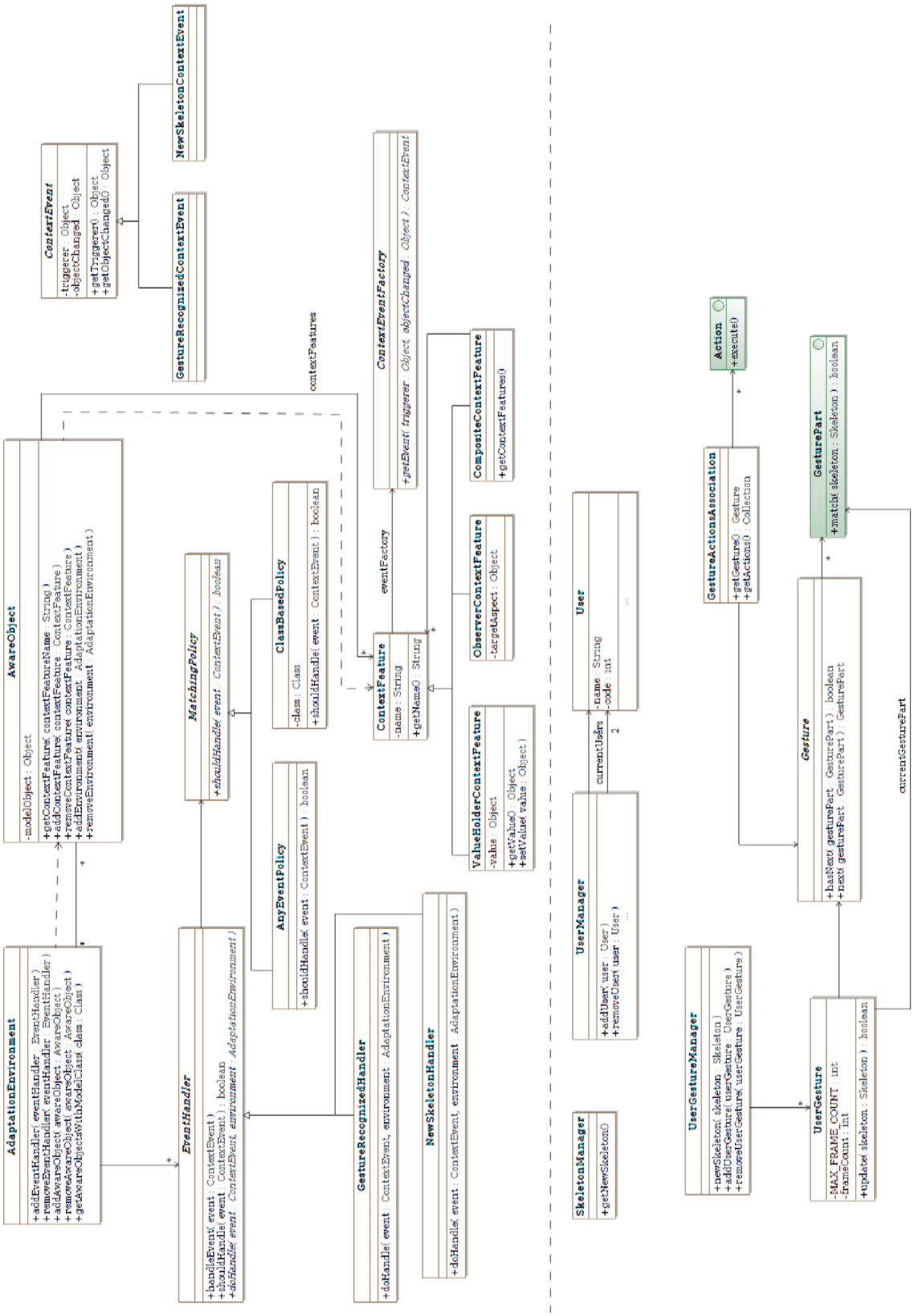

Figura 2.3.6: Combinación del Modelo de Gestos con la Capa de Contexto [Musi, 2016].

- **Framework KAILOS [Dongsoo et al., 2014] [Suk-Hoon et al., 2017]**

Un framework para posicionamiento indoor-outdoor es propuesto en [Dongsoo et al., 2014], el cual se denomina KAILOS. Este framework está pensado para usarse de manera global en edificios de todo el mundo.

Para brindar el servicio de posicionamiento KAILOS describe tres componentes: mapas indoor y outdoor (o de radio) denominado KAI-Map, un sistema de posicionamiento híbrido (denominado KAI-Pos) y un sistema integrado de navegación outdoor/indoor (KAI-Navi). En la Figura 2.3.7 se pueden apreciar estos tres componentes y como cada uno de estos, se usa para lograr brindar aplicaciones al usuario, esto se puede observar en lo que los autores denominan *Positioning Technology Stack.*

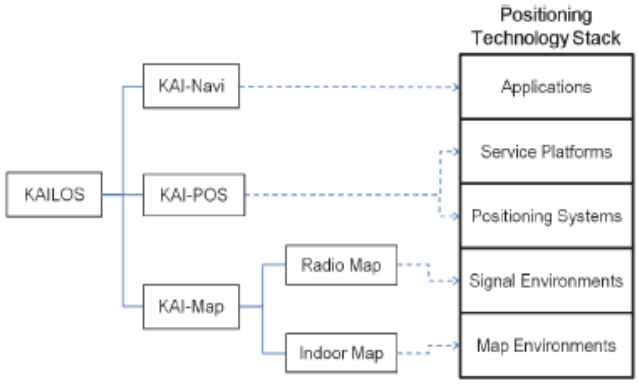

Figura 2.3.7: Componentes del framework KAILOS [Dongsoo et al., 2014].

En la Figura 2.3.8 se pueden apreciar más nivel de detalle de la estructura del componente denominado KAI-Pos. Se puede observar que la posición llega de diferentes canales y luego se procesa para generar una posición híbrida y así brindar más nivel de precisión. Se puede observar que en todos los casos, cada señal que se recibe es tratada para estimar el error que puede tener la misma, y esto es utilizado para luego poder calcular la posición híbrida.

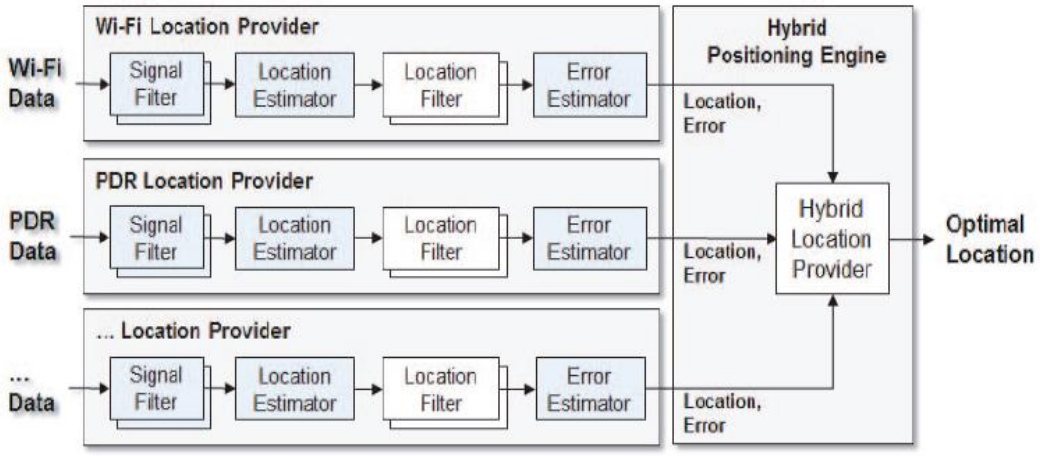

Figura 2.3.8: Estructura de KAI-Pos [Dongsoo et al., 2014].

Los autores siguieron evolucionado el trabajo presentado en [Dongsoo et al., 2014], y presentaron en [Suk-Hoon et al., 2017] dos prototipos, uno es un sistema de navegación indoor/outdoor realizado para el centro comercial COEX en 2014, y el otro prototipo realizado en el 2015 en el campus KAIST.

En los prototipos presentados en [Suk-Hoon et al., 2017] el mecanismo de sensado de posicionamiento outdoor es GPS, mientras que para el posicionamiento indoor usan WI-FI. Esta solución propuesta por los autores está focalizada en la implementación y no se describe a nivel de modelado como está diseñada la misma, sólo se menciona más nivel de detalle de los componentes generales del framework (como son KAI-Pos, KAI-Navi y KAI-Map).

En la Figura 2.3.9 se puede apreciar más detalles de los componentes, sin embargo son conceptos generales y no se profundiza en el diseño de los conceptos involucrados en el framework: KAI-Pos, KAI-Navi y KAI-Map. En dicha figura se puede apreciar como estos componentes interactúan entre sí. Se puede observar que en particular para el posicionamiento híbrido se cuenta con un algoritmo que lo determina, esto es usado tanto por el posicionamiento indoor como outdoor.

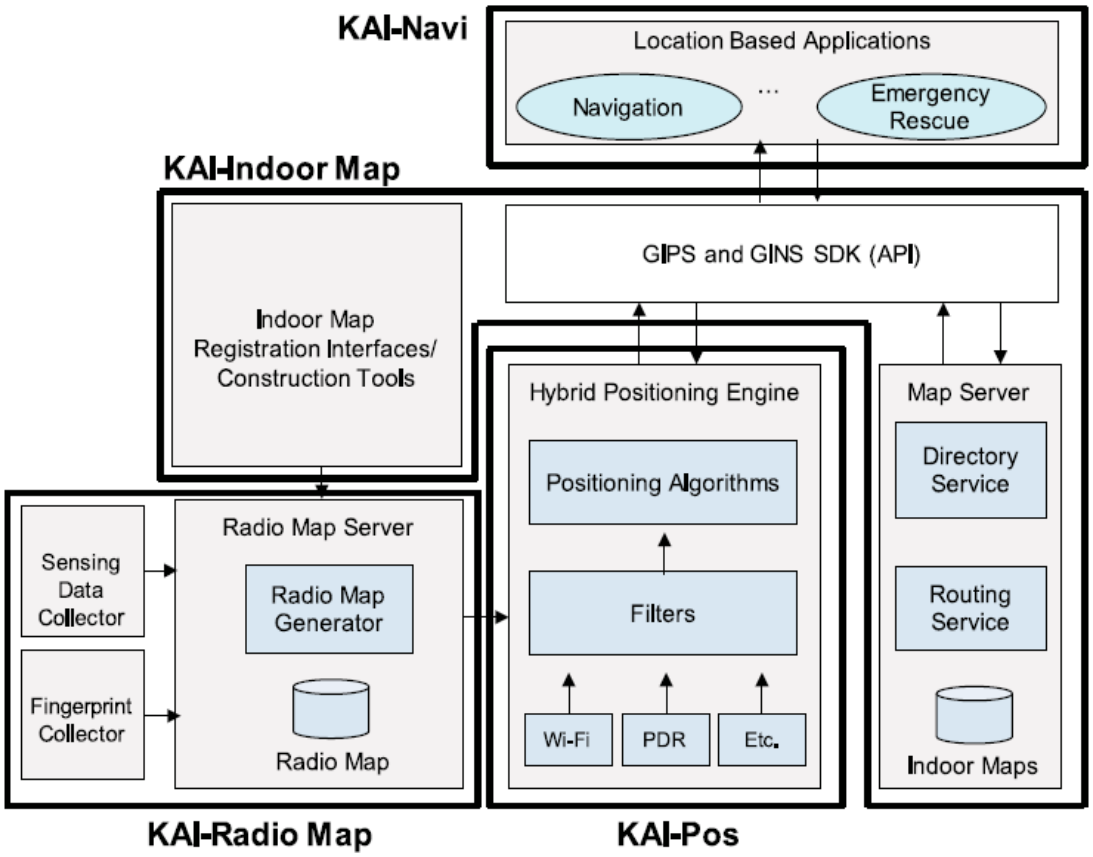

Figura 2.3.9: Detalle de los componentes del framework KAILOS [Suk-Hoon et al, 2017].

- **Framework conceptual para diseñar Aplicaciones Móviles basadas en Posicionamiento [Challiol et al., 2017]**

  En [Challiol et al., 2017] se propone un framework conceptual para diseñar Aplicaciones Móviles basadas en Posicionamiento, se hace hincapié en el concepto de separación de concern para desacoplar cada concepto involucrado, En particular, se menciona la necesidad de desacoplar y lograr independencia de los mecanismos de sensado, así la incorporación dinámica de uno de ellos impacta lo menos posible en la evolución de la aplicación. El framework busca lograr una registración genérica a los mecanismos de sensado para que estos puedan evolucionar en el tiempo. Cabe mencionar que este trabajo no propone un modelo de solución particular sino que detalla conceptos deseables en el diseño de este tipo de aplicaciones.

  En la Figura 2.3.10 se puede apreciar el framework conceptual propuesto en [Challiol et al., 2017].

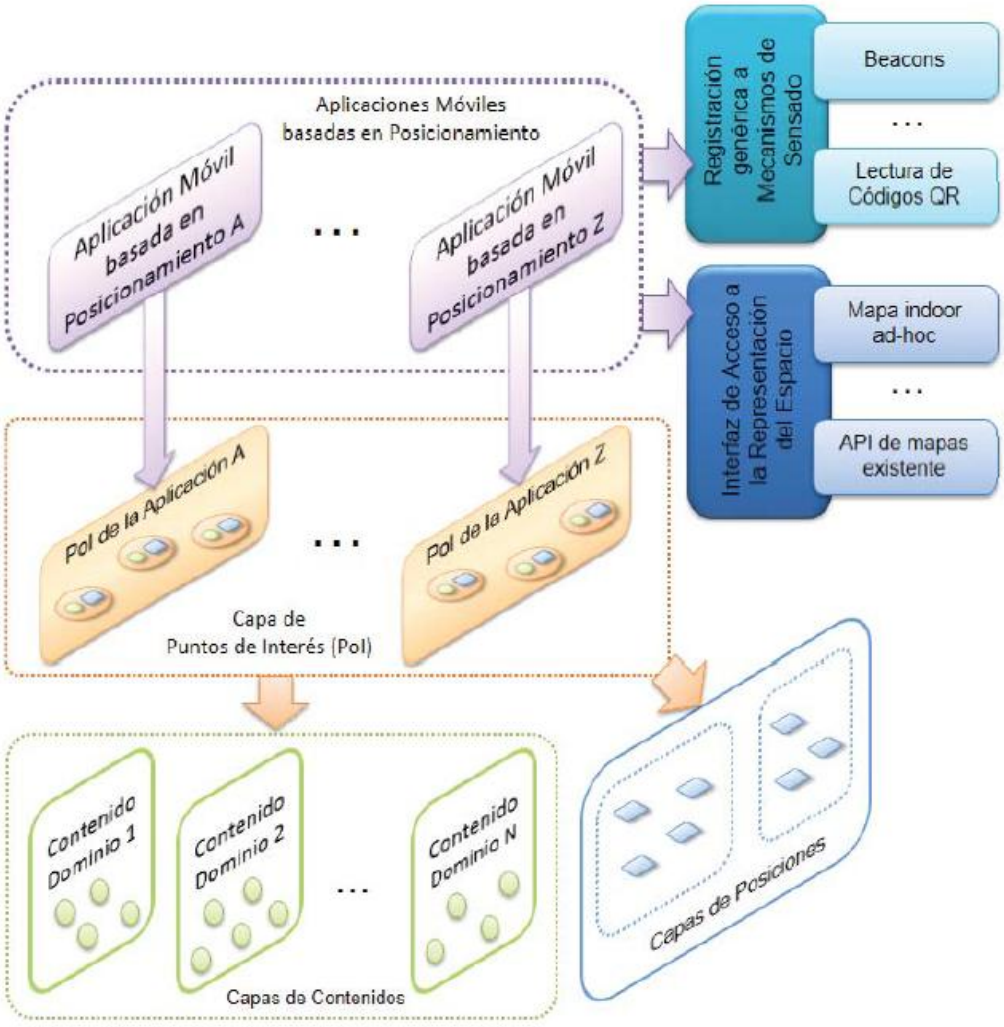

Figura 2.3.10: Framework conceptual para el diseño de Aplicaciones móviles basadas en Posicionamiento [Challiol et al., 2017].

3. Framework Propuesto

En este capítulo se describirá primero la problemática que se busca dar solución, y luego se propondrá una solución de modelado para la misma. Se mostrará cómo funcionan algunas características relevantes del modelo propuesto.

3.1 Descripción de la problemática

Como se mencionó en la Introducción, uno de los objetivos de esta tesina es proponer una solución de modelado que permita brindar soporte a aquellas aplicaciones móviles basadas en posicionamiento, en las cuales se tiene más de un mecanismo de sensado de posición funcionado simultáneamente. Es decir, la posición del usuario es obtenida usando, por ejemplo, GPS y códigos QR.

Se busca que la solución sea genérica e independiente del dominio de la aplicación, esto permitiría que la misma pueda ser usada para la construcción de aplicaciones de este tipo.

Para poder contar con un framework de construcción que tenga la funcionalidad necesaria para agilizar la construcción de este tipo de aplicaciones, se deben identificar aquellas características presentes en este tipo de aplicaciones en relación a los mecanismos de sensado de posicionamiento.

A continuación se mencionan distintas características identificadas a partir del análisis bibliográfico realizado:

- Dado que se quiere representar el posicionamiento del usuario en este tipo de aplicaciones este concepto ya debería estar definido en el framework, para agilizar la tarea del desarrollador, estos se podría abordar con la clase *ContextFeature* presentada en la Figura 2.3.4 (de la Sección 2.3).
- Se deben poder definir los sensores, y estos poderlos tratar de manera polimórfica entre ellos. Como era el caso de la clase *Sensor* presentada en la Figura 2.3.2 y Figura 2.3.5 (de la Sección 2.3).
- Se debe estar escuchando por nuevos valores de sensado de posicionamiento y reaccionar en base a estos cambios. En algunos casos, una vez que llegan estos valores, se debe analizar si los valores sensados:
    - requieren un procesamiento o ajuste para ser útiles para cada aplicación.
    - deben propagase o no, esto depende de si es relevante el cambio para cada aplicación.

  Es decir, cada aplicación debería poder definir si el valor necesita una transformación y si esto debe ser propagado o no. Ésto guarda relación con los conceptos de *Transformation* y *Dispatcher* descriptos en la Figura 2.3.5 (de la Sección 2.3).

Contar con un mecanismo de sensado combinado, requiere que la posición muchas veces se trate de manera hibrida o se deba hacer un procesamiento, como era abordado en la Figura 2.3.8 (de la Sección 2.3). Es decir, poder lograr mecanismos de posicionamiento combinados requiere definir como estos se van a relacionar, para poder así lograr mayor nivel de precisión en la posición.

Se puede apreciar que algunos conceptos ya se encuentran modelados y otros abordados de forma conceptual. Acorde a esto, y dado las ventajas de contar con la posibilidad de variabilidad en tiempo de ejecución, se tomará de base los conceptos presentados en [Fortier et al, 2007] y [Fortier et al, 2010], y mostrados en particular en las Figuras 2.3.4 y 2.3.5 de la Sección 2.3.

Si bien se toma de base los trabajos [Fortier et al, 2007] y [Fortier et al, 2010], estos no cuentan con detalles de sensores concretos algo que se propondrá como parte del framework. El framework contara con tres sensores ya definidos:

- *GPS*
- *Lector de códigos QR*
- *Sensores del estilo Beacons* (en particular el protocolo definido por *AltBeacon*[11])

Con el framework que se propondrá en la siguiente sección se espera que el desarrollador de este tipo de aplicaciones se focalice en los aspectos de dominio, heredando de nuestro framework aquellos aspectos relacionados con el sensado de posicionamiento. Es decir, aprovechando los puntos de extensión para la construcción de aplicaciones móviles basadas en posicionamiento orientadas a dominios específicos, por ejemplo turismo o educación.

3.2 Descripción del framework propuesto

Para el diseño del framework fueron de gran aporte la participación del autor de la tesina en diferentes proyectos a fines con mecanismos de sensado de posición, los cuales se detallan en el Anexo A y B. Acorde a este aprendizaje y a la bibliografía investigada es que se realiza el planteo del framewok.

Como se mencionó en la sección anterior, se tomarán de base los conceptos presentados en [Fortier et al, 2007] y [Fortier et al, 2010], y mostrados en particular en las Figuras 2.3.4 y 2.3.5 de la Sección 2.3.

En la Figura 3.1 se puede apreciar la clase *AbstractsApplications*, esta clase es la encargada de conocer todas aquellas características de los mecanismos de sensados (clase *SensingConcern* tomada de base de la Figura 2.3.5 de la Sección 2.3). Al ser una clase abstracta la misma no se puede instanciar, dicha clase en el framework fue creada para poder subclasificar de esta, y lograr así, agilizar la construcción de este tipo de aplicaciones.

[11] Página de librería *AltBeacon*: https://altbeacon.github.io/android-beacon-library/index.html (Último acceso 12/12/17)

En la Figura 3.1 se definió una subclase concreta denominada *AppSingleUser* esta clase es provista por el framework para poder ser instanciada si lo único necesario es contar con características de contexto del usuario. Esta clase ya conoce un usuario (clase *User*), el cual conoce sus características de contexto (clase *ContextFeatures* tomada de base de la Figura 2.3.4 de la Sección 2.3). Notar que las clases en gris fueron tomadas de base de modelos existentes como se mencionó anteriormente.

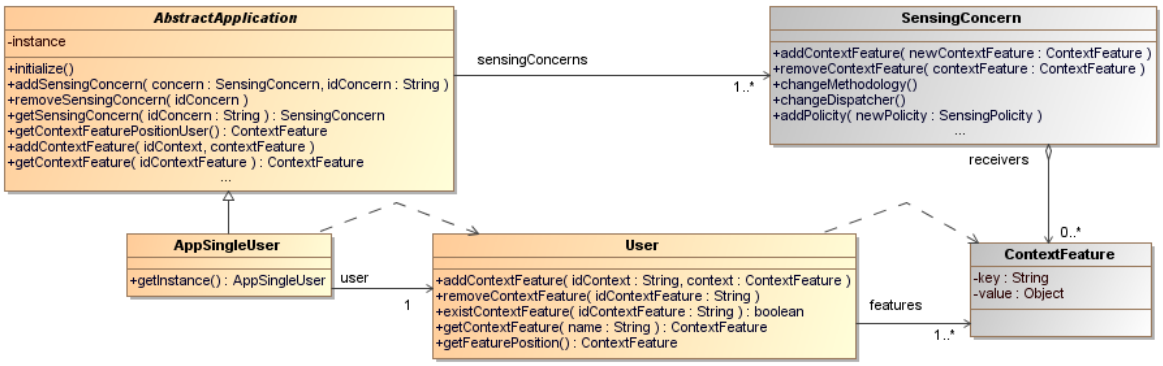

Figura 3.1: Representación general de las aplicaciones.

En la Figura 3.1 se puede observar que la clase *AppSingleUser* observa al usuario y a su vez esté observa sus características de contexto, las cuales serán modificadas por la clase *SensingConcern.* Cada vez que cambia una característica de contexto, el usuario recibe el aviso del cambio, y propaga a quien este observando, en este caso la clase *AppSingleUser.* Esto respeta el patrón de diseño *Observer* [Gamma et al., 1995].

En este tipo de aplicaciones, la representación de la posición juega un rol fundamental, como se menciona en [Leonhardt, 1998] las posiciones se pueden representar de diferentes manera (*Simbólicas* o *Geometricas*), acorde a esto se decidió representar la posición como una interfaz. Esto se puede apreciar en la Figura 3.2, donde además se pueden observar clases concretas que implementan dicha interfaz.

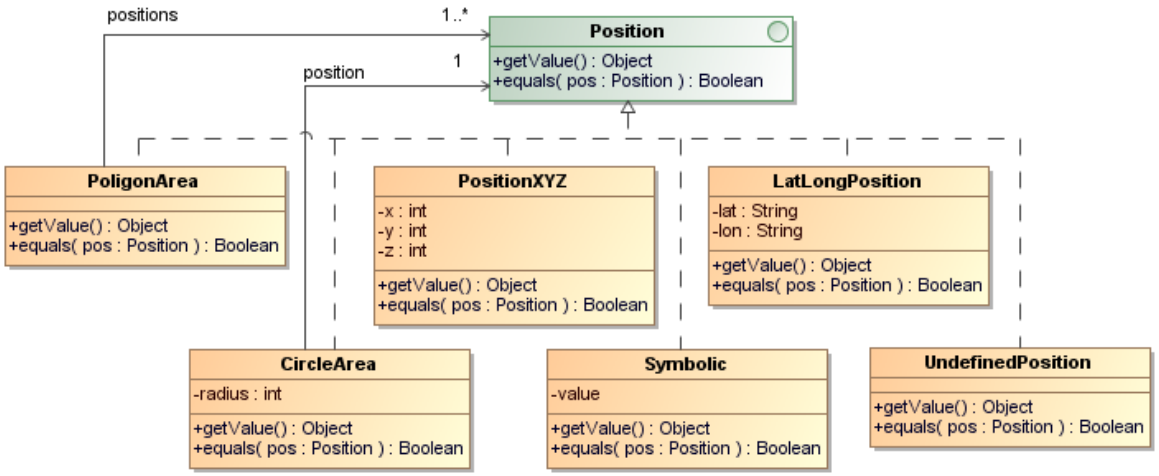

Figura 3.2: Representaciones de las posiciones.

Veamos qué representan cada una de las clases de la Figura 3.2:

- La clase *PoligonArea*, representa un área definida con un polígono, esto podría servir para representar tanto el contorno de un edificio como el de una sala dentro del mismo. Como se puede apreciar en la Figura 3.2, esta clase conoce diferentes posiciones las cuales delimitan el polígono.
- La clase *CircleArea*, representa un área circular delimitada por un punto central (posición) y el radio a partir de dicho punto. Muchas veces este tipo de representaciones es útil para representar cercanía a una posición puntual.
- La clase *PositionXYZ,* representa una posición relativa, donde (X,Y) determina las coordenadas en el plano, y el Z se usa para la altura. Esto es de utilidad para indicar posiciones relativas dentro de un edificio.
- La clase *Symbolic*, representa una posición simbólica, esta se describe con un valor. Este tipo de representaciones se usa para indicar posiciones del estilo "*Edificio de la Facultad de Informática*", donde son descriptivas del lugar. También se pueden usar para describir posiciones relacionadas a códigos QR, donde el valor de dicho código se considera simbólico.
- La clase *LatLongPosition*, representa una posición absoluta representada en relación a la latitud y longitud, muchas veces este tipo de representaciones se asocia al mecanismo de sensado GPS.
- La clase *UndefinedPosition*, representa la "*no posición*" esto respeta el concepto del patrón de diseño *Null Object* [Woolf, 1997]. Esta representación se utiliza cuando no es posible determinar una posición, por ejemplo, porque no hay funcionando ningún mecanismo de sensado.

Una vez que tenemos representadas las posiciones, veamos aquellos detalles relacionados con las características de los mecanismos de sensado.

A continuación se describirá más nivel de detalle de la clase *SensingConcern,* esta cuenta con información ya mencionada al ser presentada en la Figura 2.3.5 (de la Sección 2.3). Las clases que se representan en este framework asociadas a esta clase se pueden apreciar en la Figura 3.3 y en gris las clases tomadas de base (*Dispatcher* y la jerarquía de *SensingPolicy*).

Algunos conceptos que se decidieron ampliar/modificar para este framework son aquellas características relacionadas a la transformación del valor sensado (clases *ValueSensorTransformer* y *TransformMethodolody*), y cómo se representan los sensores (interfaz *Sensor*). Estas clases también se pueden observar en la Figura 3.3, luego se ampliarán cada una de estas.

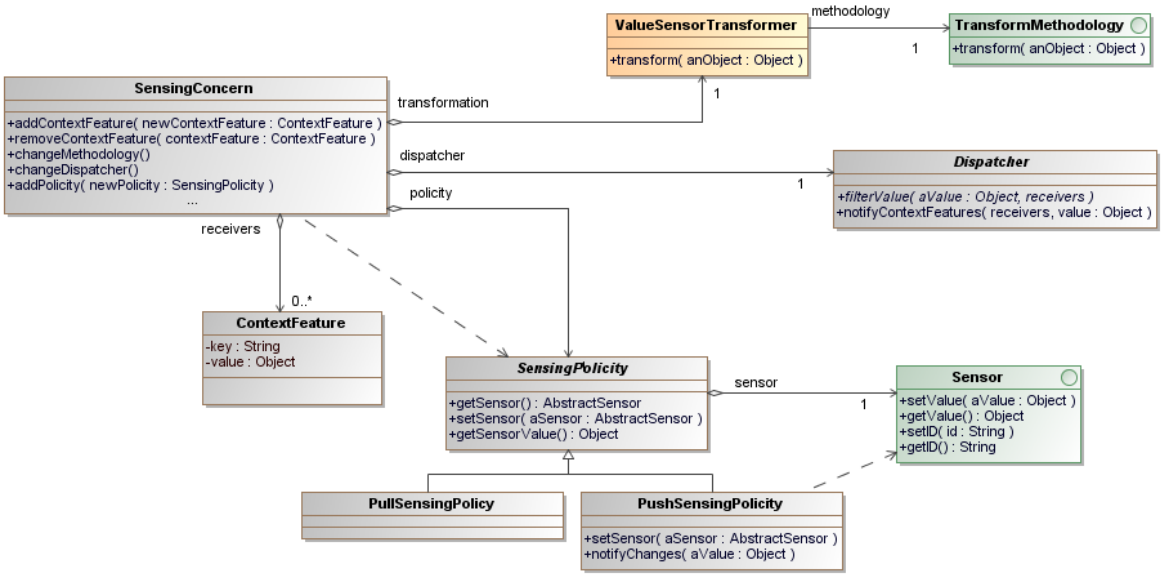

Figura 3.3: Clases relacionadas a la clase *SensingConcern*.

Veamos con más nivel de detalle la trasformación del valor sensado. Se decidió diseñar la clase *ValueSensorTransformer*, para poder tener desacoplada la metodología usada para la transformación en sí (interfaz *TransformMetodology*), como se puede observar en la Figura 3.4. Ésto se debe a que podría ocurrir que la trasformación en sí requiriera un procesamiento más complejo, en ese caso, se podría extender de la clase *ValueSensorTransformer* para involucrar otra lógica más compleja y no sólo una transformación de valores como es abordada por la interfaz *TransformMetodology*.

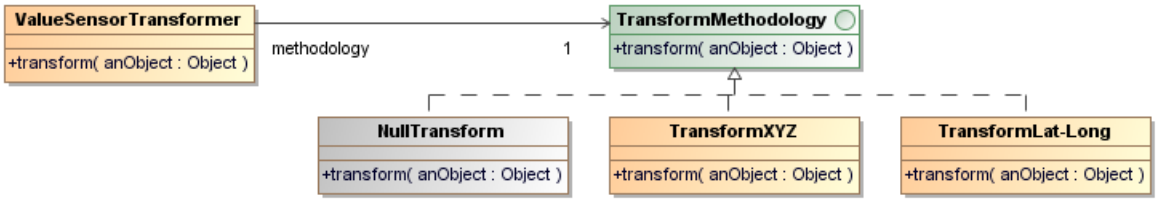

Figura 3.4: Clases relacionadas a la transformación de los valores sensados.

Se puede observar en la Figura 3.4 que se crearon tres clases concretas que cumplen la interfaz *TransformMetodology:*

- La clase *TansformLat-Long,* permite transformar el valor sensado en un valor (latitud, longitud).
- La clase *TransfromXYZ,* permite transformar el valor sensado en un valor (X,Y,Z).
- La clase *NullTransform,* permite representar la no transformación, es decir, el valor sensado no requiere ningún tipo de trasformación particular. Este concepto ya estaba definido en el modelo tomado de base como se mostró en la Figura 2.3.5 (de la Sección 2.3).

Estas tres clases descriptas anteriormente, son posibles transformaciones, sin embargo esto es un punto de extensión del framework propuesto, con lo cual podría crearse otro tipo de transformaciones dependiendo del dominio de aplicación.

Otra característica relacionada al valor sensado, es si este debe propagarse o no, la clase encargada de esto es el *Dispatcher*. Al ser esta clase abstracta se debe contar con clases concretas que definan la estrategia por la cual se decide o no propagar. Esto podría ser visto como el patrón de diseño *Strategy* [Gamma et al., 1995] dado que se tienen diferentes estrategias para decidir si el valor es propagado o no.

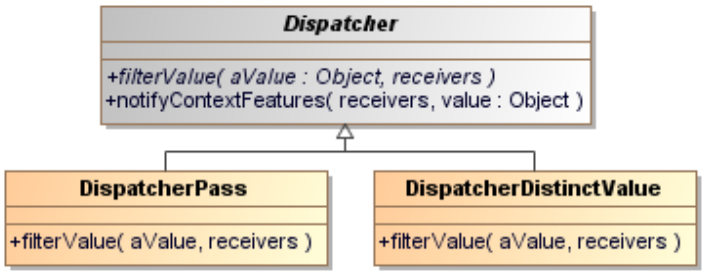

Figura 3.5: Clases relacionadas a la propagación del valor sensado.

Se puede observar en la Figura 3.5 que se definieron dos subclases de *Dispatcher*. La clase *DispatcherPass* propaga cualquier valor sensado, mientras que la clase *DispatcherDistinctValue* sólo propaga el valor sensado siempre y cuando este sea distinto del valor actual que tiene la característica de contexto (*ContextFeature*). Éstas son dos posibles estrategias de propagación, siendo un punto de extensión del framework propuesto.

Otra clase que se mencionó que cambiaba respecto al modelo usado de base es la representación del sensor, que en este framework se representa con una interfaz denominada *Sensor*, para nuestro framework tiene que tener una forma de identificación y además tiene que poder brindar el último valor sensado.

Se definió la clase *AbstractSensor* que implementa la interfaz sensor, y le da comportamiento concreto tanto para devolver un identificador como el valor sensado. Cabe mencionar que cada vez que se invoca el método `setValue(anObject)` dentro del mismo se invoca la notificación a cualquier clase que está observando a un sensor. Esto es usado por la política de sensado push, para propagar el cambio del sensor.

Los conceptos mencionados se pueden apreciar en la Figura 3.6. Además, en dicha Figura se pueden apreciar los diferentes sensores concretos propuestos por el framework.

La jerarquía de sensores también es un punto de extensión del framework. Podría ocurrir que otras clases implementen la interfaz *Sensor* sin ser estas subclases de la jerarquía de *AbstractSensor*.

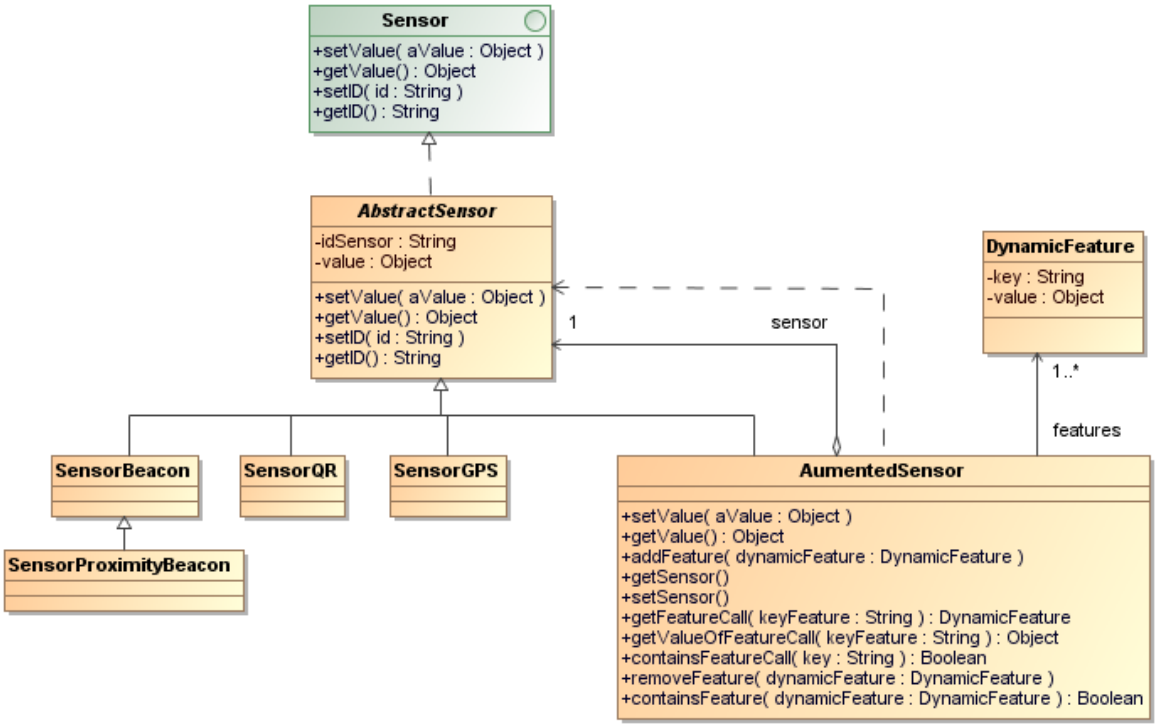

Figura 3.6: Representación de los diferentes sensores.

Veamos qué representan cada uno de los sensores diseñados en la Figura 3.6:

- La clase *SensorBeacon*, representa la abstracción de un *Beacon*. La clase *SensorProximityBeacons* representa, en particular, la abstracción de un *Beacon* de proximidad. Como se mencionó en la Sección 2.1 puede haber otros tipos de *Beacons*. Este tipo de *Beacons* sensa valores del estilo *1,2,3* los cuales son indicadores de distancias.
- La clase *SensorQR*, representa la abstracción de un lector de QR, su valor contendrá generalmente posiciones *Symbolic*.
- La clase *SensorGPS*, representa al sensor de GPS, su valor contendrá posiciones *LatLongPosition*.
- La clase *AumentedSensor*, nos permite la flexibilidad de aumentar un sensor con mayor información, esto podría ser visto como el patrón de diseño *Wrapper* (o *Decorator*) [Gamma et al., 1995]. Para representar las características se modelo la clase D*ynamicFeature* que permite representar a cualquier característica con su valor. De esta forma uno podría tener un *SensorBeacon* que está aumentado por ejemplo con una posición *LatLongPosition* para saber en qué lugar se encuentra el mismo. Estos valores aumentados sirven en mayor parte para facilitar la transformación del valor sensado en un valor útil para la aplicación, facilitando así el posicionamiento del usuario. Además, la clase *AumentedSensor* observa al sensor que conoce para poder probar cualquier cambio del mismo.

Estas representaciones de los sensores son generales y no son dependientes de las plataformas o API de sensores físicos concretos. Esto es posible ya que se creó una clase que permite desacoplar aquellas características propias de los sensores físicos, para lo cual se diseñó la clase *ManagerSensorListener*. Esta clase contiene todos los sensores junto a sus identificadores (atributo `associationStringID-Sensor`). Cada vez que se detecta que hubo un cambio en un sensor físico, este cambio es propagado a la abstracción de este sensor, la cual cumple la interfaz *Sensor* (de la Figura 3.6). Esto permite que la jerarquía de clases de *AbstractSensor* sean representaciones generales independientes de los sensores físicos concretos.

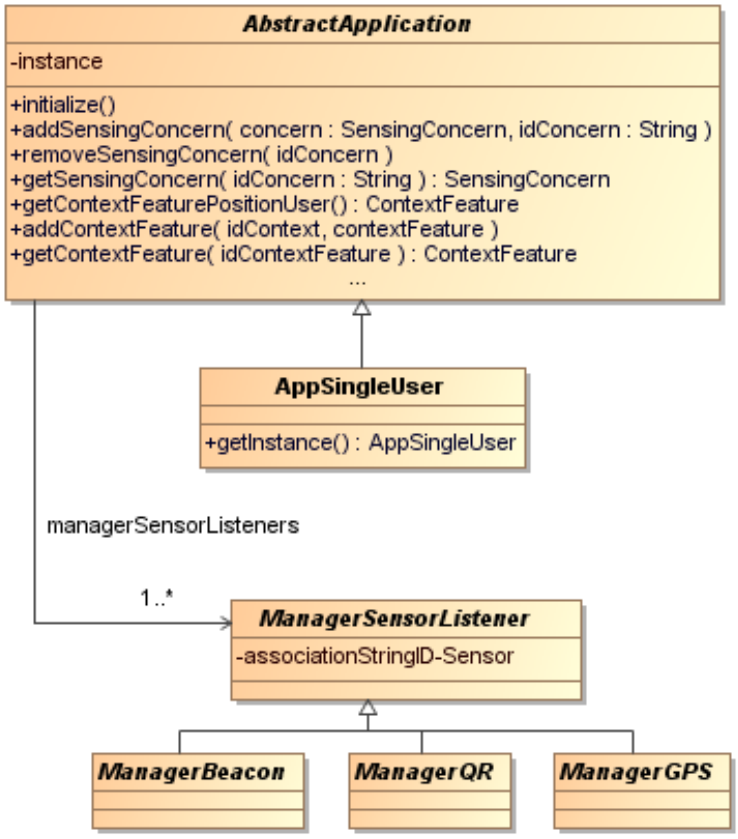

Figura 3.7: Manejadores de sensores.

En la Figura 3.7 se puede observar que se crearon tres subclases, abstractas *ManagerBeacon*, *ManagerQR* y *ManagerGPS* esto se diseñó para facilitar así la extensión del framework con manejadores de tipos de sensores concretos, ya que cada uno de ellos funciona de manera distinta. En el Capítulo 4 se mostrarán tres extensiones concretas de manejadores.

En la Figura 3.8 se puede apreciar el framework propuesto en esta tesina para el manejo de múltiples sensores de posicionamiento.

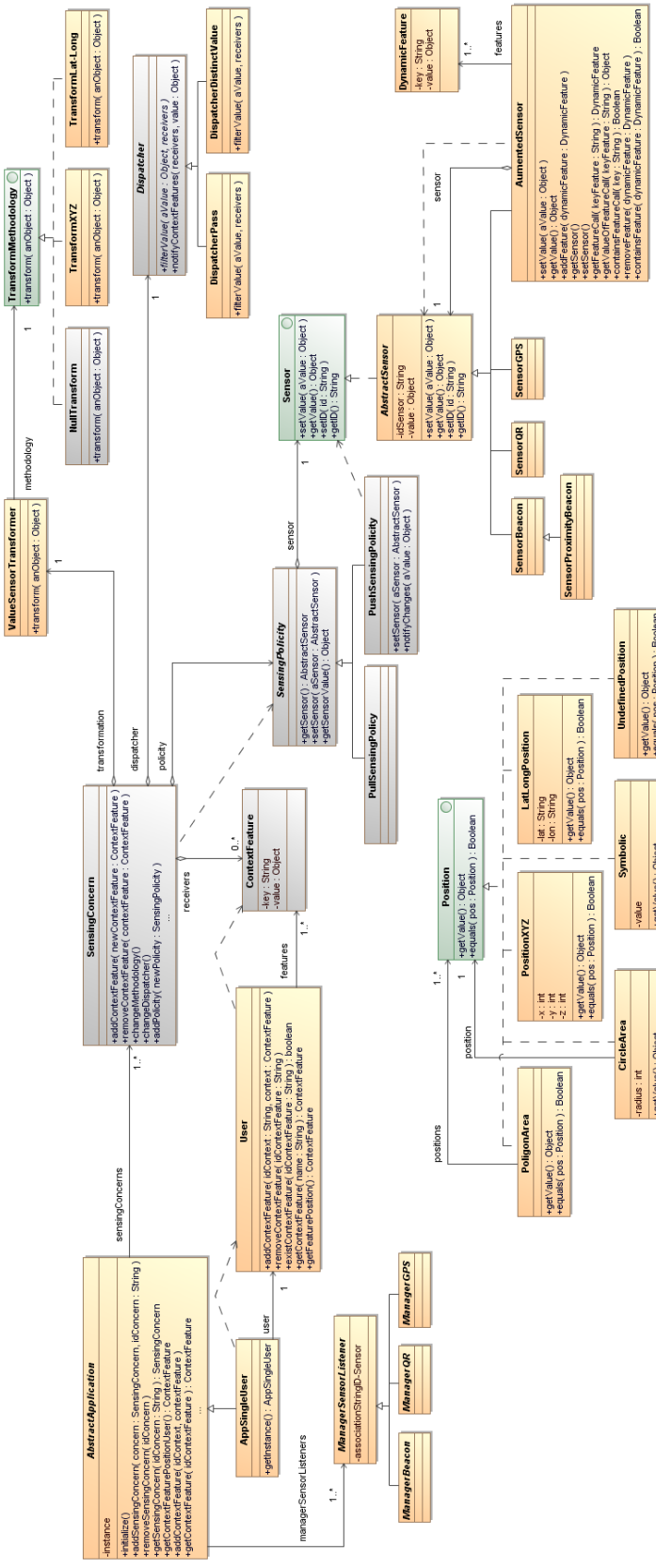

Figura 3.8: Framework propuesto.

3.3 Funcionamiento del framework propuesto

En esta sección se mostrará el funcionamiento del framework propuesto en la Sección 3.2, haciendo hincapié en particular en el funcionamiento de los sensores y cómo el valor de los mismos se propaga.

Para poder entender el funcionamiento se presenta primero un diagrama de instancias con características de contextos definidas y múltiples sensores.

Supongamos que se desea contar con tres sensores (clases concretas que implementan la interfaz *Sensor*): QR, *Beacon* y GPS. Además, que tanto el sensor de *QR* como de *Beacon* conoscan su posición (x,y,z), para aumentar el sensor con esta información se usar la clase *AumentedSensor* con una característica dinámica con información de la posición (x,y,z) correspondiente. Por lo tanto, para estos sensores mencionados se usan las siguientes clases combinadas de la siguiente manera:

- *Una instancia de AumentedSensor* que conoce:
    - Una instancia de *SensorQR*
    - Una instancia de *DinamicFeature* cuyo valor es una instancia de *PositionXYZ* (que expresa donde está posicionado el código QR)
- *Una instancia de AumentedSensor* que conoce:
    - Una instancia de *SensorBeacon*
    - Una instancia de *DinamicFeature* cuyo valor es una instancia de *PositionXYZ* (que expresa donde está posicionado el *Beacon*)
- *Una instancia de SensorGPS*

Por otro lado, se desea contar con tres características de contexto, cada una de ellas contendrá el valor sensado por cada mecanismo definido anteriormente. Es decir, una de ellas guardará la posición tomada del GPS, mientras que las otras dos guardarán *PositionXYZ*. Cada vez que se recibe un valor del QR y *Beacons* este es transformado usando la clase *TransformerXYZ*. De esta manera, se puede tener conviviendo de una manera simple los tres mecanismos de sensado.

En la Figura 3.9 se puede observar la instanciación mencionada y cómo se relacionan cada uno de los conceptos descriptos que se querían considerar. Recordemos que la clase *SensingConcern* es la que agrupa tanto la política de sensado (que en este caso todas se instanciaron con una política *push*), la transformación que se desea realizar respecto del valor sensado y el mecanismo para decidir propagar el valor sensado o no (*Dispatcher*).

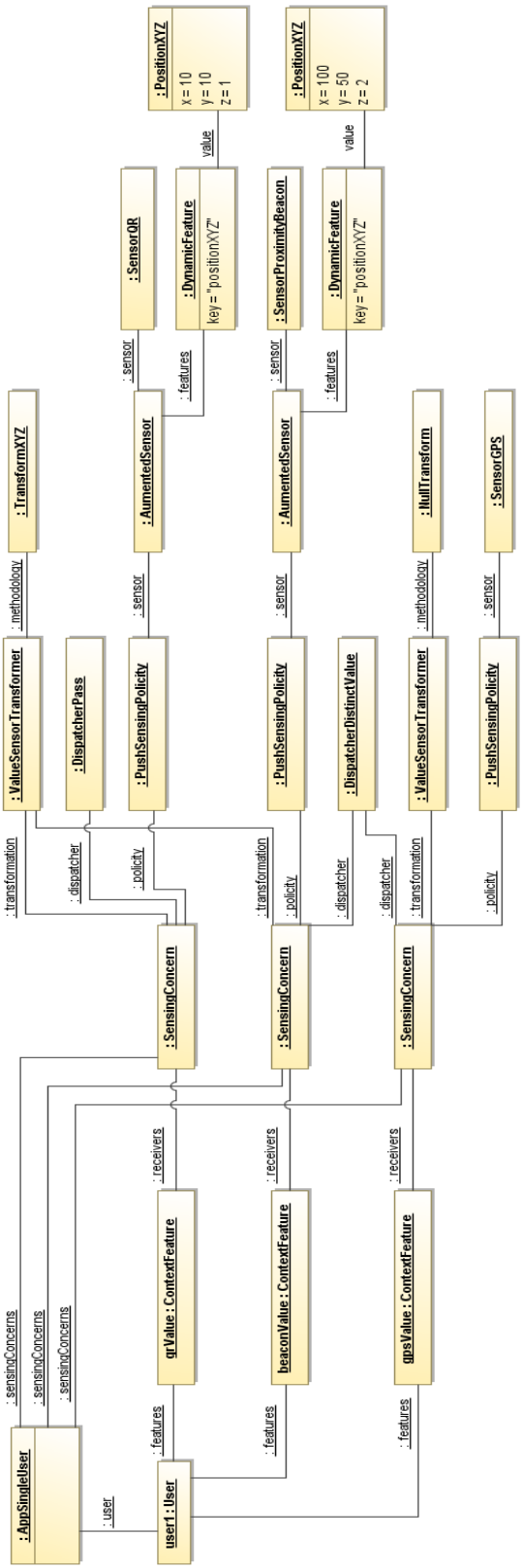

Figura 3.9: Diagrama de instancias con tres características de contexto y tres mecanismos de sensado.

Se puede apreciar en la Figura 3.9 que se instanció un *ValueSensorTransformer* (que conoce a un *TransformerXYZ*), y este es usado tanto para el sensor de QR como de *Beacon* porque su comportamiento es general, no importa el valor sensado, busca dentro de las características dinámicas la posición (x,y,z) de dicho sensor, para esto como puede apreciarse ambos sensores están aumentados con esta información. En el caso del GPS se instancia con un *NullTransformer*, dado que se no se aplicará ninguna transformación al valor sensado por el GPS.

Respecto a los *Dispatcher*, se puede apreciar en la Figura 3.9 que se instanció un *DispatcherPass* para el sensor de QR, es decir, siempre se propaga. Significa que cada vez que un usuario lee un código QR este se propaga hasta la característica de contexto. Recordemos que en este caso, la lectura de QR se usa para posicionamiento, esto funciona al estilo de las indicaciones "*Usted está aquí*". Se podría contar con otra característica de contexto adicional que guarde que significa estar en ese lugar, por si el usuario requiere saber más información y no solamente posicionarlo en el mapa.

En el caso de los sensores de *Beacon* y GPS se instanciaron con un *DispatcherDistinctValue*, es decir, si el valor sensado es igual que el valor anterior que tenía la característica de contexto, este no se propaga.

Tanto los *Transformer* como los *Dispatcher* están definidos de manera genérica, permite que sean reusados, y de esta manera se evita tener múltiples instancias de la misma clase.

En la Figura 3.10 se puede apreciar la secuencia generada a partir de la lectura de un nuevo código QR, por simplicidad, se pasa directamente a cambiar el valor del *SensorQR*, de ahí se empieza a dar aviso del cambio hasta que llega la *SensingConcern*, el cual primero delega en la transformación correspondiente, en este caso se busca la posición (x,y,z) del sensor aumentado. Luego, se delega al *dispatcher* que en este caso propaga el valor, llegando así el mismo a la característica de contexto (*qrValue*).

En la Figura 3.11 se presenta la secuencia generada a partir de un nuevo valor del beacon, por simplicidad, se pasa directamente a cambiar el valor del *SensorProximityBeacon*, de ahí se empieza a dar aviso del cambio hasta que llega la *SensingConcern*, el cual primero delega en la transformación correspondiente, en este caso al igual que para el código QR, se busca la posición (x,y,z) del sensor aumentado. Luego, se delega al *dispatcher* que en este caso propaga siempre y cuando el valor sea distinto del que ya tenía la característica de contexto (*beaconValue*).

En la Figura 3.12 se puede observar la secuencia generada a partir de un nuevo valor del GPS, por simplicidad, se pasa directamente a cambiar el valor del *SensorGPS*, de ahí se empieza a dar aviso del cambio hasta que llega la *SensingConcern*, el cual primero delega en la transformación correspondiente, en este caso, no se realiza ninguna. Luego, se delega al *dispatcher* que en este caso propaga siempre y cuando el valor sea distinto del que ya tenía la característica de contexto (*gpsValue*).

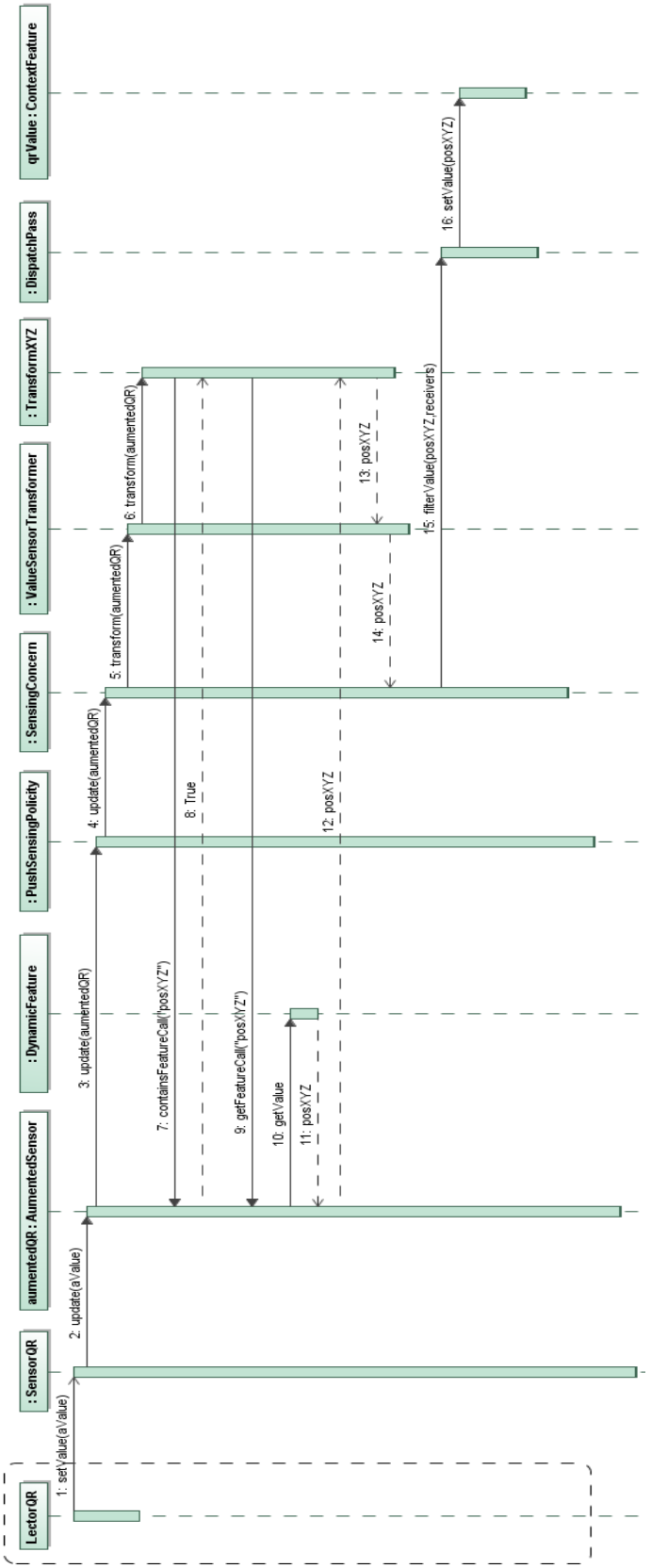

Figura 3.10: Se realiza la lectura de un código QR.

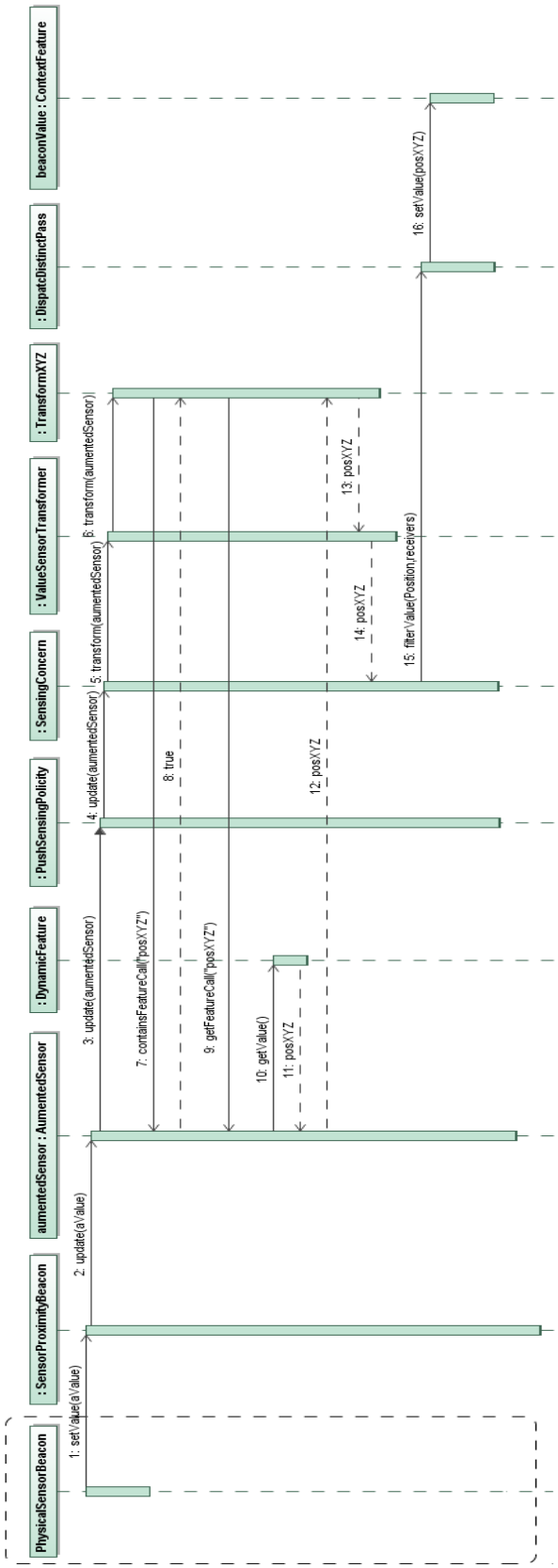

Figura 3.11: El *beacon* emite un nuevo valor.

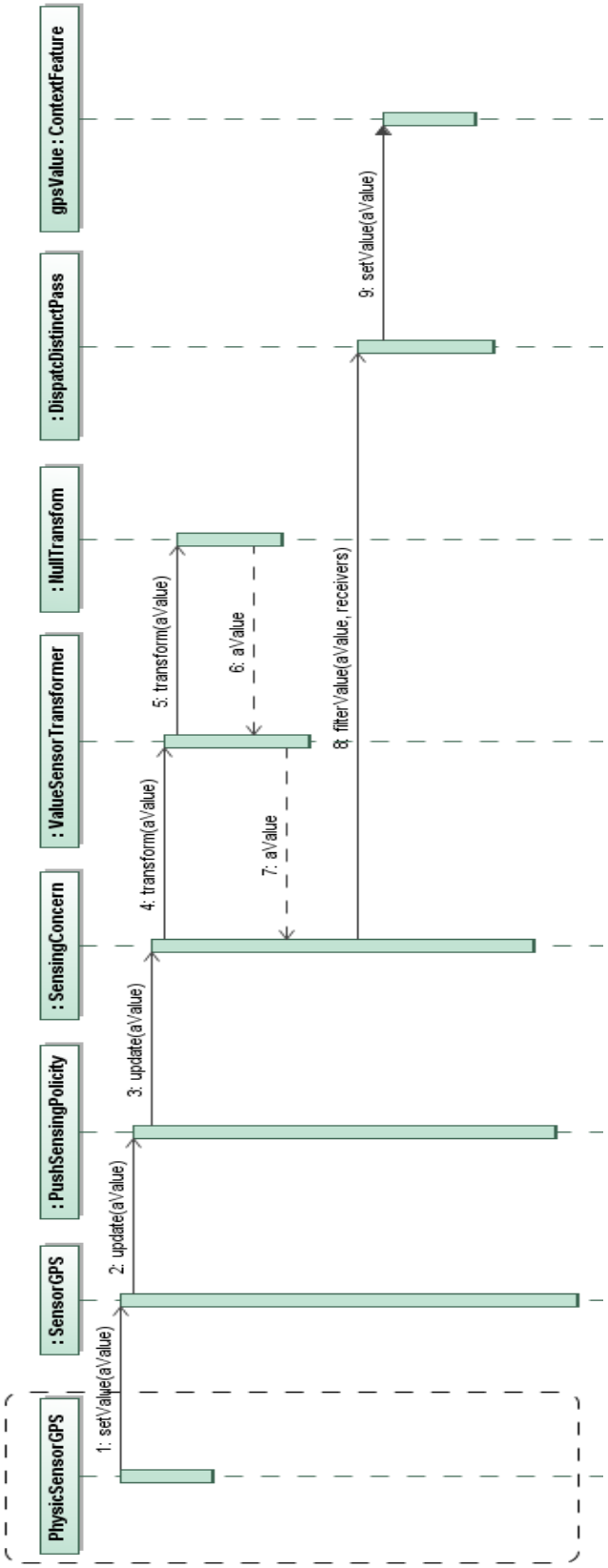

Figura 3.12: El GPS emite un nuevo valor.

De esta manera, en las Figuras 3.10, 3.11 y 3.12 se pudieron observar la secuencia generada por diferentes mecanismos de sensado. También se puede apreciar que la secuencia de ejecución es muy similar, las abstracciones de los sensores (*SensorQR, SensorProximityBeacon y SensorGPS*) reciben el nuevo valor, y este es propagado hasta el *SensingConcern*, el cual primero delega en *trasnformer* (quien transformará o no según corresponda, en el caso de no requerir transformación se tendrá una instancia de *NullTransformer*), y luego delega al *dispatcher*, el cual determina si se propaga el cambio o no a la característica de contexto.

4. Prototipos desarrollados

En este capítulo se presentarán las caracterices generales del desarrollo propuesto, en particular se crearon dos prototipos que permiten apreciar cómo se puede llevar a la práctica los conceptos del framework propuesto en la Sección 3.2.

4.1 Descripción general

Los prototipos se desarrollaron usando *PhoneGap* [PhoneGap], que permite construir aplicaciones móviles híbridas basadas en HTML, CSS y *Javascript*. Al ser híbrido se tienen las ventajas de cualquier aplicación Web pero además se pueden acceder a los sensores internos del dispositivo móvil, esto es fundamental para el tipo de aplicaciones de interés para esta tesina.

En la bibliografía se puede encontrar como *PhoneGap* (distribución libre de *Apache Cordova*[12]), la versión usada para estos desarrollos es la 3.6.3. Más detalles sobre *PhoneGap* pueden apreciarse en el Anexo C.

Para poder pasar el framework propuesto a clases *Java* y que estas puedan ser usadas dentro de un entorno *PhoneGap*, se usó el plugin presentado en [Zimbello and Challiol, 2016], el cual permite invocar desde *Javascript* métodos específicos del framework y así recibir información para poder mostrar en la vistas HTML.

Como entorno de desarrollo de los prototipos se usó *Android Studio*[13], se usaron dos computadoras distintas cada una con versiones distintas del entorno, pero dado que el mismo está planteado de manera general, en este caso, las versiones no generaban conflictos. Las dos versiones usadas son:

- *Android Studio* 3.0
- *Andorid Studio* 2.1.3

Respecto a los proyectos *PhoneGap* desarrollados para cada prototipo, a continuación se listan las principales configuraciones realizadas:

- *Build version tools: 23.0.3*
- *CompileVersion: Android 6.0 (Marshallow) API 23*
- *Min SDK version API: 18 Android 4.3 (Jeally Bean)*
- *Geaddle version: 2.10*
- *Java: JDK 1.7.0_79*[14]

El framework propuesto se implementó como clases *Java* que se empaquetaron en un .jar y el mismo es usado por cada uno de los proyectos *PhoneGap*.

[12] Página de *Apache Cordova*: https://cordova.apache.org (Último Acceso: 16-11-2017)
[13] Página de *Android Studio*: https://developer.android.com/studio/index.html?hl=es-419 (Ultimo acceso: 16/11/2017)
[14] Página de la versión de Java: http://www.oracle.com/technetwork/java/javase/7u79-relnotes-2494161.html (Ultimo acceso: 16/11/2017)

Como se mencionó en la Sección 3.2 por cada sensor que se quiera usar se debe crear su correspondiente *ManagerSensorListener*. Acorde a esto se crearon tres manager específicos uno por cada uno de los mecanismos de sensado que se utilizó (QR, Beacons y GPS). Se puede observar en la Figura 4.1 las clases creadas, *ManagerAltBeacon*, *ManagerPhoneGapPluginQR* y *ManagerAndrOidLocationGPS*.

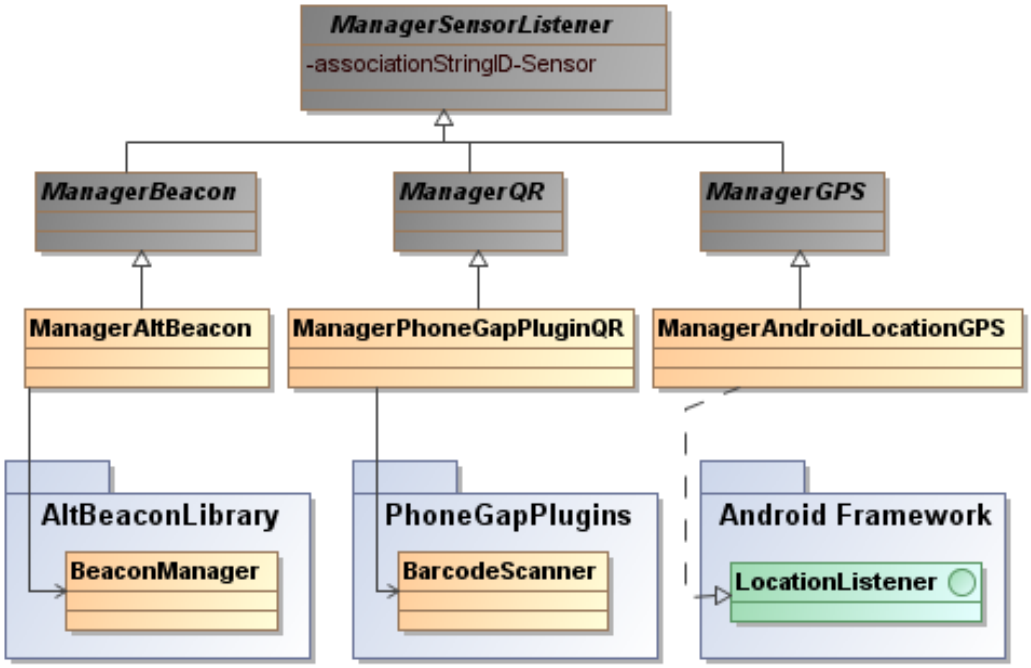

Figura 4.1: Subclases de *ManagerSensorListener*.

Veamos más nivel de detalle de las clases definidas en la Figura 4.1:

- La clase *ManagerAltBeacon* se encarga de coordinar sensores físicos de *Beacons*, en este caso particular, se puede ver que conoce un *BeaconManager* (que pertenece a la librería *AltBeacon*, en particular la versión 2.8[15]). Es decir, este manager es particular para poder trabajar con el protocolo de comunicación *AltBeacon.* Más detalles de otros protocoles de *Beacons* se pueden encontrar en el Anexo E. Si se requiriera usar otro protocolo de comunicación, esto implicaría tener otra subclase nueva de *ManagerBeacon*.

  ManagerAltBeacon está escuchando los sensores *beacons* físicos y cuando recibe una señal verifica a que *SensorBeacon* corresponde y le setea el valor. Esto hace que se inicie la propagación del valor, para que en el caso que corresponda este puede ser propagado hasta la característica de contexto correspondiente.

  Para los prototipos se usaron tres *beacons* físicos y están configurados dentro del *ManagerAltBeacon.* También se cuenta con la asociación entre los *beacons* físicos y los *SensorProximityBeacon* (abstracciones de los *beacons*).

[15] Página de AltBeacon: https://github.com/AltBeacon/android-beacon-library (Ultimo acceso: 16/11/2017)

- La clase *ManagerPhoneGapPluginQR* usa el plugin *BarcodeScanner*[16] para poder realizar la lectura de los códigos QR, y controla si el valor leído está incluido entre los códigos QR definidos para la aplicación, en caso afirmativo, busca el *SensorQR* correspondiente y le setea este valor. Esto hace que se inicie la propagación del valor, para que en el caso que corresponda este puede ser propagado hasta la característica de contexto correspondiente.

  Para los prototipos se usaron tres códigos QR y estos están configurados dentro del *ManagerPhoneGapPluginQR.* También se cuenta con la asociación entre los códigos QR y los *SensorQR* (abstracciones de los códigos).

- La clase *ManagerAndoridLocationGPS* implementa la interfaz *LocationListener*[17] que es parte del framework de *Android*. Esta es una forma de estar escuchando por eventos de GPS. Otra manera es usar la librería posicionamiento de *Google Place*[18], para esto se debería implementar una nueva subclase de *ManagerGPS.*

  Para los prototipos la clase *ManagerPhoneGapPluginQR* tiene la asociación entre GPS físico y el *SensorGPS* (abstracción del GPS). Cada vez que se recibe un nuevo valor de GPS, este es seteado en el *SensorGPS*. Esto desencadena la propagación de dicho valor, y en el caso que corresponda este puede ser propagado hasta la característica de contexto correspondiente.

Por lo tanto, los *ManagerListener* son los encargados de recibir los valores de los sensores físicos, y propagarlos a las abstracciones de los mismos.

Muchas veces configurar los sensores físicos es complejo, y para este tipo de aplicaciones generalmente es una tarea muy similar, de hecho los mismos sensores son usados en los dos prototipos implementados. Además, como se pudo apreciar en la Figura 3.9 la instanciación se puede convertir en una tarea compleja, y repetitiva.

Acorde a esto, se buscó una forma fácil de configurar nuestro framework para indicar cuáles son los mecanismos de posicionamiento que se desean usar, en este caso, se decidió realizarlo mediante la configuración de un archivo XML. Para esto desarrollamos una librería que se detalla en el Anexo D.

En el Código 4.1 se puede observar el XML relacionado a los mecanismos de sensados descriptos anteriormente que son los que usaran ambos framework. En particular, este XML facilita la configuración de las instancias de la Figura 3.9. Es decir, los usuarios del framework podrían usar directamente este XML de configuración y evitar realizar la instanciación de cada una de las clases como se mostró en la Figura 3.9. Se puede observar en el Código 4.1 que hay diferentes tags, los cuales son interpretados y a partir

[16] Página de *BarcodeScanner*: https://github.com/hypery2k/cordova-barcodescanner-plugin (Ultimo acceso: 16/11/2017)
[17] Página de *LocationListener de Android*: https://developer.android.com/reference/android/location/LocationListener.html (Ultimo acceso: 16/11/2017)
[18] Página de *LocationListener de Google Place*: https://developers.google.com/android/reference/com/google/android/gms/location/LocationListener (Ultimo acceso: 16/11/2017)

de los mismos se realizan la instanciación de las clases correspondientes al framework, incluyendo la información de los *ManagerListener* presentados en la Figura 4.1. Para más detalles relacionados a los tags y el parseo se pueden consultar en el Anexo D.

```
<app>
    <concern>
        <contextFeature>qrValue</contextFeature>
        <methodology>transformXYZ</methodology>
        <dispatcher>dispatcherPass</dispatcher>
        <politic>push</politic>
        <aumentedSensorQR>
            <feature><key>positionXYZ</key>
                <value><positionXYZ>1240,750,0</positionXYZ></value>
            </feature>
            <id>qr1</id>
        </aumentedSensorQR>
    </concern>
    <concern>
        <contextFeature>beaconValue</contextFeature>
        <methodology>transformXYZ</methodology>
        <dispatcher>dispatcherDistinctValue</dispatcher>
        <politic>push</politic>
        <aumentedSensorBeacon>
            <feature><key>positionXYZ</key>
                <value><positionXYZ>1260,670,0</positionXYZ></value>
            </feature>
            <id>87/58262</id>
        </aumentedSensorBeacon>
    </concern>
    <concern>
        <contextFeature>gpsValue</contextFeature>
        <methodology>nullTranform</methodology>
        <dispatcher>dispatcherDistinctValue</dispatcher>
        <politic>push</politic>
        <sensorGpsAndroid/>
    </concern>
</app>
```

Código 4.1: Configuración XML de los mecanismos de sensados.

Cabe mencionar que ambos prototipos están implementados de forma general para probar el funcionamiento puntual de los mecanismos de sensado de posición, acorde a esto se eligió contar con pantallas simples donde el foco fueran mapas que permitan apreciar la posición actual del usuario. Es decir, no se aborda ningún dominio puntual de aplicación.

A continuación se describen las librerías javascript usadas. Para aspectos visuales en general se usaron las siguientes librerías:

- *jCanvas*[19]**:** version v16.06.06
- *jQuery*[20]**:** version jquery-2.1.1.min.js

[19] Página de *jCanvas*: https://projects.calebevans.me/jcanvas (Último acceso: 16/11/2017)

- *jQueryMobile*[21]: version 1.4.5
- jQuery UI[22]: version 1.11.4 - 2015-03-11
- *SweetAlert2*[23] para los carteles de notificaciones o avisos.

Respecto a mapas puntuales se usaron as siguientes librerías:

- API de Google Maps[24]: versión 3.31.1

  Además de las funciones de mapas tradicionales, se usó también *Ground Overlays* [25] para superposición de imágenes (plano) sobre el mapa de google.

Las pruebas de ambos prototipos se realizaron con un Samsung S4 con sistema operativo *Android 4.4.2*. Si bien la interfaz puede adaptarse a otras dimensiones de pantallas, no se puede asegurar que todo se pueda visualizar correctamente en otro dispositivo.

4.2 Descripción de los Prototipos

En esta sección se entrará en más detalle de cada uno de los prototipos ya que ambos tienen un funcionamiento distinto en relación a como los valores sensados son interpretados.

4.2.1 Prototipo con tres características de contexto independientes, cada una de ellas guarda las posiciones relacionadas a un mecanismo de sensado de posición

El primer prototipo desarrollado tuvo la finalidad de poder probar los tres mecanismos de sensados y como poder mostrar la posición capturada por cada uno de ellos. Por esta razón en este caso, se usan tres características de contexto diferente, similar a lo presentado en la Figura 3.9.

Recordemos que en la Figura 3.9 se tenía un diagrama de instancias con un solo *beacon*, un solo código QR, y el GPS. En el caso de los prototipos cada uno fue instanciado con tres *beacons*, tres códigos QR, y el GPS.

Para poder representar mejor la posición de los *beacon* se decidió crear un transformador particular, denominado *TransformCicleArea*, el cual busca entre las características dinámicas del *beacon* una (x,y,z) o (latitud, longitud) según corresponda y a partir de allí define una posición *CircleArea* de esta manera se tiene más información para posicionar al usuario, el radio lo define el valor recibido por el *beacons*.

---

[20] Página de *jQuery*: https://jquery.com (Último acceso: 16/11/2017)
[21] Página de *jQueryMobile*: https://jquerymobile.com (Último acceso: 16/11/2017)
[22] Página de *jQuery UI:* https://jqueryui.com (Último acceso: 16/11/2017)
[23] Página de *SweetAlert2*: https://github.com/limonte/sweetalert2 (Último acceso: 16/11/2017)
[24] Página de la *API de Google Maps*: https://developers.google.com/maps/documentation/javascript/?hl=es-419 (Último acceso: 16/11/2017)
[25] Página de *Ground Overlays:* https://developers.google.com/maps/documentation/javascript/examples/groundoverlay-simple?hl=es-419(Último acceso: 16/11/2017)

Además, para poder mostrar la posición GPS sobre un mapa, se decidió crear un transformador denominado *TransformGPSValueToLatLong.* Esto es necesario porque el GPS devuelve un valor de tipo *Location* (que está dentro del paquete `android.location`), con lo cual el mismo tiene que ser procesado para obtener a partir de este la posición expresada en (latitud, longitud).

Los dos transformadores *TransformCicleArea* y *TransformGPSValueToLatLong* se pueden observar en la Figura 4.2, los cuales implementan la interfaz *TransformMethodology*.

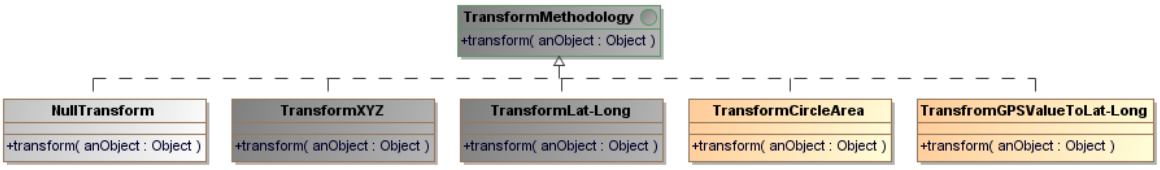

Figura 4.2: Definicion de las clases *TransformCicleArea* y *TransformGPSValueToLatLong*.

De esta manera, se puede apreciar como el framework propuesto se puede extender con trasformadores específicos según los requerimientos de las aplicaciones.

En la Figura 4.3 se pueden apreciar las instancias de las características de contexto y las abstracciones de los sensores del *Prototipo 1*. Por simplicidad, no se muestra la aplicación ni el usuario, dado que esto no varía respecto a la Figura 3.9.

Se puede observar en la Figura 4.3 que las tres abstracciones de los códigos QR tienen el mismo *transformador* y *dispatcher*, *TrasnformXYZ* y *DispatchaPass* respectivamente. Además, los tres códigos QR modifican la misma característica de contexto (*qrValue*).

Algo similar sucede con las tres abstracciones de los *beacons*, tienen el mismo *transformador* y *dispatcher*, *TrasnformCicleArea* y *DispathDistinctValue* respectivamente. Además, los tres *beacons* modifican la misma característica de contexto (*beaconValue*).

En el caso de la abstracción del GPS, tiene un *TransformGPSValueToLatLong* y un *DispathDistinctValue* (que es el mismo que usan los *beacons*).

En particular, todas las posiciones (x,y,z) están definidas en relación al edificio de La Facultad de Informática de la UNLP. Asignadas de las siguientes maneras:

- Códigos QR
  - *Puerta principal*: en la posición (1240, 750, 0)
  - *Puerta de la Fotocapiadora*: en la posición (1190, 690, 0)
  - *En secretaria de decanato:* en la posición (1010, 780, 1)
- *Beacons*
  - *En el hall central:* en la posición (1260, 670, 0)
  - *En fotocopiadora:* en la posición (820, 360, 0)
  - *En Secretaria de Postgrado:* en la posición (1010, 780, 2)

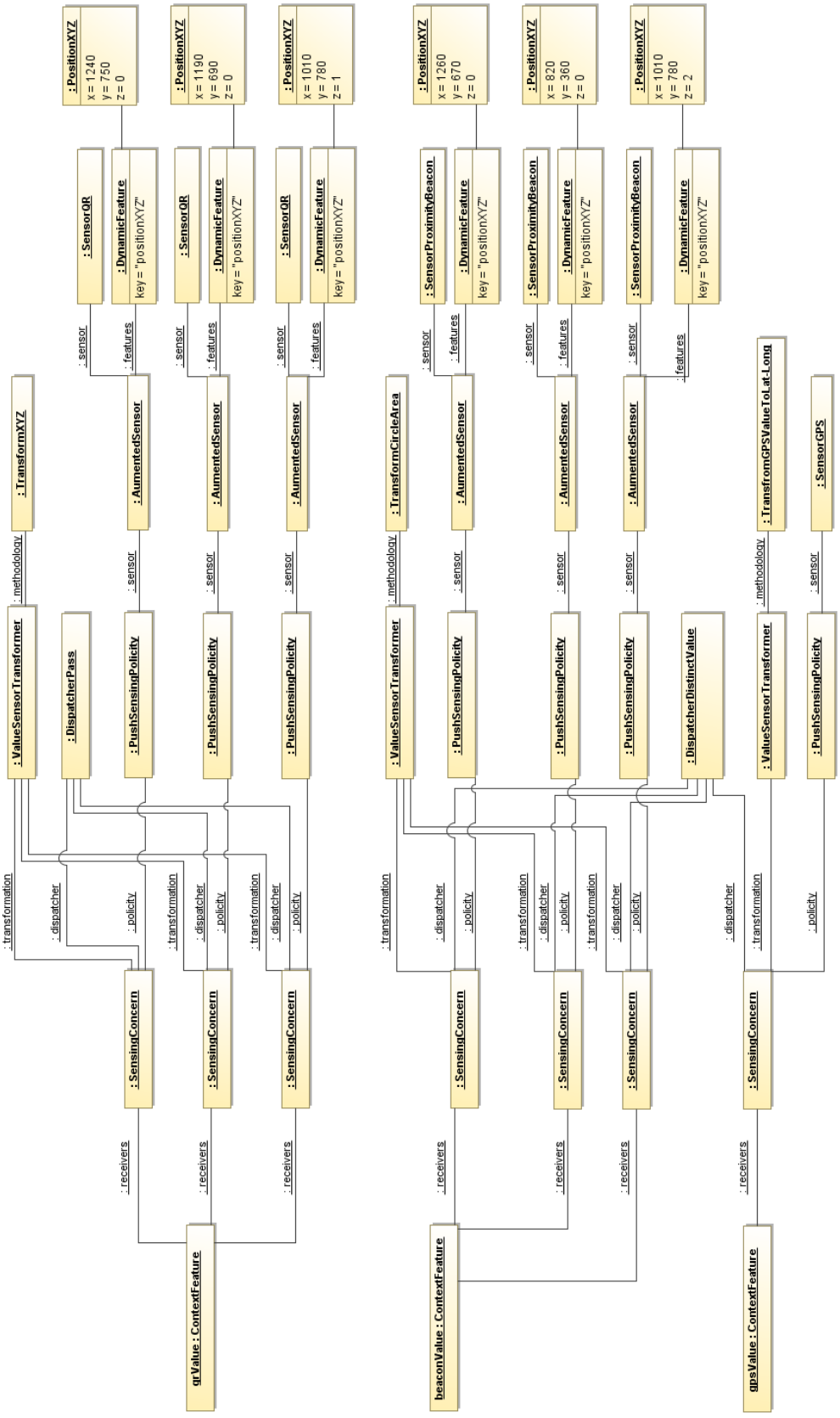

Figura 4.3: Instancias de las abstracciones de los sensores usadas en el *Prototipo 1*.

La clase *Prototype1Activity* es una subclase de *CordobaActivity*, y es la encargada de conocer el punto de entrada del framework, como se puede apreciar en la Figura 3.4. Además, estará observando a la aplicación para poder propagar los cambios que se detecten.

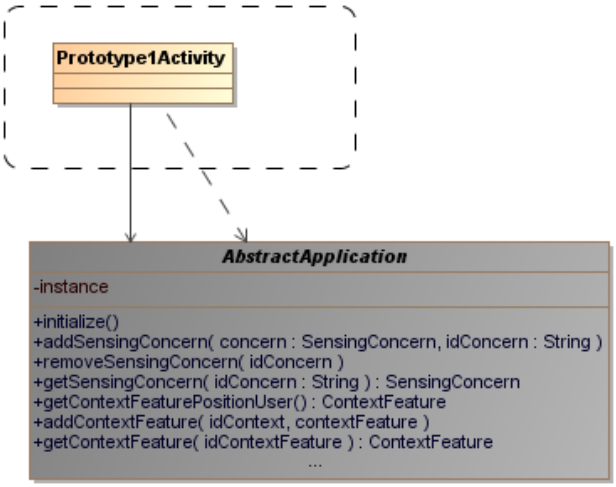

Figura 4.4: Clase *Prototype1Activity*.

En la Figura 4.5 se puede apreciar la pantalla de *Prototipo 1*, en la parte superior usando un mapa de google se muestra la posición GPS, mientras que en la parte de abajo se muestra tanto la posición detectada por los códigos QR y los *beacons*.

Figura 4.5: Pantalla del *Prototipo 1*.

Se puede observar en la Figura 4.5 que se cuenta con una barra indicativa de los pisos del edifico de la Facultad de Informática y se pueden distinguir los siguientes iconos:

 Representa el plano actual que está viendo el usuario

 Representa que en ese piso el usuario fue detectado por un *beacon*

 Representa que en ese piso el usuario leyó un código QR

El usuario puede desplazarse digitalmente por los planos de los tres pisos, pero los iconos relacionados al posicionamiento solo cambiarán de piso, cuando el usuario sea sensado por otro *beacon* o lea un código QR de otro piso.

También se puede observar en la Figura 4.5 la posibilidad de poder acceder al escaner de códigos QR, usando el siguiente icono:

Al tocar esta opción se abre el lector de códigos QR

4.2.2 Prototipo con una característica de contexto que guarda un objeto que combina las posiciones provenientes de tres mecanismos de sensado de posición

El segundo prototipo desarrollado tuvo la finalidad de poder probar la combinación de los tres mecanismos de sensados y cómo poder mostrar la posición capturada por cada uno de ellos de manera unificada. En este prototipo también se utilizan tres *beacons*, tres códigos QR, y el GPS. Sin embargo, en este caso se quiere contar con una única característica de contexto que permita guardar un objeto complejo.

El objeto que se guardará en la característica de contexto tiene la siguiente información:

- *currentPosition*, guarda posiciones *LatLongPosition* o *Undefined*
- *typeOfSensor*, guarda una descripción del último mecanismo que sensó al usuario.
- *date*, guarda la fecha de la toma del valor sensado
- *history*, guarda el historial de valores sensados en forma de asociaciones, valor-date, para poder saber la antigüedad del valor sensado.
    - *GPS*, guarda una asociación (value, date)
    - *Beacon*, guarda una asociación (id, value, radius, date)
    - *QR*, guarda una asociación (value, date)

Para poder armar este objeto que combina los valores de los sensores, se necesita contar con un transformador especializado, que tenga la lógica necesaria para poder generar este nuevo objeto, por esta razón se definió la clase *ValueCombinedSensorTransformer*

Dado que todo el procesamiento y lógica que hay que realizar es mucho más complejo que una simple transformación, dicha clase se creó como una extensión de la clase *ValueSensorTransformer*. Esta nueva clase se puede apreciar en la Figura 4.6.

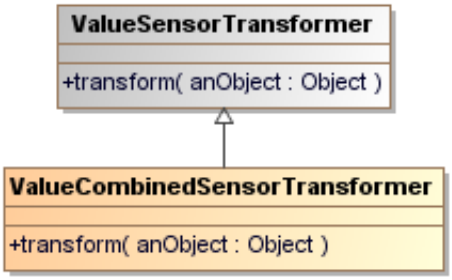

Figura 4.6: Creación de la clase *ValueCombinedSensorTransformer*.

Por otro lado, la propagación del valor sensado depende de las prioridades que se le asignan a cada mecanismo de sensado, y acorde a esto considerar si el valor debe ser propagado o no.

Una posible forma de análisis para determinar si se propaga el cambio en el *currentSituation* del objeto complejo podría ser la descrita a continuación, considerando que siempre se propaga el valor que llega al historial del objeto complejo.

- Llega un valor nuevo de QR → se propaga el cambio en el *currentSituation*
- Llega un valor nuevo de un *Beacon*
    - Si el anterior valor era de QR
        - Se analiza si el valor del mismo tiene más de 2 minutos
            - Caso afirmativo → se propaga el cambio en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era de un *Beacon:*
        - Se analiza si el valor del mismo es distinto al nuevo valor del mismo beacon o el nuevo valor es de otro beacon
            - Caso afirmativo → se propaga el cambio en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era del GPS → se propaga el cambio en el *currentSituation*
    - Si el anterior valor era *Undefined* → se propaga el cambio en el *currentSituation*

- Llega un valor nuevo de GPS
    - Si el anterior valor era de QR:
        - Se analiza si el valor del mismo tiene más de 2 minutos
            - Caso afirmativo → se propaga el cambio en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era de un *Beacon:*
        - Se analiza si el valor del mismo tiene más de 4 minutos
            - Caso afirmativo → se propaga el cambio en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era del *GPS:*
        - Se analiza si el valor del mismo es distinto al nuevo valor
            - Caso afirmativo → se propaga el cambio en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era *Undefined* → se propaga el cambio en el *currentSituation*

Esto a su vez se podría complejizar más si se considera que los *beacons* pueden emitir una señal de timeout para indicar que se salió del radio de alcance del mismo. En todos los casos, se modifica el valor del historial del *beacon* del objeto complejo indicando la posición *Undefined.*

Para el caso del timeout se podrían tener las siguientes consideraciones para determinar si se propaga o no el cambio en el *currentSituation* del objeto complejo. Un posible análisis podría ser el siguiente:

- Llega un valor de timeout de *beacon*
    - Si el anterior valor era de QR:
        - Se analiza si el valor del mismo tiene más de 2 minutos
            - Caso afirmativo → se propaga *Undefined* en el currentSituation
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era de un *Beacon* → se propaga *Undefined* en el *currentSituation*

- Si el anterior valor era del *GPS:*
    - Se analiza si el valor del mismo tiene más de 2 minutos
        - Caso afirmativo → se propaga *Undefined* en el currentSituation
        - Caso negativo → NO se propaga el cambio en el *currentSituation*
- Si el anterior valor era *Undefined* → NO se propaga el cambio en el *currentSituation*

Algo similar ocurre si se considera una señal de timeout del GPS. En todos los casos, se modifica el valor del historial del *GPS* del objeto complejo indicando la posición *Undefined.* En este caso se podría tener las siguientes consideraciones para determinar si se propaga o no el cambio en el *currentSituation* del objeto complejo.

- Llega un valor de timeout de *GPS*
    - Si el anterior valor era de QR:
        - Se analiza si el valor del mismo tiene más de 2 minutos
            - Caso afirmativo → se propaga *Undefined* en el currentSituation
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era de un *Beacon:*
        - Se analiza si el valor del mismo tiene más de 2 minutos
            - Caso afirmativo → se propaga *Undefined* en el *currentSituation*
            - Caso negativo → NO se propaga el cambio en el *currentSituation*
    - Si el anterior valor era del *GPS* → se propaga *Undefined* en el *currentSituation*
    - Si el anterior valor era *Undefined* → NO se propaga el cambio en el *currentSituation*

Acorde a lo antes descripto, se puede apreciar la complejidad relacionada con poder determinar en este caso si el valor debe o no ser propagado. Para poder realizar este análisis se definió un *dispatcher* nuevo denominado *DispatcherCombinedSensor.* Esta clase es la encargada de realizar todo el análisis anteriormente descripto y en base a eso, propagar o no el nuevo valor recibido de alguno de los mecanismos de sensado de posición.

La nueva clase *DispatcherCombinedSensor* se puede apreciar en la Figura 3.7.

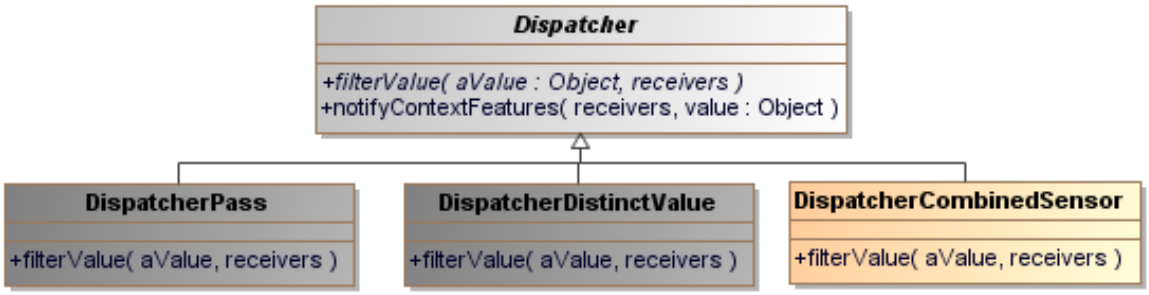

Figura 4.7: Creación de la clase *DispatcherCombinedSensor*.

En la Figura 4.8 se pueden apreciar la instancia de la característica de contexto y las abstracciones de los sensores del *Prototipo 2*. Por simplicidad, no se muestra la aplicación ni el usuario, dado que esto no varía respecto a la Figura 3.9.

Se puede observar en la Figura 4.8 que las tres abstracciones de los códigos QR, las tres abstracciones de los *beacons* y la abstracción del GPS tienen el mismo *transformador* y *dispatcher*, *ValueCombinedSensorTransformer* y *DispatcherCombinedSensor* respectivamente.

Además, todas las abstracciones de los sensores modifican la misma característica de contexto (*combinedValue*). Para el *Prototipo 2* todas las posiciones terminan siendo expresadas con el formato (latitud, longitud). Estas también coinciden con posiciones definidas en relación al edificio de La Facultad de Informática de la UNLP.

Se puede apreciar que la instancia *ValueCombinedSensorTransformer* conoce:

- a una instancia de la clase *TransformLatLong* para poder determinar la posición (latitud, longitud) tanto de los códigos QR como de los *beacons*.
- a una instancia de la clase *TransformGPSValueToLatLong* para poder transformar la posición del GPS en un valor (latitud, longitud).

Todas las posiciones (latitud, longitud) están definidas en relación al edificio de La Facultad de Informática de la UNLP. Asignadas de las siguientes maneras:

- Códigos QR
  - *Puerta principal*: en la posición (-34.90348,-57.93754)
  - *Puerta de la fotocopiadora*: en la posición (-34.90342,-57.93773)
  - *En Secretaría de Decanato:* en la posición (-34.90342,-57.93773)
- *Beacons*
  - *En el hall central:* en la posición (-34.90338,-57.93765)
  - *En fotocopiadora:* en la posición (-34.90346,-57.938261)
  - *En Secretaría de Postgrado:* en la posición (-34.90357,-57.93768)

Se puede observar que las mismas se han truncado a cinco decimales, esto hace que las mismas varíen por muy poca diferencia donde a veces no son tan precisas cuando luego se muestran en un mapa.

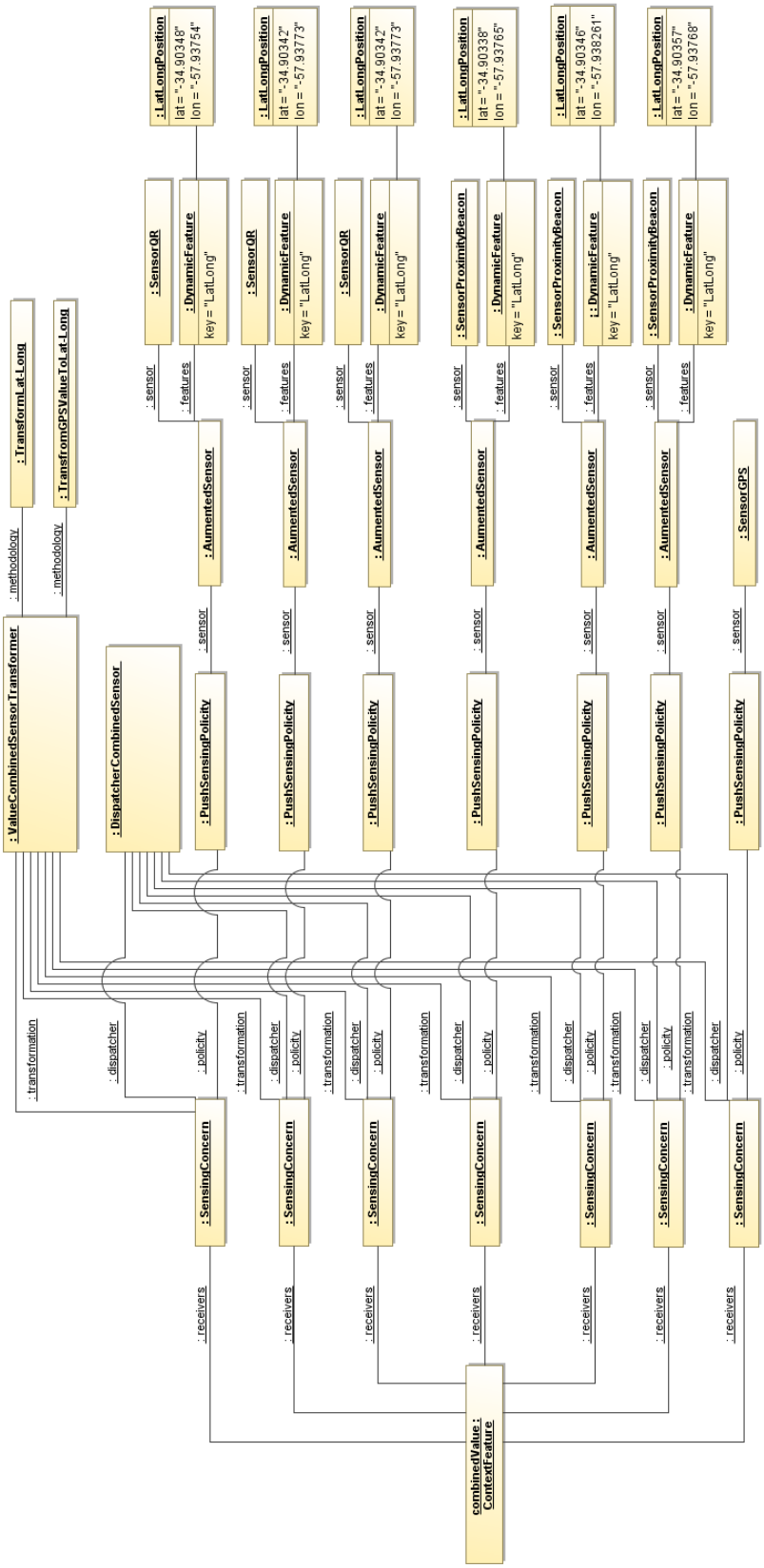

Figura 4.8: Instancias de las abstracciones de los sensores usadas en el *Prototipo 2*.

Al igual que se describió para el *Prototipo 1,* se definió una subclase de *CordobaActivity*, denominada *Prototype2Activity*, que es la encargada de conocer el punto de entrada del framework, como se puede apreciar en la Figura 3.9. Además, estará observando a la aplicación para poder propagar los cambios que se detecten.

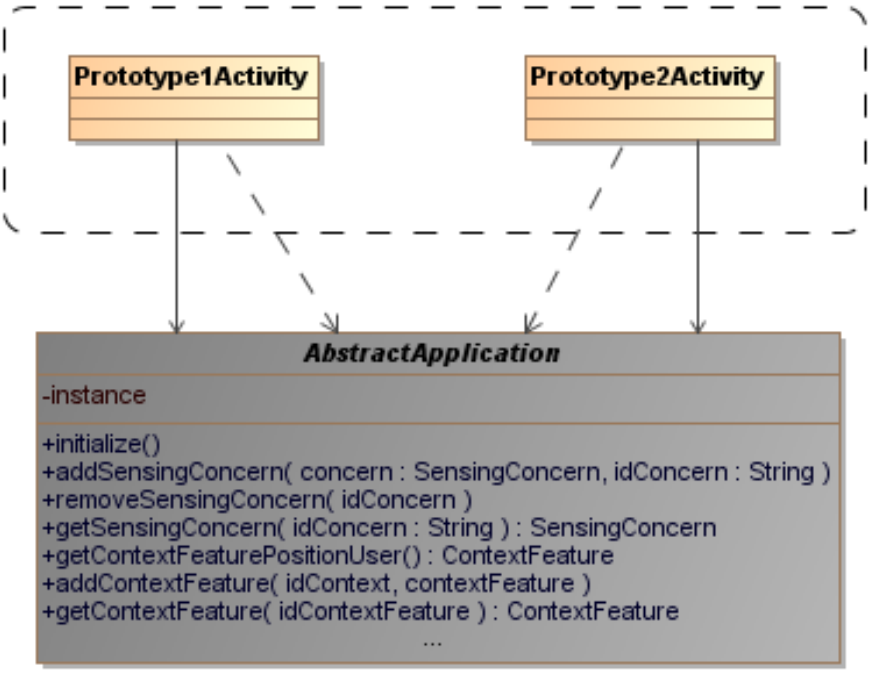

Figura 4.8: Clase *Prototype2Activity*.

En la Figura 4.9 se puede observar las nuevas extensiones realizadas al framework, las cuales surgieron a partir del implementar concreta los prototipos y detectar necesidades puntuales de cada uno de estos.

Cabe mencionar que las clases destacadas en gris fueron presentadas como parte del framework propuesto, en la Figura 3.9 se destaca a color aquellas incorporaciones realizadas para ambos prototipos.

Dependerá del tipo de aplicación que se desea realizar, puede que las clases definidas sean suficiente para el comportamiento de dicha aplicación, o podría ser necesario seguir extendiendo el framework como se fue mostrando para ambos prototipos.

La complejidad de los *transformadores* y *dispatcher* dependerá del procesamiento que se requería hacer, se pudo apreciar que para el prototipo 1 eran simples sin embargo dicha complejidad se incrementó al querer combinar más de un mecanismo de sensado y tener toda esa información reflejada en una única característica de contexto.

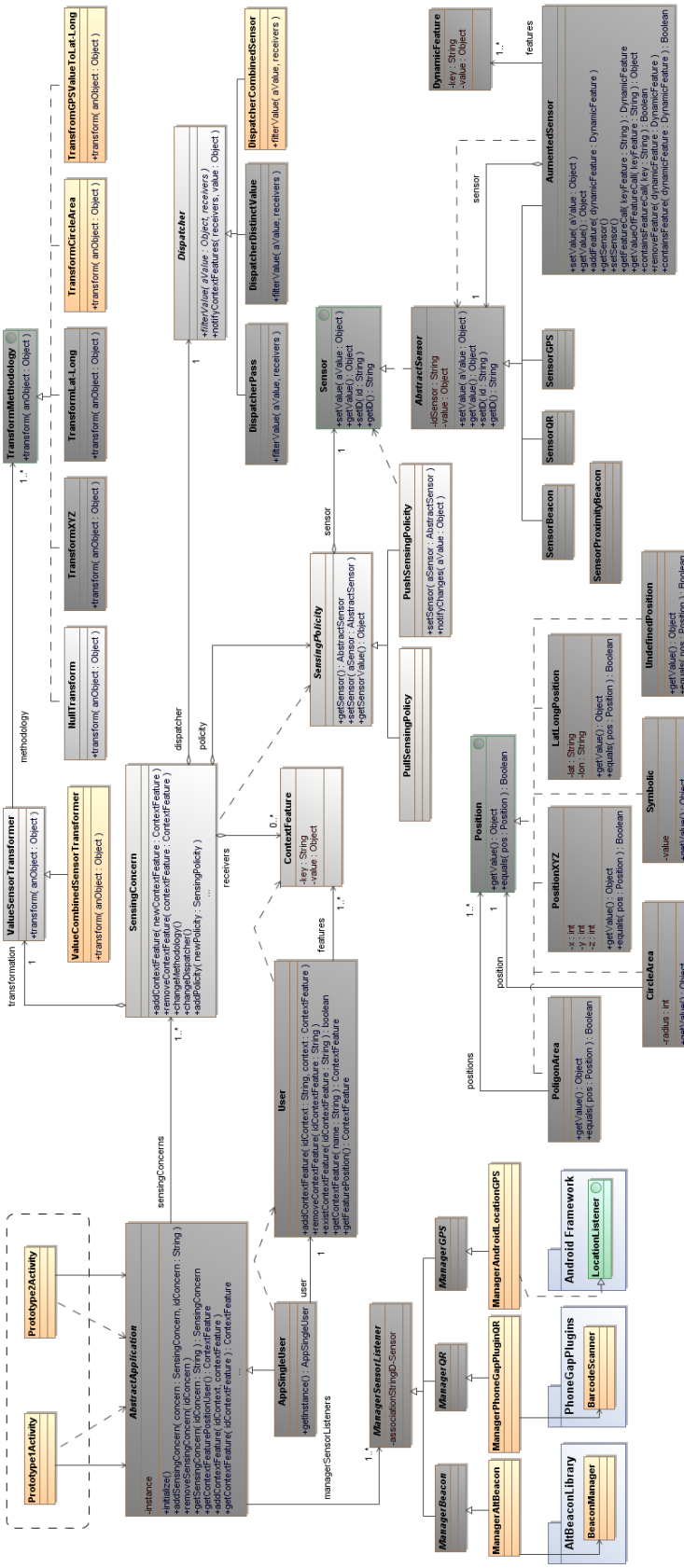

Figura 4.9: Extensiones realizadas en los prototipos.

En la Figura 4.10 se puede apreciar la pantalla de *Prototipo 2.* Por simplicidad[26], se eligió tener un solo piso representado de la Facultad de Informática, superpuesto sobre el mapa de google. Con lo cual los sensores que tienen sentido en este caso serán aquellos que se encuentran en la planta baja, es decir, dos *beacons* y dos códigos QR.

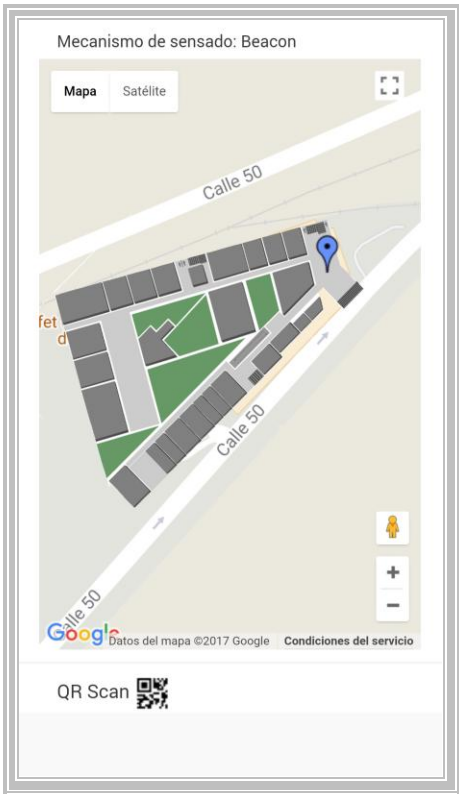

Figura 4.11: Pantalla del *Prototipo 2*.

4.3 Problemas detectados

Durante el trascurso del desarrollo de ambos prototipos fueron surgiendo diferentes problemáticas relacionadas tanto a los mecanismos de sensado como al desarrollar la configuración mediante un archivo XML.

A continuación se detallas las problemáticas que se debieron abordar en esta tesina:

- *Relacionadas con la librería AltBeacon (comunicación con los beacons físicos)*

  Cabe aclarar que la librería *AltBeacon* es open source (más detalles de otros protocolos de comunicación de *beacons* se pueden apreciar en el Anexo D). Se decidió no usar la SDK que nos proveía *Onyx Beacons* (marca de los *beacons* físicos) porque era cerrada. Al usar *AltBeacon* se pudo investigar y modificar la librería para realizar distintas pruebas.

[26] En un futuro se podría contar con funcionalidad similar a la propuesta por google para mapas indoor: https://www.google.com/maps/about/partners/indoormaps (Último acceso: 16-11-2017). Por ahora este tipo de mapas están solamente disponibles para algunos edificios.

Al comenzar a usar *AltBeacon* se detectaron las siguientes problemáticas:

- *Comportamiento inesperado de la librería AltBeacon versión 2.8:*

  Al comenzar el desarrollo la librería *AltBeacon* estaba en la versión 2.8, con lo cual se comenzó el desarrollo con dicha versión. Al realizar distintas pruebas se detectó que cada vez que se volvían a iniciar los prototipos, no se detectaba el último beacon "*leído*". Es decir, había un problema de persistencia de los datos respecto al *BeaconManager* de la librería, y se generaba inconsistencia en el funcionamiento.

  La solución que encontramos inicialmente a esto, fue borrar los datos de la aplicación "manualmente" antes de iniciarla nuevamente. Luego, la solución que se implemento es desconectarse del *BeaconManager* y volver a crear todo en la próxima ejecución de los prototipos.

  Mientras se avanzaba en el desarrollo, la gente de *AltBeacon* sacan la versión 2.9 donde hicieron explícito[27] esta opción de persistencia, solucionando así el problema de la versión 2.8. Dado los tiempos y el impacto que podía tener cambiar de versión se decidió continuar con la versión 2.8 con la solución mencionada anteriormente que desconectaba el *BeaconManager* cada vez que se cierra la aplicación.

- *Se generaba un evento de "Salida de la Región" aún estando cerca de un beacon:*

  Al ir realizando distintas pruebas con *AltBeacon*, cada cierto tiempo por más que se esté cerca del *beacon* se lanza un evento de " *Salida de la Región*", esto se debe a un tema de señal y como maneja internamente esto la librería. Por defecto, el valor está seteado para lanzar ese evento es de 10 segundos. Esto parecía excesivo, así que se probó y se llegó a la conclusión de dejarlo en 4 segundos. Es un valor aceptable que muy ocasionalmente lanza ese evento de manera errónea pero al mismo tiempo se tiene mejor feedback con el usuario ya que no tiene un "*delay*" de respuesta de 10 segundos mínimo. Con valores menos a 4 la cantidad de eventos de salida erróneos aumenta notoriamente.

- *Problemas con la señal de los beacons:*

  Como se describe en el Anexo B, dependiendo de la cercanía de los *beacons* estos comienzan a interferirse, indicando que se está más cerca de un *beacon* cuando en realidad se está más cerca de otro. A los fines de los prototipos implementados, se decidió que los *beacons* estuvieran lo suficientemente lejos

[27] Páginas donde se menciona el problema encontrado en la versión 2.8:
http://altbeacon.github.io/android-beacon-library/state-persistence.html (Ultimo acceso 16-11-2017)
https://github.com/AltBeacon/android-beacon-library/issues/410 (Ultimo acceso 16-11-2017)
https://github.com/AltBeacon/android-beacon-library/releases/tag/2.9 (Ultimo acceso 16-11-2017)

uno de otro para no interferirse. Otros problemas de señal son detallados en el Anexo D.

- *Relacionadas con PhoneGap* y el plugin presentado en [Zimbello and Challiol, 2016]

  Recordemos que *PhoneGap* permite construir aplicaciones híbridas basadas en HTML, CSS y *Javascript*. Cuando la aplicación *PhoneGap* es puramente cliente como en los casos de los prototipos, se comporta como una arquitectura cliente/servidor dentro de la misma aplicación. El "*cliente*" es la parte web, mientras que el "*servidor*" es la subclase de *CordobaActividad* donde generalmente esta clase tiene muy poca lógica implementada, y sirve para poder redireccionar a la página de inicio de la aplicación.

  En el caso de los prototipos, se tiene adicionalmente un modelo *Java* con todo el framework propuesto, con lo cual sale del funcionamiento tradicional de una aplicación *PhoneGap.* Para poder invocar métodos específicos del framework desde *Javascript* se usó el plugin presentado en [Zimbello and Challiol, 2016]. Esto permitió contar con un comportamiento fuera del uso tradicional de *PhoneGap.* Más detalles se pueden apreciar en el Anexo C.

  Trabajar con plugins implica también entender ese flujo[28] fuera de lo tradicional y cómo se comporta la aplicación. En algunos casos, cuando se invoca un plugin de *ApachecCordova*, se destruye la actividad principal. Esto ocurre, por ejemplo, cuando se abre la cámara para leer un código QR. Esto generaba inconvenientes para poder tener siempre "viviendo" la relación de conocimiento que tienen las clases *Prototype1Activity* y *Prototype2Activity* (descriptas en las Secciones 4.2.1 y 4.2.2 respectivamente) respecto a la instanciación framework. Acorde a ésto se modificó el pluign del *BarcodeScanner* para poder volver el control a las clases *Prototype1Activity* y *Prototype2Activity* y así no perder ninguna información de instanciación del framework.

- *Relacionadas con la librería usada para el archivo de configuración XML*

  Si bien en el Anexo D se mencionan las características principales de la librería utilizada, cabe mencionar que para la realización de la misma se exploró la librería *Xstream*. A continuación se hará hincapié en aquellos problemas que surgieron hasta lograr decidir la librería que finalmente quedó funcionando *para tomar los datos del archivo de configuración XML*.

  - *Problemas entre Java 8 y Xstream versión 1.4.7:*

    No se pudo usar *Java 8* dado que *Xstream* tiene problemas, los cuales están reportados en la página oficial[29] de la librería, donde comunican que para la

[28] Más detalles sobre el funcionamiento de los pluign en PhoneGasp: http://cordova.apache.org/docs/en/7.x/guide/platforms/android/plugin.html#page-toc-source (Último acceso: 16-11-2017)

[29] Páginas donde se reporta el problema de *Xstream* y Java 8 y:

próxima versión (1.5) van a volver a usar *Java 7*. Este problema fue detectado querer usar la librería con *Java 8* en el *AndroidStudio*. Por lo tanto, se decidió usar en ambos prototipos *Java 7* y modificar algunos detalles para ajustar a la interfaz y métodos usados en este versión de java.

- *Xstream y los atributos definidos como "static"*[30]

  Xstream no tiene en cuenta los atributos definidos como "static" (en las clases) porque para la librería "*no tiene sentido deserializar campos estáticos*". Para nosotros son fundamentales porque uno de los múltiples patrones que aplicamos es el *Singleton* y si no se persistiesen los campos estáticos, no se guardaría la instancia "*interna*" y por tanto perderíamos la información que precisamente queríamos persistir. Acorde a ésto, se hicieron algunas modificaciones a la librería *Xstream* para poder permitir guardar y levantar campos estáticos.

- *Clases ResourceBundle*[31] *vs. Properties*[32]

  Al definir la librería para la configuración de archivos XML se había comenzado usado la clase *ResourceBundle* para leer los archivos .properties. El problema surgió al querer cambiar en tiempo de ejecución estos archivos. La clase *ResourceBundle* es abstracta y el uso se hace por medio del método de instancia `ResourceBundle.getBundle()`, el cual no reflejaba los cambios en una nueva lectura del archivo. Por esta razón, se decidió usar la clase *Properties*, la cual si permitía tomar los cambios cuando dichos archivos cambian en tiempo de ejecución.

- *Lectura y escritura en tiempo de ejecución Android*

  Para poder tener archivos de configuración que pudieran cambiar dinámicamente, es decir, que el usuario del framework puede cambiar valores de los mecanismos de sensado, se decidió trabajar, por ahora, con la carpeta *Downloads* de los dispositivos móviles para trabajar con archivos que se tengan que modificarse en tiempo de ejecución. Esta decisión se debe a que había que asegurar que la carpeta del dispositivo tenga permisos de lectura/escritura. En un futuro se podría crear carpetas específicas dentro de los dispositivos móviles. Por una cuestión de tiempo se tomó esta decisión que permitía lograr una solución a la problemática de escribir archivos en tiempo de ejecución.

---

https://github.com/x-stream/xstream/issues/49 (Último acceso: 16-11-2017)
http://x-stream.github.io/index.html (Último acceso: 16-11-2017)

[30] Página sobre los atributos estáticos y la librería *Xstream*: http://xstream.10960.n7.nabble.com/How-to-handle-static-fields-with-xstream-td5562.html (Último acceso: 16-11-2017)
[31] Clase *ResourceBundle*: https://docs.oracle.com/javase/7/docs/api/java/util/ResourceBundle.html (Último acceso: 16-11-2017)
[32] Interfaz Properties: https://docs.oracle.com/javase/7/docs/api/java/util/Properties.html (Último acceso: 16-11-2017)

5. Ejemplos de uso de los prototipos desarrollado

En este capítulo se presentarán ejemplos de usos de los dos prototipos desarrollados. Se mostrará como el cambio en los valores de los sensores se ve reflejado en la pantalla del dispositivo móvil. Cabe mencionar que las pruebas fueron realizadas en un *Samsung S4*, y se fueron simulando para poder mostrando las pantallas visualizadas en este Capítulo.

5.1 Ejemplos de uso del *Prototipo 1*

En esta sección se presentará una descripción del *Prototipo 1*. Supongamos que arrancamos el prototipo con el GPS y los *beacons* desconectados. En la Figura 5.1 se puede visualizar la pantalla que se recibe. Sólo se cuenta con el ícono del *ojo* que muestra que plano se está visualizando actualmente.

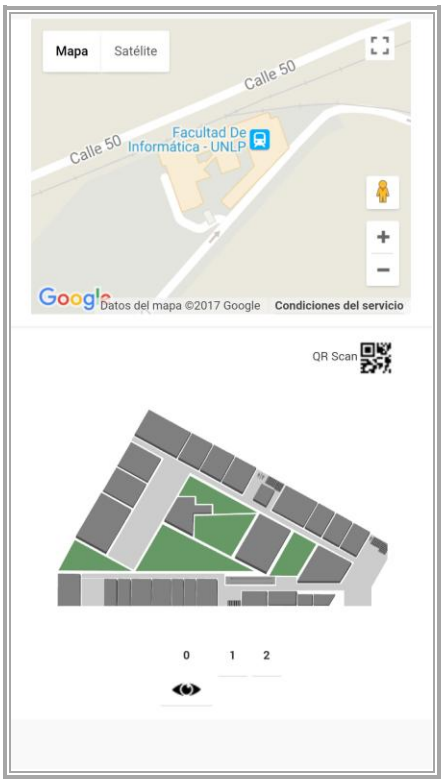

Figura 5.1: *Prototipo 1* – Pantalla de inicio con el GPS y *beacons* desconectados.

Supongamos que se enciente tanto el GPS como se encienden los *beacons*, en este momento depende de donde esté posicionada la persona será si es sensada o no por alguno de los *beacons*. Para este ejemplo, supongamos que el usuario está entrando al edificio de la facultad. La pantalla que recibe se puede apreciar en la Figura 5.2, se puede

apreciar la posición GPS, pero además se visualiza el *beacon* que esta sensando al usuario, en este caso, el *beacon* que está en el hall de entrada del edificio.

Figura 5.2: *Prototipo 1* – Pantalla con el GPS encendido y un *beacon* sensa al usuario.

El usuario encuentra el código QR posicionado en la fotocopiadora, y elige escanear el código, en este caso, la aplicación le abre el lector de código QR como se puede apreciar en la Figura 5.3.

Figura 5.3: *Prototipo 1* – Se elige la opción de escanear un códigos QR.

Al leer el código QR mostrado en la Figura 5.3, el usuario recibe la pantalla que se muestra en la Figura 5.4. Se puede observar que el usuario todavía sigue siendo sensado también por el *beacon*.

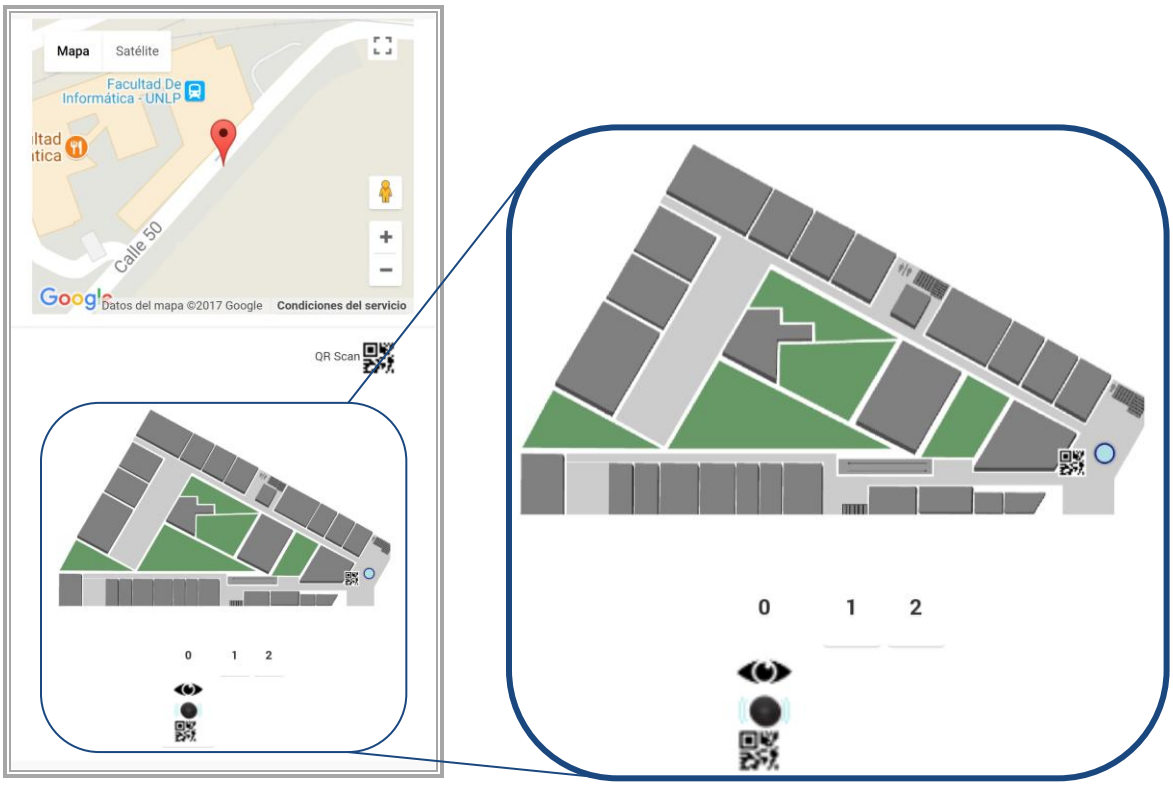

Figura 5.4: *Prototipo 1* – Lectura de un códigos QR.

Supongamos que el usuario empieza a caminar por la facultad, y en un momento dado es sensado por el *beacon* posicionado en la fotocopiadora, en ese momento recibe una pantalla como se muestra en la Figura 5.5.

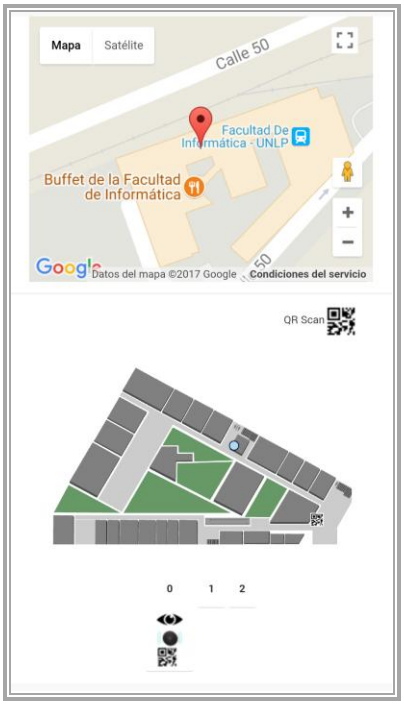

Figura 5.5: *Prototipo 1* – El usuario es sensado por otro *beacon*.

En la Figura 5.5 se puede apreciar que el código QR sigue estando marcado, esto se debe a que cada mecanismo de sensado es independiente uno del otro, por lo tanto, cada uno va manteniendo el ultimo valor sensado. En el segundo prototipo que se presentará en la siguiente sección se podrá apreciar un mejor manejo de esta información.

5.2 Ejemplos de uso del *Prototipo 2*

En esta sección se presentará una descripción del *Prototipo 2*. Supongamos que arrancamos el prototipo con el GPS y los *beacons* desconectados. En la Figura 5.6 se puede visualizar la pantalla que se recibe, donde se indica que el mecanismo de sensado es indefinido debido a que no hay ninguno disponible en este momento.

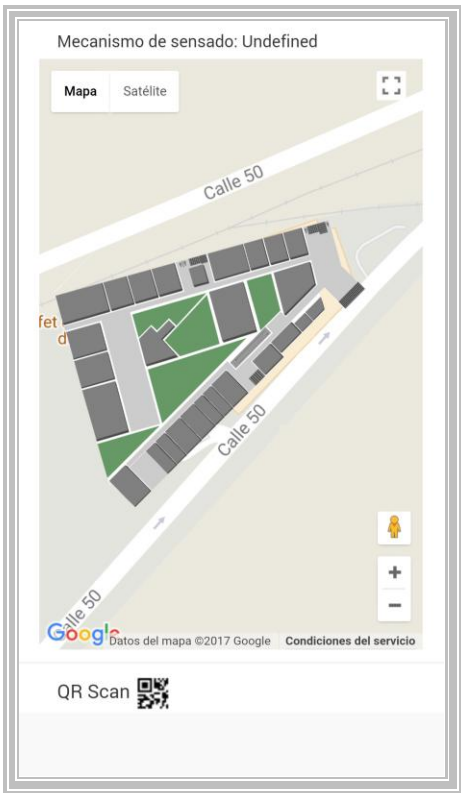

Figura 5.6: *Prototipo 2* – Pantalla de inicio con el GPS y *beacons* desconectados.

Supongamos que se enciende el GPS, dado que es el único mecanismo de sensado, este tiene prioridad y se muestra la posición del usuario, en este caso está en la puerta de la Facultad de Informática, como se puede visualizar en la Figura 5.7. Se puede apreciar en dicha figura que el mecanismo de sensado ahora se indica que es GPS.

Cabe mencionar que dependiendo de cuánto tarda el GPS en empezar a detectar los valores de los sensores, es el tiempo en que el usuario está a la espera de recibir su posición. Esto a veces puede tardar varios minutos.

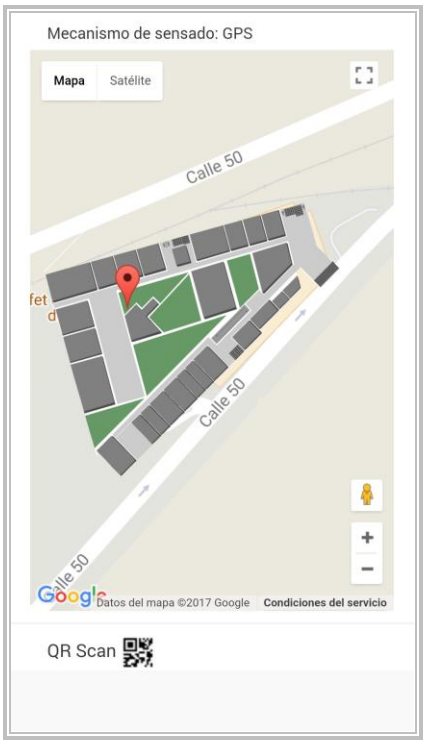

Figura 5.7: *Prototipo 2* – Se enciende el GPS.

Al encender los beacons, cuando el usuario es sensando por el beacon que está en el hall de entrada del edificio, se le actualiza la posición ya que este mecanismo tiene prioridad respecto al GPS. Esto se puede visualizar en la Figura 5.8

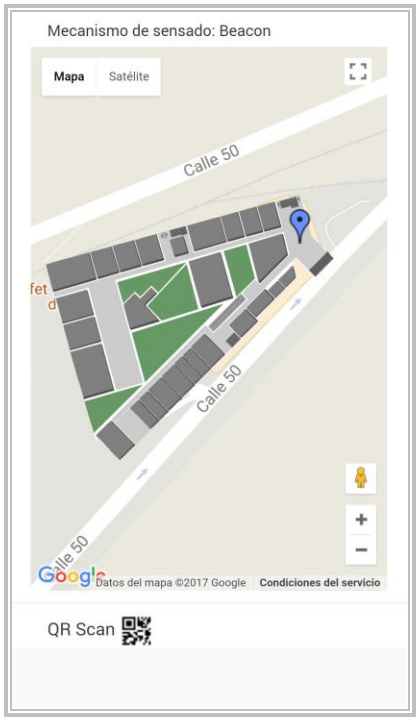

Figura 5.8: *Prototipo 2* – Se enciende los *beacons* y el usuario es sensado por uno de ellos.

Se puede apreciar en la Figura 5.8 que el mecanismo de sensado ahora se indica que es *Beacons*. Esto permite ir apreciando cuál es el mecanismo de sensado que está determinando la posición actual del usuario.

Supongamos que ahora el usuario decide leer un código QR, entonces se le abre el lector de QR, y al leer dicho código recibe una pantalla como se puede apreciar en la Figura 5.9. En dicha figura se puede observar que se le cambia la posición actual al usuario, esto se debe a que la lectura de códigos QR tiene prioridad sobre los otros mecanismos de sensado.

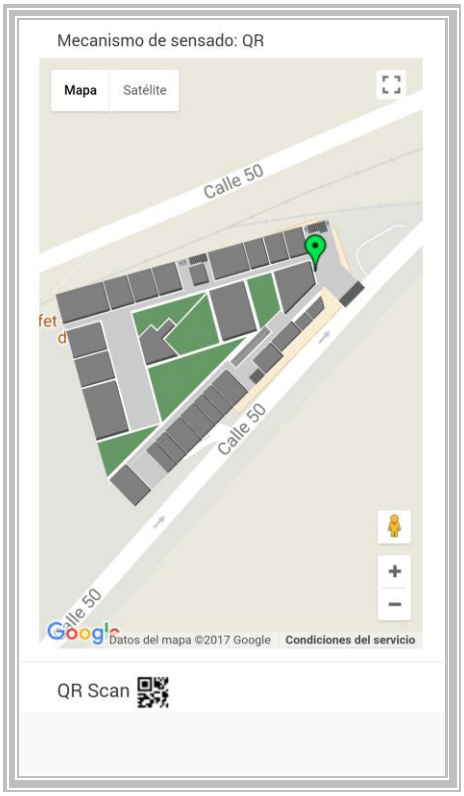

Figura 5.9: *Prototipo 2* – El usuario lee un código QR.

El usuario comienza a caminar por el hall de la facultad hasta la fotocopiadora (dónde está posicionado otro de los *beacons*). Supongamos que todavía no pasaron dos minutos desde que se leyó el último código QR. Por lo tanto, se le sigue mostrando al usuario todavía el código QR, como se mostró en la Figura 5.9.

En un momento dado, pasan los dos minutos entonces el usuario recibe la pantalla que se puede visualizar en la Figura 5.10, donde se le indica su nueva posición, la cual fue detectada mediante uno de los *beacons*.

Cabe mencionar que si bien se sigue recibiendo señal de GPS, en este caso, el beacon tiene prioridad y por dicha razón este determina la posición actual del usuario.

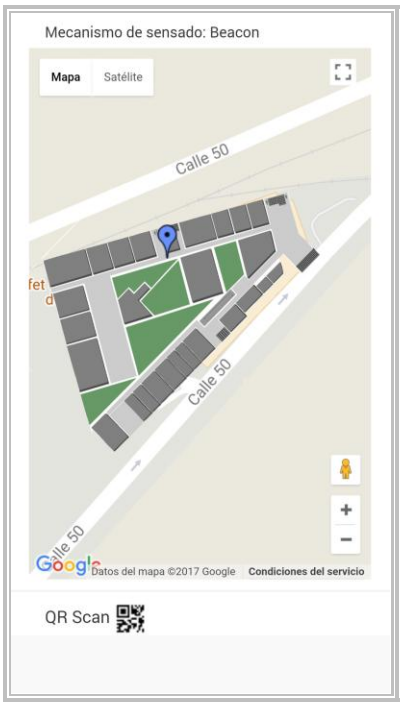

Figura 5.10: *Prototipo 2* – Pasaron dos minutos desde que se leyó el último código QR y el usuario es sensado por un *beacon*.

Supongamos que por alguna razón se interrumpe la señal del *beacon*, es decir se genera un "*timeout*" desde el *beacon*, en este caso, pasa a tener prioridad la señal del GPS, como se puede apreciar en la Figura 5.11.

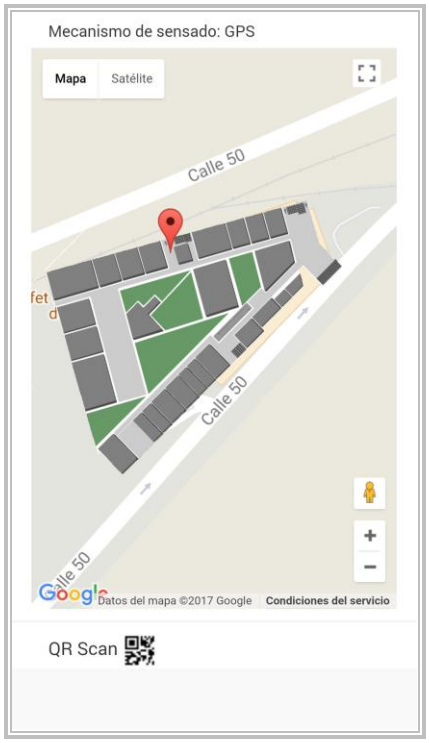

Figura 5.11: *Prototipo 2* – *Timeout* del beacon y se toma la posición del GPS.

6. Conclusiones y Trabajo Futuros

En esta tesina se presentó un framework para brindar soporte a aquellas aplicaciones móviles basadas en posicionamiento, en las cuales se tiene más de un mecanismo de sensado de posición funcionado simultáneamente. Este framework se presentó en el Capítulo 3, y se pudo apreciar en el Capítulo 4 como podía ser extendido para soportar distintos requerimientos relacionados con el procesamiento o combinación de los valores sensados.

En particular, se exploraron los mecanismos de sensado de posición, GPS, *Beacons* y códigos QR. Ésto se pudo apreciar en los dos prototipos presentados en el Capítulo 4, pero además los ejemplos de uso presentados en el Capítulo 5 permitieron apreciar como los mismos se comportan. Ambos prototipos fueron desarrollados usando *PhoneGap* junto con el pluign presentado en [Zimbello and Challiol, 2016].

El prototipo presentado en la Sección 4.1 muestra las posiciones de los tres mecanismos de sensado de manera independe entre sí, sin embargo, la mayor complejidad se dio al implementar el prototipo presentado en la Sección 4.2, donde se combinan estos mecanismos de sensado para mostrar una única posición al usuario.

Cabe mencionar que framework propuesto es genérico e independiente del dominio de la aplicación. Es decir, la solución propuesta brinda las bases para contar con la integración y combinación de mecanismos de sensado de posicionamiento para cualquier dominio.

Dado que configurar los mecanismos de sensados son tareas complejas y requieren conocimiento de las especificaciones técnicas de bajo nivel de cada uno de estos, se propuso una forma fácil de configurar estos mecanismos mediante la configuración de un archivo XML. Los detalles relacionados a esta configuración se presentaron en el Anexo D.

Los prototipos se desarrollaron usando *PhoneGap* [PhoneGap] y combinan simultáneamente el uso de GPS, *Beacons* y códigos QR para determinar la posición del usuario. Esto permite explorar cómo se comportan estos mecanismos funcionando simultáneamente, y cómo se pueden crear este tipo de aplicaciones de manera sencilla para el desarrollador, sin tener que indagar en aspectos técnicos de bajo nivel de cada mecanismo de sensado.

Al proponer un framework con estas características se espera que los desarrolladores de este tipo de aplicaciones se focalicen principalmente en los aspectos de dominio, heredando del framework aquellos aspectos relacionados con el sensado de posicionamiento. Para poder determinar la utilidad del mismo se deberá en un futuro ver cómo es usado por distintos desarrolladores. De la experiencia de la tesina, se pudo comprobar que el desarrollo del segundo prototipo (presentado en la Sección 4.2.2) fue más rápido dado que ya se contaba con mucha información de los sensores. El mayor tiempo fue destinado a definir tanto el *trasformador* como *dispatcher* específicos, como así también la interfaz que combina mapas indoor-outdoor.

A continuación se listarán algunos posibles trabajos futuros que se desprenden de la tesina presentada:

- Usos del framework por distintos desarrolladores, para poder determinar la utilidad del mismos y a partir de esto poderlo enriquecer para facilitar aún más a los desarrolladores de este tipo de aplicaciones.
- Analizar la incorporación de *transformadores* y *dipatcher* más complejos como parte del framework, anticipándose a cubrir los distintos requerimientos que podrían surgir en las aplicaciones móviles basadas en posicionamiento.
- Analizar cómo se podría desacoplar la heurística de combinación de sensores, esto no sólo impactaría en nuevas clases que deberían crearse en el framework, sino también en cómo esta información podría ser definida usando un archivo de configuración XML. Algunas consideraciones que deberían analizarse:
    - Si se tuvieran definidos los puntos de interés de la aplicación, esto permitiría, por ejemplo, determinar que un usuario está dentro de un edificio, en dicho caso, la prioridad la tienen los *beacons* y el código QR. Se podría tener una heurística que analiza que mientras el usuario está dentro del edificio, el dato recibido por GPS es desestimado.
    - Se podría considerar en una heurística la trayectoria que viene realizando un usuario, para en base a eso determinar a qué mecanismo de sensado le da prioridad.
- Analizar requerimientos especiales para dominios particulares, por ejemplo, en Anexos A y B, se probaron algunas características del framework en dominios puntuales, sin embargo se pudo determinar que cuando los beacons se usan para posicionar mesas de trabajos muy cercanas entre sí, no es un mecanismo que funcione correctamente. Acorde a ésto, también se podrían dar recomendaciones a los desarrolladores para que hagan un uso más eficiente y adecuado de cada mecanismo de sensado. Por ejemplo, definir los códigos QR genéricos para poder ser usados por distintas aplicaciones.
- Analizar e incorporar el manejo de mapas como parte de facilitar a los desarrolladores la construcción de este tipo de mecanismos de sensado. Como se menciona en [Challiol et al., 2017] esto también es un punto de variabilidad que podría ser reusado por aplicaciones de este estilo al igual que son los mecanismos de sensado de posición.
- Analizar y probar otros tipos de *beacons*, por ejemplo, los de posición que mencionan la gente de *Estimote*. No sólo podría impactar en el framework propuesto, dado que se debería analizar la posibilidad de crear una nueva abstracción de sensor (que extienda de *SensorBeacon*), sino también implica analizar cómo se debería implementar el *MangerListener* concreto para este tipo de *beacons*. Se podría rediseñar el framework para poder tener desacoplado el prototipo asociado a los sensores y así reusar para todos el mismo

MangerListener. De esta manera, el mismo manager podría tener conviviendo *beacons* con distintos protocolos.

- Analizar y probar el uso de la librería de *Google Place* para obtener la posición GPS del usuario, para esto se debería implementar un *MangerListener* concreto. Esto podría implicar analizar si dos *MangerListener* destinados a tomar el dato del GPS podrían ser utilizados simultáneamente o como entre ambos se podría brindar más precisión.
- Analizar la incorporación de otros mecanismos de sensado como puede ser utilizando redes Wifi o NFC, esto no solamente impacta en el framework propuesto, sino que habrá que analizar cómo esto puede ser implementado en *PhoneGap.*
- Al plantear el framework de manera general, el mismo podría ser usado no solamente para sensores físicos, sino también para sensores virtuales [Rivero-Rodriguez et. al, 2016]. Por ejemplo, se podría explorar como tomar información de redes sociales, para determinar por ejemplo, que amigos tiene cercanos a la posición actual del usuario, y manejar esta información como una característica de contexto particular.
- Otra característica a explorar a futuro, es como el framework propuesto se podría aplicar a sensores que no sólo tomen la posición del usuario, sino tener integrado el concepto de ambiente y sus datos contextuales.
- Analizar la posibilidad de construir el XML de configuración de los sensores in-situ, es decir, recorriendo y detectando por ejemplo, los *beacons* que se podrían tener en un edificio, o reusar códigos QR ya existentes. Para ello, se podría contar con una herramienta auxiliar que vaya descubriendo mecanismos de sensado disponibles, *beacons* o que estos sean especificados explícitamente en el caso de los códigos QR.

Bibliografía

[Abdullah, 2016] Abdullah, N. (2016). “Context Aware Application Using Beacons”. Senior Projects Spring 2016. 324. http://digitalcommons.bard.edu/senproj_s2016/324. Último acceso: 01/04/17

[Alegre et al., 2016] Alegre, U., Augusto, J.C., Clark, T.: Engineering context-aware systems and applications: A survey. Journal of Systems and Software 117, 55-83 (2016)

[AskApp] Vídeo de *Shelley Bernstein* (*Vice Director of Digital Engagement & Technology at the Brooklyn Museum)* cuenta la experiencia obtenida de proyectos anteriores hasta llegar a su reciente nuevo proyecto: la aplicación Ask. https://www.youtube.com/watch?v=ul5cWZCToxg (Último acceso 15/08/17).

[BeaconsEnAmsterdam] Vídeo sobre el impulso que le está dando *Google* a los beacons. En particular entre los minutos 6:46 hasta 7:52, hablan del uso de *Beacons* en *Amsterdam* y accesibilidad a los mismos por parte de los programadores. https://www.youtube.com/watch?v=C-9X-j-wDps&feature=youtu.be

[Bekkelien, 2012] Bekkelien, A., Deriaz, M., & Marchand-Maillet, S. (2012). Bluetooth indoor positioning. Master's thesis, University of Geneva.

[Bauer and Dey, 2016] Bauer, C., & Dey, A. K. (2016). Considering context in the design of intelligent systems: Current practices and suggestions for improvement. Journal of Systems and Software, 112, 26-47.

[Challiol et al., 2017] Challiol, C., Lliteras, A. B., & Gordillo, S. E. (2017). Diseño de aplicaciones móviles basadas en posicionamiento: un framework conceptual. In *XXIII Congreso Argentino de Ciencias de la Computación*, pp. 682-691.

[Cheng et al., 2016] Cheng, R. S., Hong, W. J., Wang, J. S., & Lin, K. W. (2016). Seamless guidance system combining GPS, BLE Beacon, and NFC technologies. Mobile Information Systems, 2016.

[Dey et al., 1997] Dey, A. K., Abowd, G. D., Pinkerton, M., & Wood, A. (1997). “Future Computing Environments. CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software”. In Proceedings of the 10th annual ACM symposium on User interface software and technology. ACM, pp. 75-76.

[Dongsoo et al., 2014] Han, D., Lee, S., & Kim, S. (2014). KAILOS: KAIST indoor locating system. In Proceedings of 2014 International Conference Indoor Positioning and Indoor Navigation (IPIN), IEEE, pp. 615-619.

[El-Rabbany, 2002] El-Rabbany, A. (2002). Introduction to GPS: the global positioning system. Artech house.

[Emmanouilidis et al., 2013] Emmanouilidis, C., Koutsiamanis, R. A., & Tasidou, A. (2013). Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. Journal of Network and Computer Applications, 36(1), 103-125.

[ExperienciaAutobusesBucarest] Proyecto de transporte público usando beacons http://newatlas.com/ibeacon-smart-public-transport-bucharest/37901/ (Último Acceso: 04/04/17).

[Estimote] Página de Estimote: https://estimote.com (Ultimo acceso 12-8-2017)

[Fahy and Clarke, 2004] Fahy, Patrick, and Siobhan Clarke. (2004). "CASS–a middleware for mobile context-aware applications." In Proceedings of Workshop on context awareness, MobiSys 2014.

[Fortier et al., 2007] Fortier, A., Challiol, C., Rossi, G., & Gordillo, S. (2007). Physical Hypermedia: a Context-Aware approach. In Proceedings of the CAiSE (Vol. 7, pp. 499-513).

[Fortier et al, 2010] Fortier, A., Rossi, G., Gordillo, S. E., & Challiol, C. (2010). Dealing with variability in context-aware mobile software. Journal of Systems and Software, 83(6), 915-936.

[Gamma et al., 1995] Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. Reading: Addison-Wesley, 49(120), 11.

[Hansen et al., 2012] Hansen, F. A., Kortbek, K. J., & Grønbæk, K. (2012) Mobile urban drama: interactive storytelling in real world environments. New Review of Hypermedia and Multimedia, 18(1-2), 63-89.

[Hui-Chun et al., 2010] Chu, H. C., Hwang, G. J., Tsai, C. C., & Tseng, J. C. (2010) "A two-tier test approach to developing location-aware mobile learning systems for natural science courses". Computers & Education 55 (4), pp. 1618-1627.

[Jepson et al., 2012] Igoe, T., Coleman, D., & Jepson, B. (2014). Beginning NFC: Near Field Communication with Arduino, Android, and PhoneGap. O'Reilly Media, Inc.

[Lai et al., 2010] Lai, Y. C., Han, F., Yeh, Y. H., Lai, C. N., & Szu, Y. C. (2010, June). A GPS navigation system with QR code decoding and friend positioning in smart phones. In Education Technology and Computer (ICETC), 2010 2nd International Conference on (Vol. 5, pp. V5-66). IEEE.

[Leonhardt, 1998] Leonhardt, U. (1998). Supporting location-awareness in open distributed systems. Tesis Doctoral. University of London. 1998.

[MuseoDeBrooklyn] Experiencia usando Beacons del museo de Brooklyn https://www.brooklynmuseum.org/community/blogosphere/2015/02/04/the-realities-of-installing-ibeacon-to-scale (Último Acceso: 04/04/17).

[Musi Gentile, 2015] Musi Gentile, F.A. (2015). Modelo de gestos considerando context. Tesina de Grado. Facultad de Informática, UNLP.

[NFCEjemplos] Página ejemplo uso de NFC http://destinonegocio.com/negocio-por-internet/nfc-cambia-escenario-pymes (Ultimo acceso 12-8-2017)

[Ning, 2015] Ning, J. (2016). An iBeacon-Based Location-Aware Advertising System (Master's thesis, University of Waterloo). JiaMin Ning (2016). An iBeacon-Based Location-Aware Advertising System. UWSpace. http://hdl.handle.net/10012/10254 Último acceso: 02/08/17

[PhoneGap] Página de *PhoneGap*: http://phonegap.com (Último acceso: 18/04/17).

[Pokémon Go] Página oficial del juego Pokémon Go http://pokemongo.nianticlabs.com/es/ (Último Acceso: 04/04/17).

[Rivero-Rodriguez et al., 2016] Rivero-Rodriguez, A., Pileggi, P., & Nykänen, O.A. (2016). Mobile Context-Aware Systems: Technologies, Resources and Applications. International Journal of Interactive Mobile Technologies, 10(2), pp. 25-32.

[Sneps-Sneppe and Namiot, 2016] Sneps-Sneppe, M., & Namiot, D. (2016). “On physical web models”. In Proceeding of International Siberian Conference on Control and Communications (SIBCON). IEEE, pp. 1-6.

[Soon, 2008] Soon, T. J. (2008). QR code. Synthesis Journal, 2008, 59-78.

[Suk-Hoon et al., 2017] Jung, S. H., Lee, G., & Han, D. (2017). Methods and Tools to Construct a Global Indoor Positioning System. IEEE Transactions on Systems, Man, and Cybernetics: Systems.

[Xanthopoulos and Xinogalos, 2016] Xanthopoulos, S., & Xinogalos, S. (2016). A Review on Location Based Services for Mobile Games. In Proceedings of the 20th Pan-Hellenic Conference on Informatics, ACM, p. 28.

[Xin-Yu et al., 2015] Lin, X. Y., Ho, T. W., Fang, C. C., Yen, Z. S., Yang, B. J., & Lai, F. (2015). A mobile indoor positioning system based on iBeacon technology. In Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE (pp. 4970-4973). IEEE.

[Waze] Página oficial de la aplicación Waze https://www.waze.com (Último Acceso: 04/04/17).

[Weyns et al., 2015] Weyns, D., Caporuscio, M., Vogel, B., Kurti, A.: Design for Sustainability = Runtime Adaptation ∪ Evolution. In: the 2015 European Conference on Software Architecture Workshops, pp. 62-69. ACM, New York (2015)

[Woolf , 1997] Woolf, B. (1997). Null object, Pattern languages of program design 3.

[Zimbello and Challiol, 2016] Zimbello, A., & Challiol, C. (2016): Experimentando un uso no tradicional de PhoneGap. In XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016), pp. 1016-1024.

Anexo A: Participación en el proyecto "*Eventos Temporales Posicionados en Espacios Indoor*"

El proyecto "*Eventos Temporales Posicionados en Espacios Indoor"* se dio en el marco de la convocatoria de alumnos a Proyecto de Desarrollo de Aplicaciones e Innovación (2015-2016), de la Facultad de Informática, UNLP. El autor de esta tesina formó parte del equipo de alumnos que integró este proyecto, y dado que esta participación permitió conocer sobre la tecnología *Beacons*, y este conocimiento sirvió de base para el planteo de la presente tesina, es que se describe a continuación los principales detalles de este proyecto a fin de poder mostrar en que consistió dicho aprendizaje.

En el marco del proyecto, se ha trabajado en distintos aspectos, los cuales se detallan a continuación:

- Modelado e implementación de *Eventos Temporales Posicionados*, se trabajo en el modelado de los conceptos relevantes, se decidió realizar la implementación usando *PhoneGap* para plasmar dicho modelo pero también poder usar tecnología estándar como HTML, CSS y JavaScript.
- Respecto del *posicionamiento* se decidió investigar el uso de *Beacon*, en particular se está trabajando con tres *Beacons*[33], se realizaron mediciones para lograr precisar la posición del usuario dentro de un área. Se integró este sensado de posicionamiento como parte del prototipo.
- Otro aspecto del prototipo es la *asistencia en la movilidad*, para lo cual se definió una librería implementada en JavaScript que permite visualizar distintos pisos de un edificio. Esta librería permite visualizar los eventos posicionados sobre un plano como así también la posición actual del usuario.
- Integración de los aspectos anteriormente mencionados, contando así con un prototipo funcional.

Para la II *Expo Ciencia y Tecnología (Octubre, 2016)* se mostró con un prototipo funcional que permite mostrar eventos temporales posicionados dentro del edificio de la Facultad de Informática. Los eventos del prototipo están estipulados para una fecha determinada, con lo cual, al cambiar la fecha del dispositivo se puede apreciar que los mismos varían.

Como se puede apreciar en la descripción de los resultados obtenidos del proyecto, la exploración del uso de *Beacons* como mecanismos de posicionamiento, aportó una base para el planteo de la presente tesina.

En la Figura A.1 se puede apreciar el banner presentado en la II *Expo Ciencia y Tecnología (Octubre, 2016).*

[33] Marca usada como parte del proyecto: https://www.onyxbeacon.com

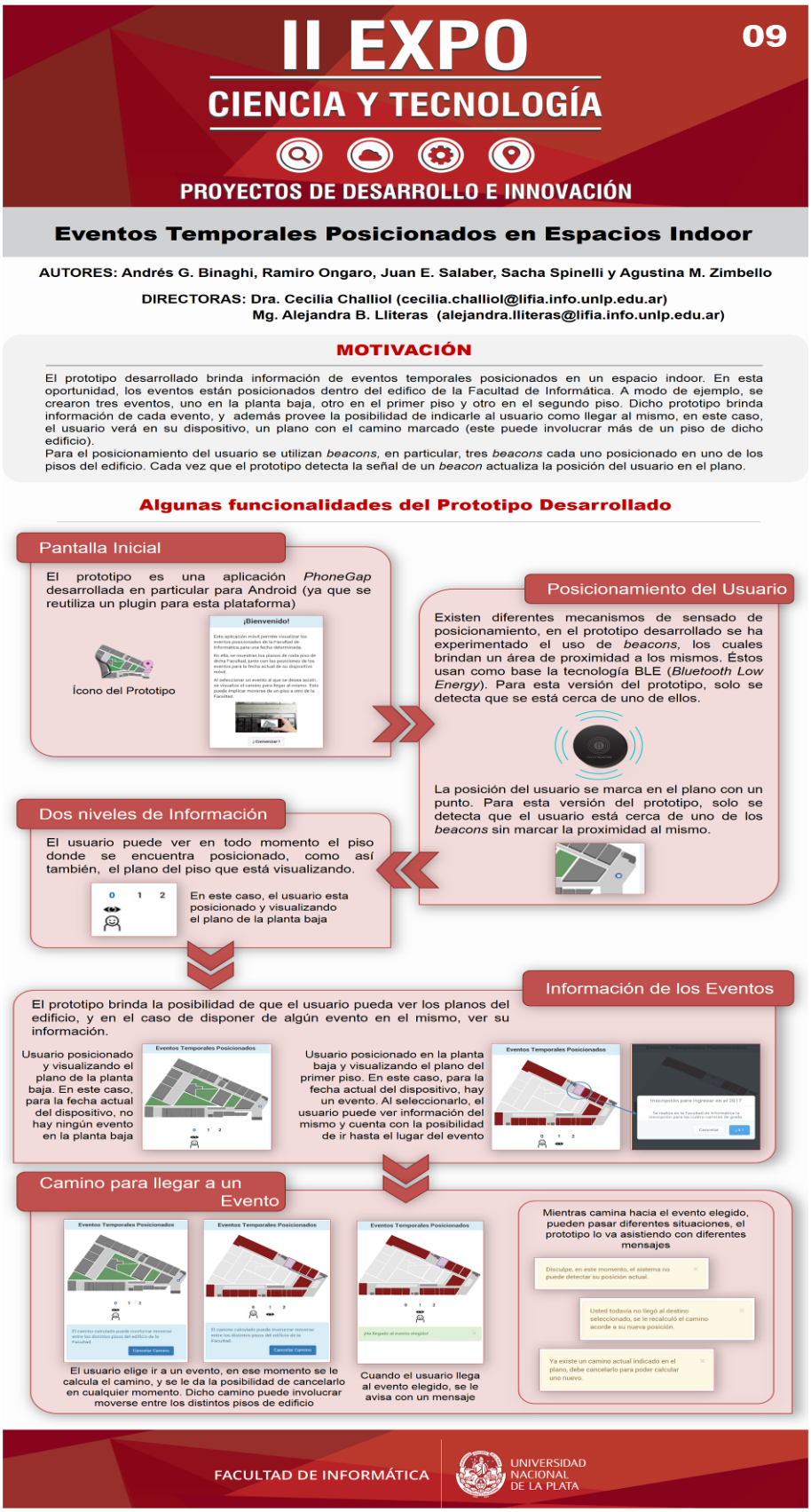

Figura A.1: Banner del Proyecto "*Eventos Temporales Posicionados en Espacios Indoor*"

Anexo B: Participación en el proyecto "*Actividades Educativas Posicionadas*"

El proyecto "*Actividades Educativas Posicionadas"* se dio en el marco de la convocatoria de alumnos a Proyecto de Desarrollo de Aplicaciones e Innovación (2016-2017), de la Facultad de Informática, UNLP. El autor de esta tesina formó parte del equipo de alumnos que integró este proyecto, y dado que el mismo permitió conocer sobre la tecnología de códigos QR y *Beacons*, y este conocimiento sirvió de base para el planteo de la presente tesina, es que se describe a continuación los principales detalles de este proyecto a fin de poder mostrar en que consistió dicho aprendizaje.

En el marco del proyecto, se ha trabajado en distintos aspectos, los cuales se detallan a continuación:

- Se obtuvo un prototipo para el dominio de *Actividades Educativas Posicionadas* que involucran elementos concretos. Donde los *Beacons* son usados para identificar que el alumno llega a una posición donde se encuentra una actividad educativa, este caso particular, una mesa con elementos concretos (cada uno con un código QR). Para lograr desacoplar las características físicas propias de los *Beacons* del uso de estos desde aplicaciones específicas (en este proyecto, las *Actividades Educativas Posicionadas*), se desarrolló una API para tal fin. La misma se usó en el prototipo desarrollado y se espera que sea escalable para configurar los *Beacons* en cualquier otro dominio. Esto forma parte de lo propuesto en esta tesina.
- Se realizaron distintas pruebas sistemáticas que permitieron identificar diferentes comportamientos de los *Beacons*. Por ejemplo, cómo afecta la orientación del celular o del *Beacon* a la señal recibida. Además, como para este domino puntual, los *Beacons* están ubicados sobre mesas, dependiendo de la cercanía de las mismas se genera superposición de señal. Esto genera que muchas veces se identifique que se está en una mesa, cuando en realidad se está en otra cercana.
- Se pudo identificar que dependiendo del dominio es la viabilidad del uso de *Beacons* como mecanismo de sensado. Por ejemplo, en el proyecto "*Eventos Temporales Posicionados en Espacios Indoor*" (Anexo A) no hubo conflicto de señal de los mismos, ya que el prototipo consideraba *Beacons* en diferentes pisos de un edificio. Sin embargo, en el dominio de *Actividades Educativas Posicionadas* cuando las mismas son muy cercanas, como por ejemplo en el caso de tener mesas en un aula, este tipo de sensado no es viable por conflicto de las señales entre los mismos.

Como se puede observar en este proyecto se profundizó sobre el funcionamiento de los Beacons, sobre todo en el conflicto de señales que producen los mismos. Además, esto se combinó con la lectura de códigos QR para poder identificar los elementos concretos.

En la Figura B.1 se puede apreciar el banner presentado en la III Expo Ciencia y Tecnología (Octubre, 2017).

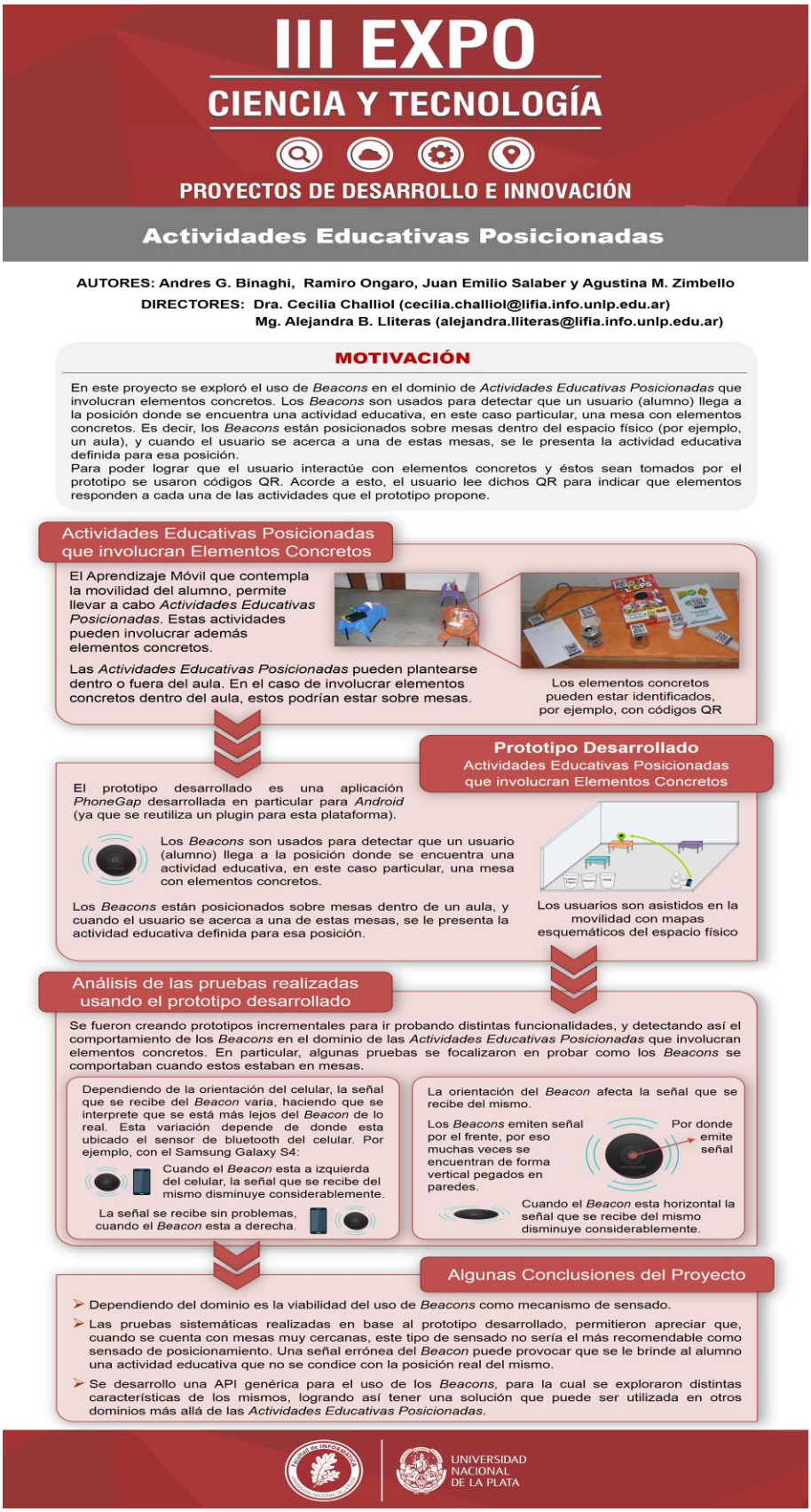

Figura B.1: Banner del Proyecto "Actividades Educativas Posicionadas"

Anexo C: Características principales de *PhoneGap* y el plugin definido en [Zimbello and Challiol, 2016]

En este Anexo se describirán las principales características que tiene un proyecto *PhoneGap* [PhoneGap], el cual se usó para el desarrollo del prototipo.

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*.

Si bien *PhoneGap* se puede utilizar como un framework híbrido multiplataforma, en esta tesina sólo se utilizó la versión para *Android.*

Cabe mencionar que *PhoneGap* provee el concepto de plugin para extender su funcionalidad. Cada plugins provee 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*. Como se detalla en [Zimbello and Challiol, 2016].

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[34] que pueden ser descargados y utilizados, por ejemplo, para acceder a la cámara del dispositivo o comunicarse con una base de datos.

En *PhoneGap* toda la lógica de control queda contenida en los *JavaScript* salvo que se delegue en un servidor. Este es el uso tradicional de *PhoneGap*, sin embargo, esto generaba pasar, por ejemplo, las clases del framework a código *JavaScript*. Para evitar esto, y seguir con la filosofía de clases definidas, se usó el plugin presentado en [Zimbello and Challiol, 2016], el cual permite invocar desde *Javascript* métodos específicos del framework y así recibir información para poder mostrar en la vistas HTML.

El plugin general presentado en [Zimbello and Challiol, 2016] 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 permite que pueda ser usado con cualquier modelo de dominio.

Se puede apreciar en la Figura C.1 el funcionamiento del plugin presentado en [Zimbello and Challiol, 2016], el cual es usado para invocar métodos puntuales de una clase de un modelo *Java*. En dicha figura se puede observar que la invocación se realiza con la siguiente información: *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

[34] Plugins de *PhoneGap*: https://build.phonegap.com/plugins (Último acceso: 12-11-2017)

en el método indicado, es una colección con la forma `<tipo,valor>`). Esta información que llega al plugin es usada con reflection[35] 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.

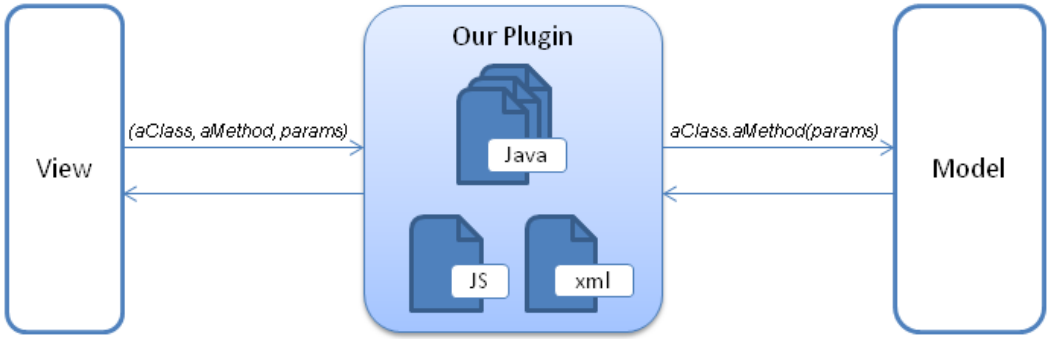

Figura C.1: Comunicación relacionada al plugin definido en [Zimbello and Challiol, 2016]

Desde los *Javascript* del prototipo se invocan los métodos del framework, en particular de la clase *AppSingleUser*. Y de esta manera se accede al punto de entrada del modelo.

[35] Para más información sobre reflection se puede consultar: https://docs.oracle.com/javase/tutorial/reflect (Último acceso: 16-11-2017).

Anexo D: Librería XML para agilizar la construcción de Aplicaciones Móviles basadas en Posicionamiento usando el framework propuesto

Para poder llegar a la versión de librería que actualmente se está usando en los prototipos, se fue explorando distintas opciones, las cuales se contarán en este Anexo, y se describirán los desafíos por los cuales se fue pasando.

Se comenzó explorando la librería *Xstream*[36] (versión 1.4.7), la cual sirve para serializar objetos en XML y de vuelta a objetos. Esto se exploró usando la versión de *Eclipse Neon*[37].

Se empezó realizando un análisis de como configurar la información que se quería instanciar entonces se decidió separar entre un archivo que tenga la información xml a instanciar y otro archivo que permita la especificación del significado de cada tag del archivo xml. Esto es tomado por *Xstream* para poder así instanciar la clase que corresponda. Entonces, se eligió que los archivos a ser leídos fueran configurados como se muestran a continuación:

```
archivoXML=src/configurationAndXML/example.xml
archivoConfiguracion=configurationAndXML/example
```

El path indicado como `archivoXML` tiene asignada la direccion del archivo xml que se desea levantar. Mientras que el path indicado como `archivoConfiguracion` asignada la dirección del archivo de configuración correspondiente al xml.

En particular, el archivo seteado como archivoConfiguracion tiene, por ejemplo, la siguiente forma:

```
model.PositionXYZ=PosXYZ
_model.PositionXYZ-x=X
_model.PositionXYZ-y=Y
_model.PositionXYZ-z=Z
```

Del lado izquierdo se especifica la clase del modelo, y con "_" se van indicando los atributos de dicha clase. Del lado derecho se especifica el nombre que tendrá el tag en el archivo XML, esto es usado por *Xstream* para leer el archivo .xml, y con esta configuración determinar que clases instanciar.

Para este ejemplo de configuración mostrada anteriormente, se podría tener el siguiente xml:

```
<PosXYZ>
      <X>100</X>
      <Y>50</Y>
      <Z>1</Z>
</PosXYZ>
```

[36] Página de *Xstream*: http://x-stream.github.io (Último acceso: 16-11-2017)
[37] Página de *Eclipse Neon*: https://www.eclipse.org/neon (Último acceso: 16-11-2017)

Una vez especificado el framewok, se empezó a analizar cómo poder lograr tener la configuración del mismo, y cuál sería la forma que tendría que especificar el desarrollador en el xml. Al comenzar con este proceso se identificó que cada *observer* debía ser especificado, lo cual implicaba referenciar a un objeto ya creado, acorde a esto se empezó a detectar que esto se volvía cada vez más complejo. Entonces algo que estaba pensado para simplificar complejizaba la tarea del desarrollador.

En el Código D.1 se puede observar un ejemplo simplificado de cómo quedaba una parte del xml acorde al framework propuesto. En dicho código se puede visualizar el tag `<reference>` este es usado para indicar que el valor se saca de otro tag, con lo cual se indica el path donde esta ese tag xml.

```
<sensingConcerns>
  <entry>
    <string>QrConcern</string>
    <mobileExperience.model.sensing.SensingConcern>
      <dispatcher class="mobileExperience.model.sensing.dispatchers.SensingDispatchPass"/>
      <policity class="mobileExperience.model.sensing.politics.PushSensingPolicity">
        <observers>
          <mobileExperience.model.sensing.SensingConcern reference="../../.."/>
        </observers>
        <sensor class="mobileExperience.model.sensing.sensors.AumentedSensor">
          <observers>
            <mobileExperience.model.sensing.politics.PushSensingPolicity reference="../../.."/>
          </observers>
          <idSensor>QrSensorAumented</idSensor>
          <TAG>AumentedSensor</TAG>
          <dynamicFeatures>
            <entry>
              <string>positionXYZ</string>
              <mobileExperience.model.sensing.sensors.DynamicFeature>
                <key>positionXYZ</key>
                <value class="mobileExperience.model.position.PositionXYZ">
                  <x>100</x>
                  <y>50</y>
                  <z>1</z>
                </value>
              </mobileExperience.model.sensing.sensors.DynamicFeature>
            </entry>
          </dynamicFeatures>
          <sensor class="mobileExperience.model.sensing.sensors.QrSensor">
            <observers>
              <mobileExperience.model.sensing.sensors.AumentedSensor reference="../../.."/>
            </observers>
            <idSensor>qrSensor</idSensor>
          </sensor>
        </sensor>
      </policity>
```

Código D.1: Ejemplo simplificado de archivo xml para ser parseado por *Xstream*.

A partir del Código D.1 se aprecia la complejidad que implicaba para el desarrollador usar un archivo de configuración de este estilo, por esta razón se decidió no usar *Xstream* y generar un parseador propio, donde ciertos tags ya tuvieran toda una lógica asociada y así simplificar la tarea del desarrollador. Un ejemplo de los tags usados en el xml de configuración que finalmente usan los prototipos pudo observarse en el Código 4.1.

En el Código D.2 se puede apreciar una simplificación del archivo xml para el *Prototipo 1*. Se puede observar que se cuenta con tag xml ya definidos y el desarrollador los usa y va indicando valores particulares, esto simplifica mucho la tarea respecto de lo que era la configuración de *Xstream* mostrada en el Código D.1.

```
<app>
    <concern>
        <contextFeature>qrValue</contextFeature>
        <methodology>transformXYZ</methodology>
        <dispatcher>dispatcherPass</dispatcher>
        <politic>push</politic>
        <aumentedSensorQR>
            <feature><key>positionXYZ</key>
                <value><positionXYZ>1240,750,0</positionXYZ></value>
            </feature>
            <id>qr1</id>
        </aumentedSensorQR>
    </concern>
    <concern>
        <contextFeature>beaconValue</contextFeature>
        <methodology>transformCircleArea</methodology>
        <dispatcher>dispatcherDistinctValue</dispatcher>
        <politic>push</politic>
        <aumentedSensorBeacon>
            <feature><key>positionXYZ</key>
                <value><positionXYZ>1260,670,0</positionXYZ></value>
            </feature>
            <id>87/58262</id>
        </aumentedSensorBeacon>
    </concern>
    <concern>
        <contextFeature>gpsValue</contextFeature>
        <methodology>transformGPSValueToLat-Long</methodology>
        <dispatcher>dispatcherDistinctValue</dispatcher>
        <politic>push</politic>
        <sensorGpsAndroid/>
    </concern>
</app>
```

Código D.2: Archivo xml simplificado para el *Prototipo 1.*

En el Código D.2 se simplifica el archivo para mostrar una configuración para cada uno de los mecanismos de sensados (código QR, *beacon* y GPS), se puede apreciar que dichos valores se corresponden con las instancias mostrados en la Figura 4.3 (de la Sección 4.2.1).

Para cada *SensingConcern* se define en el siguiente orden: la característica de contexto, el transformador, el dispatcher, la política de sensado, y el sensor que tendrá dicha política, el cual puede ser un sensor aumentado.

En el Código D.3 se puede apreciar una simplificación del archivo xml para el *Prototipo 2,* en el cual se muestra una configuración para cada uno de los mecanismos de sensados

(código QR, *beacon* y GPS). Dichos valores se corresponden con las instancias mostradas en la Figura 4.8 (de la Sección 4.2.2).

```
<app>
    <concern>
        <contextFeature>qrValue</contextFeature>
        <tranformer>valueCombinedSensorTransformer</tranformer>
        <dispatcher>dispatcherCombinedSensor</dispatcher>
        <politic>push</politic>
        <aumentedSensorQR>
            <feature><key>LatLong</key>
                <value><latLongPosition>-34.90348,-57.93754</latLongPosition></value>
            </feature>
            <id>qr1</id>
        </aumentedSensorQR>
    </concern>
    <concern>
        <contextFeature>beaconValue</contextFeature>
        <tranformer>valueCombinedSensorTransformer</tranformer>
        <dispatcher>dispatcherCombinedSensor</dispatcher>
        <politic>push</politic>
        <aumentedSensorBeacon>
            <feature><key>LatLong</key>
                <value><latLongPosition>-34.90338,-57.93765</latLongPosition></value>
            </feature>
            <id>87/58262</id>
        </aumentedSensorBeacon>
    </concern>
    <concern>
        <contextFeature>gpsValue</contextFeature>
        <tranformer>valueCombinedSensorTransformer</tranformer>
        <dispatcher>dispatcherCombinedSensor</dispatcher>
        <politic>push</politic>
        <sensorGpsAndroid/>
    </concern>
</app>
```

Código D.3: Archivo xml simplificado para el *Prototipo 2.*

A partir de los Códigos D.2 y D.3 se pudieron apreciar los diferentes tags que se pueden usar en el archivo de configuración. Reducir la complejidad del archivo xml, implicó que por cada tag se tiene implementada la lógica relacionada al mismo para lograr a partir de su parseo la instanciación correspondiente. A continuación se brindan detalles de la interpretación que se le da a cada tag.

Se asume que se crea una instancia de la clase *AppSingleUser* cada vez que se inicia uno de los prototipos. Al parsear el archivo de configuración, se identifican primero los *SensingConcern*, esto se realiza a partir del tag `<concern></concern>` dentro del mismo se define con el siguiente orden:

- *La característica de contexto*

  Para indicar la característica de contexto se usa el tag `<contextFeature></contextFeature>` donde sólo se debe agregar dentro del mismo el *Key* de dicha característica, el parser asume que es una característica

del usuario y se lo agrega al mismo, más allá de también estar referenciado por el *SensingConcern.*

- *El transformador*

  Para indicar el transformador usa el tag **<methodology></methodology>** cuando se asume que se va a usar por default el *ValueSensorTransformer*, en este caso, solo se debe indicar la metodología que utilizará el mismo para transformar. Dentro del tag **<methodology>** se pueden especificar los siguientes valores:

  - `transformXYZ`,
  - `transformCircleArea`
  - `transformGPSValueToLat-Long`

  Al parsear estos valores se crean la instancia del *transformador* correspondiente, y la misma se setea en un *ValueSensorTransformer*, el cual queda relacionado al *SensingConcern.*

  En el caso de requerir configurar el *transformador* con una instancia de *ValueCombinedSensorTransformer,* se usa el tag **<tranformer></tranformer>** con el valor `valueCombinedSensorTransformer`. Al parsear este valor se crean la instancia correspondiente, y esta queda relacionada al *SensingConcern.*

- *El disptcher*

  Para indicar el *dispatcher* usa el tag **<dispatcher></dispatcher>** dentro del mismo se pueden especificar los siguientes valores:

  - `dispatcherPass`
  - `dispatcherDistinctValue`

  Al parsear estos valores se crean la instancia del *dispatcher* correspondiente, y queda relacionada al *SensingConcern.*

- *La política de sensado*

  Para indicar la política de sensado usa el tag **<politic></politic>** dentro del mismo se pueden especificar los siguientes valores:

  - `push`
  - `pull`

  Al parsear estos valores se crean la instancia de las políticas de sensado correspondiente, y está queda relacionada al *SensingConcern.*

- *El sensor (de la política de sensado)*

  Para indicar el sensor se pueden usar los siguientes tags:

  - **<aumentedSensorQR></aumentedSensorQR>**
  - **<aumentedSensorBeacon></aumentedSensorBeacon>**
  - **<sensorGpsAndroid></sensorGpsAndroid>**

Dentro de los tags de los sensores aumentado se debe especificar las características dinámicas usando el tag `<feature></feature>` y el identificador del sensor usando el tag `<id></id>`. En el caso del sensor de GPS se brinda un identificador por default dado que el mismo es único. Tanto los códigos QR como *beacons* deben ser identificados para luego ser almacenados por el parser en el correspondiente *ManagerListener*.

Para cada caracterica dinámica se debe especificar la *key* usando el tag `<key></key>` y el *value* usando el tag `<value></value>`. Los valores que pueden tomar las caracterices dinámicas por ahora esta limitado a posiciones tanto del estilo (x,y,z) como (latitud, longitud), para esto se usan los siguientes tags:

- `<positionXYZ></positionXYZ>`
- `<latLongPosition></latLongPosition>`

Dentro de estos se debe especificar los valores que toman las instancias de las posiciones como se pudo apreciar en el Código D.2 y D.3.

Una vez parseado cada sensor, se crea una instancia del mismo y este queda asociado a la política de sensado.

Acorde a lo detallado anteriormente, se puede observar que detrás de cada tag especificado en el archivo de configuración se tiene una lógica de parseo, en el caso de requerir otro tipo de tag se deberá crea la interpretación del mismo y agregarlo como parte del parse.

De esta manera, se pudo simplificar la tarea de configurar mediante el uso de un archivo xml.

Anexo E: Protocolos relacionados a los *Beacons*

En este Anexo se describen los distintos protocolos existentes al momento de escribir esta tesina relacionados con la comunicación con los *beacons* físicos. Estos prototipos muchas veces sirven también como librerías y permiten decodificar el valor real sensado por un *beacon*.

A continuación se listan distintas librerías para acceder a los valores reales de los *beacons*, cada una de estas definen distintos protocolos:

- *EddyStone protcol (google)*: https://github.com/google/eddystone/blob/master/protocol-specification.md (Último acceso: 16-12-2017)
- *Ibeacon protocol (Apple):* https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf (Último acceso: 16-12-2017)
- *AltBeacon protocol* https://github.com/AltBeacon/spec (Último acceso: 16-12-2017)

En la Figura E.1 se puede apreciar como cada protocolo define distintos valores, estos son los que se usan para interpretar los valores sensados por los *beacons* físicos. Se pueden apreciar que estos protocolos están asociados a la clase *BeaconParser*, la cual en este caso es de la librería *AltBeacon* sin embargo esta clase puede ser configurada para usar los distintos protocolos mostrados en la Figura E.1. Esto es posible porque *AltBeacon* es general para cualquier beaons por eso ofrece la flexibilidad de combinarse con varios protocolos existentes.

org.altbeacon.beacon.BeaconParser

Modifier and Type	Constant Field	Value
public static final String	ALTBEACON_LAYOUT	"m:2-3=beac,i:4-19,i:20-21,i:22-23,p:24-24,d:25-25"
public static final String	EDDYSTONE_TLM_LAYOUT	"x,s:0-1=feaa,m:2-2=20,d:3-3,d:4-5,d:6-7,d:8-11,d:12-15"
public static final String	EDDYSTONE_UID_LAYOUT	"s:0-1=feaa,m:2-2=00,p:3-3:-41,i:4-13,i:14-19"
public static final String	EDDYSTONE_URL_LAYOUT	"s:0-1=feaa,m:2-2=10,p:3-3:-41,i:4-21v"
public static final String	URI_BEACON_LAYOUT	"s:0-1=fed8,m:2-2=00,p:3-3:-41,i:4-21v"

Figura E.1: Especificaciones de los diferentes protocolos[38].

En la Figura E.1 que hay distintos protocolos, sin embargo, muchas marcas de *beacons* crean sus propios protocolos para aprovechar mejor el funcionamiento de sus propios *beacons*. Por ejemplo, *Estimote* tiene su propio protocolo[39].

Cabe mencionar que en el caso de los prototipos implementados, se utilizó *AltBeacon* por la ventaja de ser open source y poder así explorar mejor el funcionamiento interno del mismo.

[38] Imagen tomada de la siguiente página: https://altbeacon.github.io/android-beacon-library/javadoc/org/altbeacon/beacon/BeaconParser.html (Último acceso: 16-11-2017)

[39] Información sobre los prototipos relacionados a los beacons: https://community.estimote.com/hc/en-us/articles/208546097-What-is-a-beacon-protocol-Can-beacons-broadcast-multiple-packets-simultaneously- (Último acceso: 16-11-2017)

Los *beacons* usados para los prototipos son de la marca *Onyx Beacons*, los cuales proveen su propia SDK y protocolo sin embargo el mismo es cerrado, por esta razón no se optó por utilizarlo en los prototipos implementados.