UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Implementación de un Prototipo de Aplicación Web Móvil Sensible al Contexto
Autores: Armendariz, Ignacio Enrique – Ecclesia, Sebastian Gabriel
Director: Dra. Silvia Gordillo
Codirector: Dra. Cecilia Challiol
Asesor profesional: -
Carrera: Licenciatura en Informática – Plan 2003

Resumen

El objetivo de este trabajo es investigar sobre los aspectos relacionados con los cambios que se deben registrar en una Aplicación Web Móvil sensible al contexto, y cómo estos afectan el comportamiento de la aplicación. Se implementó un prototipo de aplicación web móvil sensible al contexto, para analizar las diferentes características que debe tener la misma.
En el prototipo implementado se relevan los datos de los sensores que interesan considerar y se captan los cambios producidos en los valores sensados desde el dispositivo móvil para que se reflejen dichos cambios de manera automática en una Aplicación Web Móvil. Esto involucra implementar el mecanismo que refleje automáticamente el cambio en los valores de los sensores. A partir del prototipo implementado se logra una caracterización de este tipo de aplicaciones. Algunos de los sensores utilizados en el prototipo son: GPS, acelerómetro, giroscopio, sensor de luz y temperatura. El prototipo se implementó en el dominio turístico, es decir, al cambiar los datos de los sensores se actualiza la información turística al usuario, por ejemplo, mostrando los puntos de interés en su rango de visión (acorde a su posición actual y su orientación).

Palabras Claves

Aplicación Web Móvil Sensible al Contexto, Contexto, Sensores (GPS, Acelerómetro, Temperatura, Luz Giroscopio, Orientación, Dirección), Servicios Web, Spring, Android, JSF, Primefaces.

Trabajos Realizados

Análisis inicial sobre los distintos tipos de mecanismos para el acceso al contexto.
Planteo de una solución general para tomar los datos de contexto y reflejarlos en los dispositivos móviles de los usuarios.
Implementación de un prototipo funcional que se divide en dos aplicaciones: Aplicación Móvil Nativa Android y Aplicación Web JEE
Simulación del comportamiento de la aplicación en diferentes contextos y análisis de los resultados.

Conclusiones

La investigación fue realizada para brindar una solución eficiente a la hora de obtener contextos (por ejemplo, a través de sensores) y hacer que la aplicación se comporte de una manera distinta acorde al valor de los mismos. La solución general propuesta tiene la ventaja de poder adaptarse a los cambios tecnológicos que vayan surgiendo con el paso del tiempo. El prototipo permitió verificar como utilizar distintos valores de sensores para brindar diferentes respuestas.

Trabajos Futuros

Si bien el prototipo fue desarrollado para un dominio turístico, la solución general detallada en esta tesis puede ser aplicada a cualquier dominio. En el prototipo planteado se consideraron algunos contextos provenientes de los sensores del dispositivo y solo afectaban a un usuario. Algunas extensiones respecto de este tema podrían ser considerar otros contextos más complejos. También se podría modificar el funcionamiento del prototipo agregando más usuarios, para poder detectar la eficiencia del Servidor respecto al tiempo de respuesta entre que recibe un cambio y lo refleja en el usuario.

Fecha de la presentación: Junio 2015

Implementación de un Prototipo de Aplicación Web Móvil Sensible al Contexto

Índice

Índice de figuras

Capítulo 1: Introducción

1.1 Descripción general de los temas

Una aplicación web móvil es una aplicación que se ejecuta en un dispositivo móvil, por ejemplo, un celular. Si a este concepto se le agrega la posibilidad de poder responder al entorno donde se está ejecutando, tal aplicación es sensible al contexto.

A continuación se mencionan algunas definiciones para entender mejor a que se refiere el concepto de contexto. En [Dey,2000] se define "*El contexto es cualquier información que pueda ser utilizada para caracterizar la situación de una entidad. Una entidad puede ser una persona, un lugar o un objeto que es considerado relevante para la interacción entre el usuario y una aplicación, eventualmente incluyendo al usuario y a la aplicación misma*". *En* [Schilit, 1994] se define "*Las aplicaciones sensibles al contexto son aplicaciones que cambian dinámicamente o adaptan su comportamiento basándose en el contexto de la aplicación o del usuario*".

En la actualidad, aun con los avances tecnológicos, todavía en las aplicaciones web móviles no es directo el acceso a los datos de los sensores internos de los dispositivos para brindar información sensible al contexto. Estos sensores internos se utilizan para obtener información del dispositivo y además, detectar cambios en el contexto, por ejemplo, la orientación, el posicionamiento, temperatura, dirección, etc.

En [Espada et al., 2012a] y [Espada et al., 2012b] se propone una aplicación nativa que tiene un browser embebido para poder lograr una aplicación Web móvil sensible al contexto. La aplicación nativa es la que accede a los datos de los sensores y encapsula los requerimientos web. Esto tiene la limitante, y es que no se trabaja con una aplicación web pura del lado del cliente sino que implementan su propio browser.

Lo antes descripto motiva el presente trabajo, en el cual se busca lograr un prototipo de aplicación web móvil que corra en un browser tradicional del dispositivo y se pueda actualizar automáticamente ante los cambios de los valores de los sensores.

Si bien HTML5 especifica que tendrá acceso a los sensores internos de los dispositivos, en la actualidad se puede acceder solo a los datos del GPS, razón por la cual, para acceder a los datos de los sensores se contará con una aplicación nativa que enviará los datos directos a un servidor, que luego actualizará la aplicación web (sensible al contexto) en cuestión. Esta aplicación de toma de los datos de los sensores se desarrollará utilizando Android.

1.2. Objetivo

Investigar sobre los aspectos relacionados con los cambios que se deben registrar en una aplicación web móvil sensible al contexto y cuándo en el dispositivo móvil se sensa información de contexto. Respondiendo mediante eventos a los cambios que se produzcan en su entorno.

Se pretende además, proponer algunas mejoras a las arquitecturas descriptas en [Espada et al., 2012a] y [Espada et al., 2012b] que abordan la misma problemática.

Para ello se implementó un prototipo de aplicación web móvil sensible al contexto que permite estudiar las diferentes características que debe tener el mismo. Éste prototipo responde a los *estímulos* generados por los sensores de movimiento, ubicación, orientación, luz y temperatura actualizando la aplicación web en tiempo real.

El prototipo fue implementado en el dominio turístico y se muestra mediante un ejemplo concreto la información que recibe el usuario a medida que lo utiliza. Acorde a los datos del contexto, la aplicación web móvil se comporta distinta, brindando así información contextual.

1.3. Organización de la tesis

El documento de tesis está dividido en seis capítulos que abarcan desde las definiciones técnicas hasta la descripción de la arquitectura y el prototipo implementado.

En el **Capítulo 2** se definen y se desarrollan las especificaciones técnicas y las diferentes tecnologías a tener en cuenta como conocimiento previo y para un mejor entendimiento de los capítulos siguientes.

En el **Capítulo 3** se describe en forma detallada el problema a solucionar, la arquitectura propuesta y su comparación con otras arquitecturas similares para el mismo propósito. También se describe la interacción entre el dispositivo móvil, el servidor y el navegador web.

El **Capítulo 4** explica cómo fue implementado el prototipo de la aplicación sensible al contexto. Se detalla su funcionamiento, las herramientas utilizadas para su desarrollo y las diferentes partes que lo conforman.

La demostración de cómo funciona el prototipo se define en el **Capítulo 5**,

exponiendo mediante imágenes los diferentes casos de prueba con sus datos de entrada específicos, y mostrando los resultados obtenidos durante la ejecución de dichas pruebas.

En el **Capítulo** 6 se detallan las Conclusiones y Trabajos Futuros.

Capítulo 2: Background

2.1. Aplicaciones Móviles

Una aplicación móvil [Stüber, 2012] es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS, BlackBerry OS, Windows Phone, entre otros.

El desarrollo de aplicaciones para dispositivos móviles requiere tener en cuenta las limitaciones de estos dispositivos. Los dispositivos móviles funcionan con batería y tienen procesadores menos poderosos que los ordenadores personales. Los desarrollos de estas aplicaciones también tienen que considerar una gran variedad de tamaños de pantalla, datos específicos de software y configuraciones. El desarrollo de aplicaciones móviles requiere el uso de entorno de desarrollo integrado (IDE).

No todas las aplicaciones funcionan en todos los aparatos móviles. Cuando se dispone de cierto dispositivo móvil, se debe tener en cuenta el sistema operativo y el tipo de aplicaciones que corresponde a ese aparato. Los sistemas operativos móviles Android, Apple, Microsoft y BlackBerry tienen tiendas de aplicaciones que operan en línea en las cuales se pueden buscar, descargar e instalar las aplicaciones. Algunos comerciantes minoristas también operan tiendas de aplicaciones en internet. Únicamente se podrá utilizar “la tienda” que ofrezca las aplicaciones que sean compatibles con el sistema operativo del dispositivo en cuestión.

En su mayoría, las aplicaciones móviles suelen ser de propósito más específico y acotadas en su funcionalidad, a diferencia de los sistemas o aplicaciones web y desktop, que se ejecutan en terminales más potentes y suelen ser más robustos en términos de funcionalidad, capacidad de cómputo y comunicación. Actualmente, gran parte de las aplicaciones más conocidas poseen una versión para dispositivos móviles.

Cuando se instala una aplicación en el móvil, es posible que se pida autorización para que se permita acceder a la información del dispositivo. Desde algunas aplicaciones se puede acceder, por ejemplo a:

- Lista de contactos de teléfono y de email.
- Registro de llamadas.
- Datos transmitidos por internet.
- Información del calendario.

- Datos de localización del dispositivo.
- Código de identificación exclusivo de su aparato.

Algunas aplicaciones móviles [Aplicaciones Móviles] solamente pueden acceder a los datos necesarios para su funcionamiento, y otras pueden acceder a datos que no están relacionados con el propósito de la aplicación.

Si mientras se usa el móvil, se está suministrando información, alguien puede recolectarla, ya sea el creador de la aplicación, la tienda de aplicaciones, un anunciante o una red de publicidad. Y de ser así, es posible que los datos sean compartidos con otras compañías. Hay algunas aplicaciones que usan datos específicos de localización para ofrecer mapas, cupones para tiendas cercanas, o información sobre personas que tal vez conozca y que se encuentran en las inmediaciones. Algunas aplicaciones suministran datos de localización a redes de publicidad que pueden combinarse con otra información almacenada en sus bases de datos para dirigir específicamente anuncios basados en sus intereses y su ubicación geográfica.

2.1.1 Aplicaciones Web Mobiles

Generalmente, las aplicaciones no residen totalmente en el dispositivo móvil, como suele ocurrir en la mayoría de las aplicaciones *desktop* o de escritorio. Sino que se trata de aplicaciones distribuidas que en su mayoría consta de una parte cliente y de una parte servidor.
Cuando una aplicación móvil requiere una conexión a Internet, existen dos caminos posibles:

- **Wi-fi (Wireless Fidelity)** [Stüber, 2012]**:** Es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con Wi-Fi, tales como: un ordenador personal, una consola de videojuegos, un smartphone o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Para poder usarlas es necesario estar dentro del rango de una red pública o privada. La mayoría de los puntos de conexión wifi de uso público – como los de las cafeterías, aeropuertos y hoteles – no codifican la información que se envía a través de internet y no son conexiones seguras.

- **Sistema de comunicaciones móviles** [Parsons, 1989]**:** Las comunicaciones móviles se dan cuando tanto el emisor como el receptor están, o pueden estar, en movimiento. La movilidad de estos dos elementos que se encuentran en los extremos de la comunicación hace que no sea factible la utilización de hilos (cables) para realizar la comunicación en dichos extremos. Por lo tanto utilizan básicamente la

comunicación vía radio. Algunos ejemplos de este tipo de comunicación son GSM (2G), GPRS (2.5G), UMTS (3G), etc.

La principal diferencia entre estos dos tipos de comunicaciones, es que cuando se utiliza una conexión a Internet de tipo Wi-fi, tanto la cantidad de datos como el ancho de banda no suele ser un problema, ya que generalmente detrás de este mecanismo de comunicación hay un servicio de banda ancha. En el lado opuesto se encuentran los sistemas de comunicaciones móviles provistos por la compañía telefónica. Éstos suelen tener restricciones respecto de la cantidad de datos transferidos y su ancho de banda, además de que pueden llegar a tener costos adicionales.

Es por esto que cuando se desarrolla una aplicación móvil que necesita conexión a Internet, se intenta minimizar la cantidad de datos transferidos cuando se utilizan los sistemas de comunicaciones provistos por las telefónicas.

2.1.2 Aplicaciones Sensibles al Contexto

Dey [Dey, 2000], dio una de las primeras definiciones de contexto, a continuación se indica como enuncio este concepto:

> *"El contexto es cualquier información que pueda ser utilizada para caracterizar la situación de una entidad. Una entidad puede ser una persona, un lugar o un objeto que es considerado relevante para la interacción entre el usuario y una aplicación, eventualmente incluyendo al usuario y a la aplicación misma."*

Otra defunción relévate en el área, es la de aplicaciones sensibles al contexto. Schilit [Schilit, 2004] las define de la siguiente manera:

> "*Las aplicaciones sensibles al contexto son aplicaciones que cambian dinámicamente o adaptan su comportamiento basándose en el contexto de la aplicación o del usuario.*"

Esta combinación de estímulos proveniente de sensores, se denomina contexto. Por ejemplo, una oficina puede tener una serie de sensores que miden la intensidad de la luz, cuyos valores pueden ser: oscuro, normal, brillante y el ruido con valores posibles: silencio, moderado y ruidoso. Partiendo de la combinación de estos datos, se podría obtener el contexto "horario_no_laboral", resultado de combinar intensidad de luz = oscuro y ruido = silencio. Una aplicación sensible al contexto podría estar programada para responder al contexto horario_no_laboral", bloqueando las puertas de la oficina, apagando luces, fotocopiadoras y demás aparatos que no deban

usarse en este contexto. Actualmente disponemos de equipos portátiles con recursos similares a las computadoras personales de escritorio.

El software, ha evolucionado notablemente desde los comienzos de la computación. En los primeros años se la consideraba parte del hardware, luego comenzó a tener entidad propia y después de la crisis del software se convirtió en una rama de la ingeniería. Sin embargo, el crecimiento del software como disciplina siempre ha estado por detrás del desarrollo del hardware.

Uno de los desafíos de la ingeniería de software moderna es aprovechar los recursos que aporta el hardware, en los cada vez más pequeños dispositivos móviles, transformándolos en servicios al usuario. Actualmente, aparatos tales como PDA y PalmTop, tienen prácticamente la misma configuración en hardware y software que una computadora de escritorio. Este hecho posibilitó que una persona pueda llevar su computadora personal a cuestas, aunque también se debió solucionar aspectos técnicos auxiliares. El problema que más restricciones imponía en el uso de cualquier dispositivo móvil, era la conexión a periféricos y/o redes mediante cables. El desarrollo de las conexiones inalámbricas fue de gran importancia para el uso generalizado de equipos portátiles, por ejemplo, teléfonos celulares, notebooks, etc.

Luego de tener la posibilidad de llevar la computadora personal en los viajes fuera de la oficina y poder conectarse a redes en forma inalámbrica, surgió otra necesidad en el usuario, esta es recibir la información necesaria en su dispositivo móvil, sin tener que pedirla explícitamente. Por ejemplo, si el usuario es un estudiante e ingresa a la facultad con su PDA, seguramente consultará en ésta horarios de clase, exámenes y otros aspectos relacionados al campo académico; en cambio, cuando esté con su PDA en un club, practicando su deporte favorito, probablemente consulte los días y horarios de los próximos torneos. La necesidad del usuario de obtener información relevante según la situación particular en la que se encuentre, motivó el desarrollo de las aplicaciones sensibles al contexto. Dado que el aporte más significativo de los dispositivos móviles es la capacidad de funcionar en diversos lugares, la primera necesidad a satisfacer fue adaptar la información brindada al usuario, dependiendo del lugar en el que éste se encuentre. De hecho las primeras aplicaciones sensibles al contexto fueron sensibles a la locación. Nuevamente, surgió otra necesidad por parte del usuario, que es recibir la información adecuada dependiendo no sólo del lugar donde está, sino también del momento, situación anímica, etc.

La creciente necesidad de adaptación obligó a replantear el diseño de las aplicaciones, reconociéndose como un tipo especial, dadas sus características distintivas.
En los primeros momentos de las aplicaciones sensibles al contexto, la adaptación necesaria se hacía modificando el comportamiento de los objetos que componían la

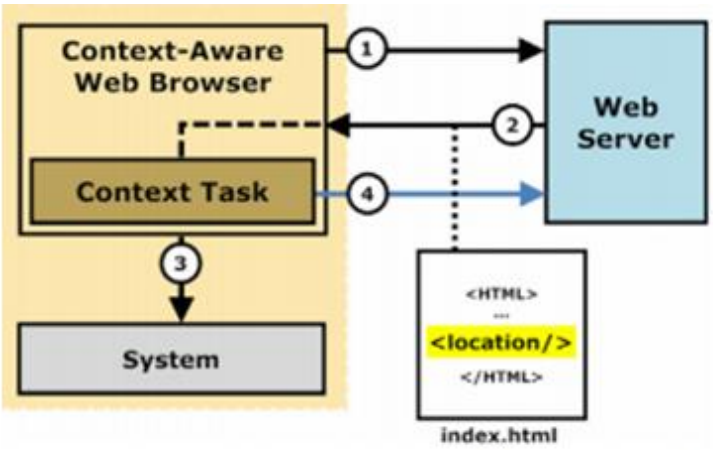
aplicación propiamente dicha. Por ejemplo, en un sistema administrativo de una facultad, seguramente existirá una entidad alumno. Para acomodar las distintas conductas que debe tomar el sistema cuando el alumno cambia de locación, se cambia el código, agregándole decisiones en los programas. Este paso tedioso debía realizarse cada vez que se requería que el sistema reaccione a más situaciones, como podría ser el aspecto de contexto temporal.

Uno de los pasos hacia delante más notables, desde el punto de vista del diseño, fue separar la conducta sensible al contexto de la conducta propia de la aplicación. De este modo ambos tipos de aplicaciones pueden evolucionar en forma independiente.

La tendencia actual en aplicaciones sensibles al contexto, es dotarlas de la capacidad de reaccionar a aspectos de contexto no determinísticos, de una manera similar a cualquier otro aspecto de contexto.

La capacidad de capturar la información de contexto es una de las debilidades de las aplicaciones web en comparación con las aplicaciones móviles nativas [Gossweiler et al., 2011]. Los navegadores web para móviles tradicionales (Chrome, Opera Mini, Firefox, Safari, etc) no tienen ningún mecanismo para permitir que las aplicaciones web usen los sensores del dispositivo para capturar información de contexto y enviar esta información al servidor web. La introducción de la información de contexto en las aplicaciones web ofrece nuevas posibilidades para el desarrollo, dando la posibilidad de implementar algunas de las características que antes eran exclusivos de las aplicaciones nativas.

En [Espada et al., 2012a] y [Espada et al., 2012b] se propone una posible implementación de una aplicación sensible al contexto. Consiste en una aplicación nativa que tiene un browser embebido, accede a los datos de los sensores y encapsula los requerimientos web. Esto tiene la limitante que no se trabaja con una aplicación web pura del lado del cliente. Para este caso, sería un esquema como el que se muestra en la Figura 1.



(1) El navegador web sensible al contexto hace una petición a un servidor web.
(2) El servidor web responde una página HTML que contiene información contextual (etiquetas XML)
(3) El navegador web sensible al contexto procesa la página y ejecuta las correspondientes tareas de contexto.
(4) El resultado de la tarea se envía al servidor mediante un servicio web.

Figura 1: Descripción de la implementación propuesta en [Espada et al., 2012b]

Lo propuesto por los autores en [Espada et al., 2012b] es tener una aplicación nativa con un browser embebido, esto quiere decir que no se está trabajando con un browser tradicional sino con una aplicación que muestra información en dicho browser. Como desventaja, no se está manteniendo el concepto de multiplataforma que caracteriza la tecnología tradicional web.

2.1.3 Mecanismos de Sensado

El sensado tiene como objetivo tomar los datos provenientes de sensores y convertirlos en objetos que puedan ser procesados. Los sensores son los objetos que recogen valores directamente del ambiente (contexto).
Existen diferentes tipos de sensores mediante los cuales se pueden obtener datos para procesar [Albaladejo Da Silva, 2011]:

- **Acelerómetro:** Quizá el más conocido de todos los sensores es el famoso acelerómetro. Esta pequeña pieza de hardware se encuentra ya en casi todos los teléfonos inteligentes y viene en dos tipos: acelerómetro de dos dimensiones y de tres dimensiones. La diferencia entre éstos no sólo radica en su estructura, sino en sus capacidades.
  El más robusto de ambos es el acelerómetro de 3 dimensiones y es el más ampliamente utilizado. Así, este sensor puede entender la posición de nuestro dispositivo fácilmente para, por ejemplo, cambiar la pantalla de posición horizontal a posición vertical dependiendo de cómo lo sostengamos.
  Es lógicamente uno de los sensores más utilizados por las aplicaciones, ya que permite leer en modo vertical u horizontal entre diversas aplicaciones. Para la navegación web a veces es más fácil leer en modo horizontal por el

tamaño de las letras, mientras que algunos desarrolladores bloquean el cambio de posición para que sus aplicaciones tengan una solo forma de visualizarse.

- **Sensor de luz ambiental:** El sensor de luz también es una pequeña construcción de hardware que básicamente detecta la intensidad de la luz del ambiente para ajustar el brillo de la pantalla. De este modo, si por ejemplo el dispositivo está en un cuarto oscuro, este sensor ajustará la intensidad de la luz de la pantalla para que disminuya su brillo. En cambio, en condiciones de luz ambiental muy fuerte, el sensor detectará esto para ajustar el brillo incrementando su intensidad.

- **Sensor de proximidad:** Este pequeño sensor basado en hardware puede medir la distancia que existe entre el equipo y algún otro objeto, como por ejemplo nuestro cuerpo. Así, cuando acercamos el equipo a nuestra oreja para contestar una llamada, el dispositivo apaga la luz de la pantalla para ahorrar batería.
  Pero no es lo único. En la misma situación descrita, si tocamos la pantalla por accidente mientras estamos en una llamada, no pasará nada porque el sensor de proximidad bloqueará la pantalla para evitar este tipo de errores. Sin embargo, cuando alejamos el equipo de nuestra oreja, automáticamente la pantalla volverá a ser funcional.

- **Sensor magnético:** El sensor magnético, que a veces recibe también el nombre de brújula digital, es precisamente un sensor que funciona ubicando al equipo con respecto a los polos de la tierra y nos permite saber la dirección hacia la que se encuentra el Norte del campo magnético más fuerte. Se suelen utilizar para obtener la orientación respecto al Norte magnético terrestre. Tienen mucha utilidad en robots, cuadricópteros o similares, para la navegación autónoma o para devolver datos de orientación a distancia.
  Existen sensores de dos y tres ejes, estos últimos se suelen utilizar para dispositivos voladores.

- **Giroscopio:** Este sensor se encarga de medir el giro de un dispositivo en dirección diagonal gracias a la aceleración angular, algo de lo que no es capaz por sí solo el acelerómetro. Juntos, el acelerómetro y el giroscopio pueden detectar los cambios en la posición del dispositivo en 6 ejes.

  Su aplicación a los equipos también incrementa el costo, y como no es muy necesario, no todos los equipos cuentan con el giroscopio y es más común encontrarlo en equipos de gama alta. Uno de los primeros en tenerlo fue el iPhone 4, aunque ahora ya abunda entre otros modelos.

2.2. Tecnologías

2.2.1 Android

Android [Android] es un Sistema Operativo además de una plataforma de Software basada en el núcleo de Linux. Diseñado en un principio para dispositivos móviles, Android permite controlar dispositivos por medio de bibliotecas desarrolladas o adaptados por Google mediante el lenguaje de programación Java.

Android, al contrario que otros sistemas operativos para dispositivos móviles como iOS o Windows Phone, se desarrolla de forma abierta y se puede acceder tanto al código fuente como a la lista de incidencias donde se pueden ver problemas aún no resueltos y reportar problemas nuevos.

Inicialmente, Android fue desarrollada por Google Inc. aunque poco después se unió Open Handset Alliance, un consorcio de 48 compañías de Hardware, Software y telecomunicaciones, las cuales llegaron a un acuerdo para promocionar los estándares de códigos abiertos para dispositivos móviles.

Google sin embargo, ha sido quien ha publicado la mayoría del código fuente de Android bajo la licencia de Software Apache, una licencia de software libre y de código abierto a cualquier desarrollador.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (surface manager), un framework OpenCore, una base de datos relacional SQLite, una Interfaz de programación de API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, SSL y una biblioteca estándar de C Bionic. El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2,8 millones de líneas de lenguaje C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++ [Android].

2.2.2 Web Services - Hessian

Los Web Services [Alonso et al., 2004] son una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las

organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Mediante los web services, las aplicaciones Android pueden enviar información hacia el servidor web, pero necesitan “acordar” un determinado protocolo para comunicarse. Este protocolo define cómo será el formato de la comunicación. Una posibilidad es utilizar la librería Flamingo [Flamingo], de código fuente abierto y provista por la marca Exadel. Esta librería permite comunicar una aplicación Android mediante Hessian con un servidor web.

Hessian [Hessian] es un protocolo de web services binario y que los web services sean implementados sin la necesidad de utilizar un framework de gran tamaño o complejidad y sin la necesidad de tener que aprender un nuevo conjunto de protocolos. Además, por ser un protocolo binario, está adaptado a enviar información binaria sin la necesidad de tener que extender el protocolo con información adjunta.

Hessian optimiza el tiempo de respuesta comparado con otros protocolos de web services. La Figura 2 muestra una comparativa teniendo en cuenta el tamaño de elementos que se envían por protocolo y los tiempos de respuesta para cada uno.

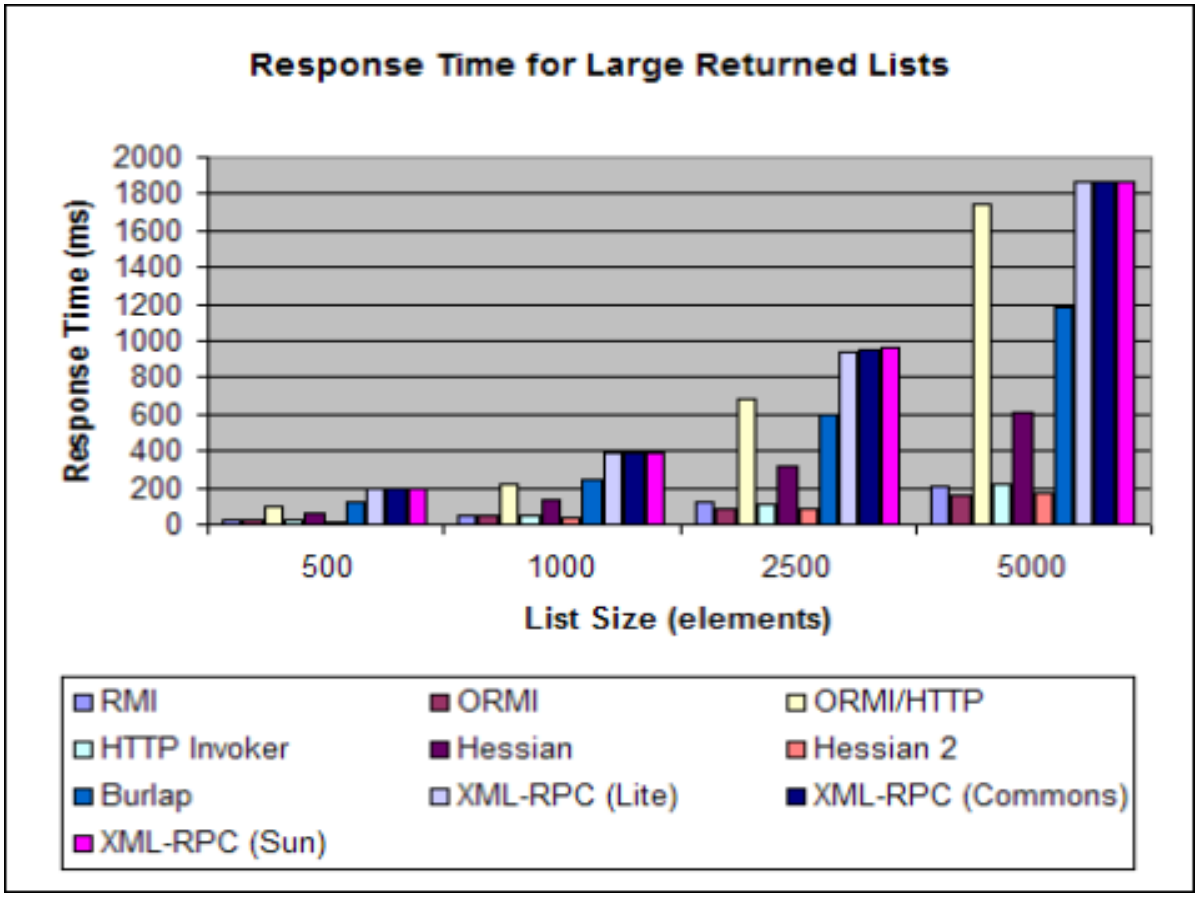


Figura 2: Tiempo de respuesta en retornar listas con N elementos [Hessian-Comparación]

2.2.3 Servidor de Aplicaciones

El servidor de aplicaciones [Servidor] es el lugar donde residen las aplicaciones web. Usualmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

2.2.3.1 JEE

JEE [JEE] es una plataforma de programación —parte de la Plataforma Java— para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una especificación. Similar a otras especificaciones del Java Community Process, Java EE es también considerada informalmente como un estándar debido a que los proveedores deben

cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes a Java EE; estandarizado por The Java Community Process / JCP.

2.2.3.2 Spring Framework

Spring [Spring] es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java.
Spring comprende diversos módulos que proveen un rango de servicios. Entre los más importantes podemos mencionar:

- **Contenedor de inversión de control:** Permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias, que es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto.

- **MVC mediante Spring Web Flow:** Este módulo permite la implementación del patrón MVC mediante XMLs que se comportan como controladores administrando el flujo de navegación de la aplicación.

2.2.3.3 Java Server Faces (JSF)

Java Server Faces [JSF] es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario).
JSF se compone de:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.

- Un conjunto por defecto de componentes para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.
- Administración de estados.

- Beans administrados.

2.2.3.4 Primefaces - Push

Primefaces [Primefaces] es una librería de componentes JSF que facilitan la creación de las aplicaciones web. Primefaces está bajo la licencia de Apache License V2. Primefaces es una biblioteca liviana, todas las decisiones tomadas se basan en mantener una velocidad de respuesta óptima.

PrimeFaces también incluye PrimePush, un servlet push Ajax basado en el framework Atmosphere. El método *push* [Push] es útil para la construcción de estilo multicast, aplicaciones en las que un evento de servidor puede ser transmitido a varios clientes, como de escritorio y los usuarios móviles a la vez.

2.2.3.5 Maven

Maven [Maven] es una herramienta de software para la gestión y construcción de proyectos Java. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero actualmente es un proyecto de nivel superior de la Apache Software Foundation. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Maven provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios.

Capítulo 3: Solución Propuesta para crear Aplicaciones Móviles Sensible al Contexto

3.1. Descripción del problema a solucionar

Existen determinados tipos de aplicaciones que requieren de la información de su entorno para brindar soluciones personalizadas [Schilit, 2004]. Esto mejora el poder de adaptación de la aplicación y aporta información específica de acuerdo a las necesidades del usuario en un tiempo y espacio determinado.

Este tipo de problemas no posee una única solución, sino que depende de su entorno para poder brindar la solución adecuada en el momento adecuado, teniendo en cuenta toda la información disponible del entorno al momento de tomar una decisión. Por ejemplo, supongamos que un turista visita una ciudad que desconoce y quisiera visitar los lugares más importantes pertenecientes a ésta. Lo ideal, sería que exista una aplicación que ofrezca al turista información personalizada de acuerdo a su ubicación, hora del día, día de la semana, etcétera. Esta información es de vital importancia para la aplicación y es obtenida, por ejemplo, a través de los sensores del dispositivo. La sumatoria de todos estos datos o variables con los que trabaja la aplicación, es denominada *contexto* [Dey, 2000] como se mencionó en el Capítulo 2.

Si se desea que una aplicación turística como la antes descripta esté disponible para la mayoría de los dispositivos actuales, sin importar su sistema operativo, hardware y demás componentes particulares de cada tecnología, se está pensando en una solución multiplataforma.

En [Espada et al., 2012b] se propone una solución para la problemática planteada pero no cumple la condición de ser multiplataforma debido a que se compone de una aplicación nativa con un browser embebido, lo cual hace imposible correr esta aplicación en distintos sistemas operativos. Al ser una aplicación nativa puede acceder a los datos de los sensores de manera directa sin problema para podérselo pasar al browser. La aplicación móvil nativa tiene embebido un navegador web, y ese acoplamiento hace que el navegador web necesite de capacidades *no-standards* para funcionar. Si se quisieran utilizar los datos del dispositivo móvil para dos propósitos distintos, entonces se tendría que crear una nueva aplicación móvil.

Para lograr una aplicación móvil sensible al contexto independiente de la plataforma sería ideal poderla plantear como un aplicación web. Es decir, que desde el browser se envíen los datos de contexto del usuario y también se reciban la actualización de otros datos de contexto (por ejemplo, datos del tránsito).

Gracias al avances de HTML5 [HTML5] una solución multiplataforma esta cada vez más cercana. La definición de HTML5, ofrece la posibilidad de poder acceder a la información de los sensores del dispositivo que realiza el request. Sin embargo, esta tecnología es muy reciente y aún no es soportada por todos los browsers, y depende de cada implementación ya que no es standard. Adicionalmente, HTML5 tiene acceso a datos de ciertos sensores, pero no de todos, haciendo de esta opción una alternativa prometedora pero aún inviable [HTML5Spec].

Acorde a lo antes descripto, se propone una solución multiplataforma pensada para que cuando HTML5 brinde total soporte al acceso de los datos de todos los sensores del dispositivo. Es decir, se va a usar un browser tradicional y adicionalmente una aplicación móvil nativa para poder simular el acceso a los sensores que todavía no brinda HTML5. Esta aplicación móvil permitirá tomar los datos de los sensores (información del contexto) y la enviará al servidor de manera independiente al browser mencionado.

En un futuro cuando HTML5 brinde total soporte al acceso de los sensores del dispositivo la solución planteada quedaría funcionando solo con un browser tradicional. Es decir, la solución planteada está pensada para que cuando la tecnología (HTML5) evolucione se pueda refactorizar con mínimos cambios, y así seguir funcionando.

3.2. Descripción general de la solución propuesta

Acorde a las limitantes que tiene HTML5 como se describió en la sección anterior, se propuso una Arquitectura que consiste en una variante del modelo Cliente-Servidor.
La solución propuesta es independiente del dominio que se quiera representar. A continuación se hará hincapié en la comunicación necesaria entre cliente y servidor para poder intercambiar los datos de los sensores y así reaccionar (mostrar) al cambio de contexto.

Por un lado, está la parte Cliente, donde se deberá tener:

- Un navegador web tradicional (que muestra los datos acorde a la naturaleza de la aplicación sensible al contexto que se quiere representar).
- Una aplicación móvil (nativa) encargada de tomar datos de los sensores y enviarlos al servidor.

Y por otro lado, la parte Servidor que consiste en una aplicación web que realiza lo siguiente:

- La recepción de los datos de los sensores de cada dispositivo móvil (planteado como Web Services)

- La actualización del navegador web, o browser, en tiempo real con los datos sensados.

Cómo se mencionó anteriormente, en el dispositivo móvil pueden convivir dos aplicaciones que no se comunican entre sí, sino que lo hacen a través del servidor. La aplicación móvil cliente, encargada de tomar los datos de los sensores (sensado), envía al servidor estos datos cada cierto intervalo de segundos. Luego, como se explicará más adelante en la parte del servidor, el navegador web se actualiza con los datos recibidos desde el servidor, en forma asincrónica.

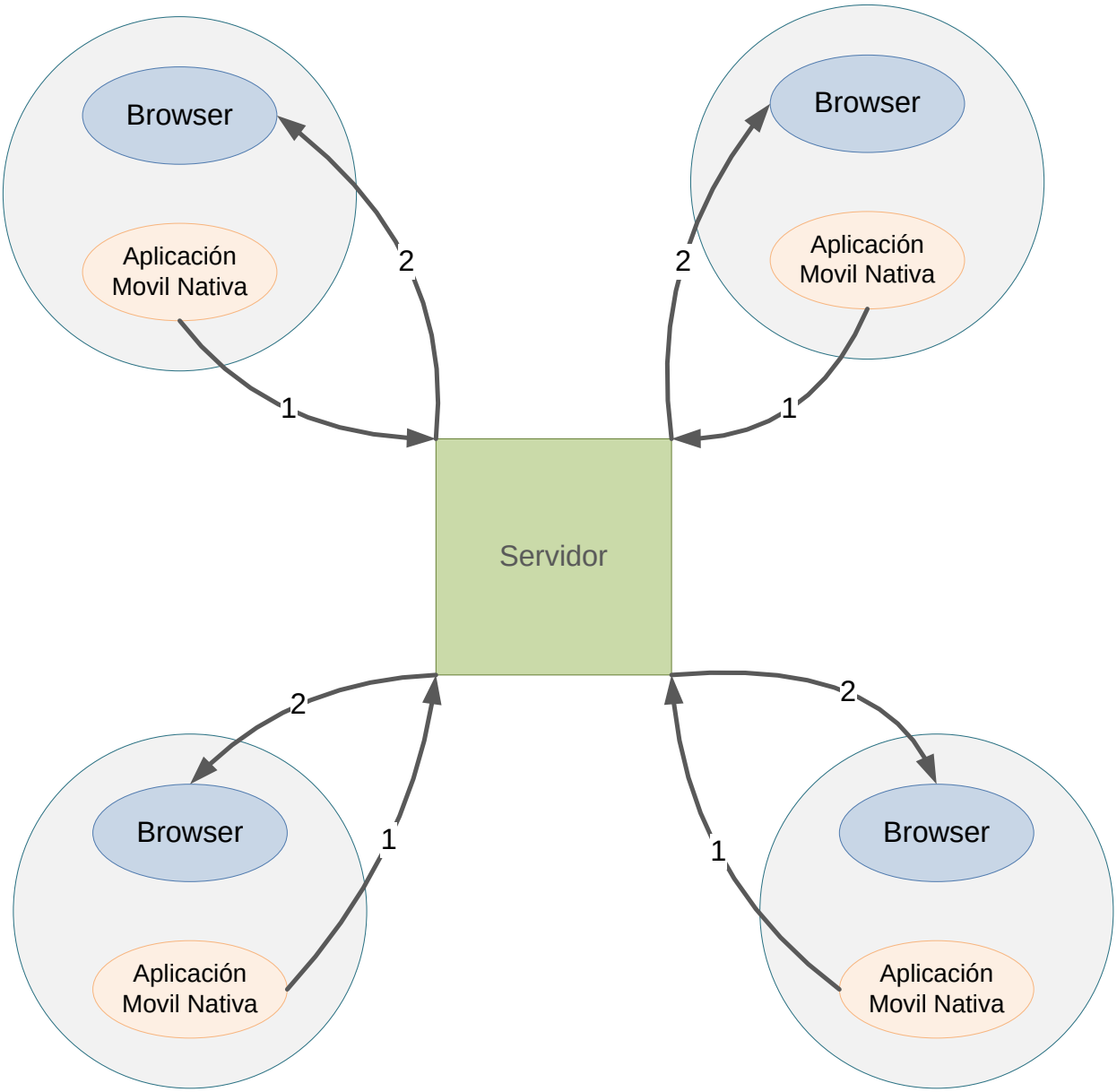
Veamos más detalles de la funcionalidad del servidor, como se mencionó anteriormente el servidor se encarga de:

- Recibir y almacenar los datos de todos los sensores para cada dispositivo. Es decir, para cada dispositivo que está usando la aplicación móvil se registran los datos de contexto, y estos se van actualizando a medida que estos cambian (avisando de estos cambios al servidor).
- Enviar la actualización de los datos recibidos al navegador web (o los navegadores) que se necesita refrescar. Un dato de un dispositivo, por ejemplo, la posición puede ser de interés para varios browser (mostrar donde están posicionados determinados usuarios).

Las dos funciones principales del servidor se encuentran totalmente desacopladas. Esta independencia hace que sea irrelevante el mecanismo por el que se obtienen los datos de los sensores, permitiendo que en el futuro los datos puedan provenir de otra aplicación, de HTML5 (enviados por el mismo browser), u otro mecanismo que sirva para ese propósito.

El intercambio de los datos entre el cliente y el servidor se realiza en forma totalmente asíncrona, esto es, el cliente envía (mediante la aplicación móvil) datos cada N segundos mientras esté ejecutándose, pero antes verificando previamente que los valores para el sensor en cuestión hayan cambiado respecto al valor que ya tenía. Cada cliente o dispositivo móvil (ya que pueden llegar a ser muchos dispositivos intercambiando datos con el servidor), envía estos datos al servidor. Por su parte, el servidor recibe estos datos y también verifica que esa información recién obtenida (del dispositivo) difiera de la información que se está mostrando en el navegador del cliente. En el caso que estos datos difieran, el servidor debe enviar los nuevos valores al navegador (browser) correspondiente, así este se puede actualizar acorde a la naturaleza de la aplicación.

Un esquema de la solución propuesta se puede apreciar en la Figura 3.



1: Cada dispositivo móvil envía periódicamente al servidor información de sus sensores.
2: El servidor envía los nuevos datos al navegador correspondiente, éste realiza la lógica y refresca la pantalla con la nueva información. En este caso, todos los browser están interesados en todos los datos de contexto.

Figura 3: Esquema de la solución propuesta

El desacoplamiento de la aplicación web respecto de la aplicación móvil encargada de obtener la información de los sensores, hace que se pueda trabajar en diferentes capas. Se podría enunciar esa división de la siguiente manera:

- La aplicación móvil nativa encargada de obtener los datos sensados y de enviarlos al servidor.
- El servidor, encargado de obtener los datos enviados por los clientes y enviar las actualizaciones a la aplicación web.

- La aplicación web, que se ejecuta sobre cualquier navegador que usa los datos de uno o más dispositivos para realizar su lógica necesaria.

3.3. Interacción entre el dispositivo móvil, el servidor y el navegador web

Para poder ejemplificar mejor la interacción tomemos un caso simplificado de la solución propuesta como se muestra en la Figura 4. Donde se tienen distintas aplicaciones móviles nativas enviando datos de sus sensores (contexto) y un solo browser que refleja esos datos de contexto.

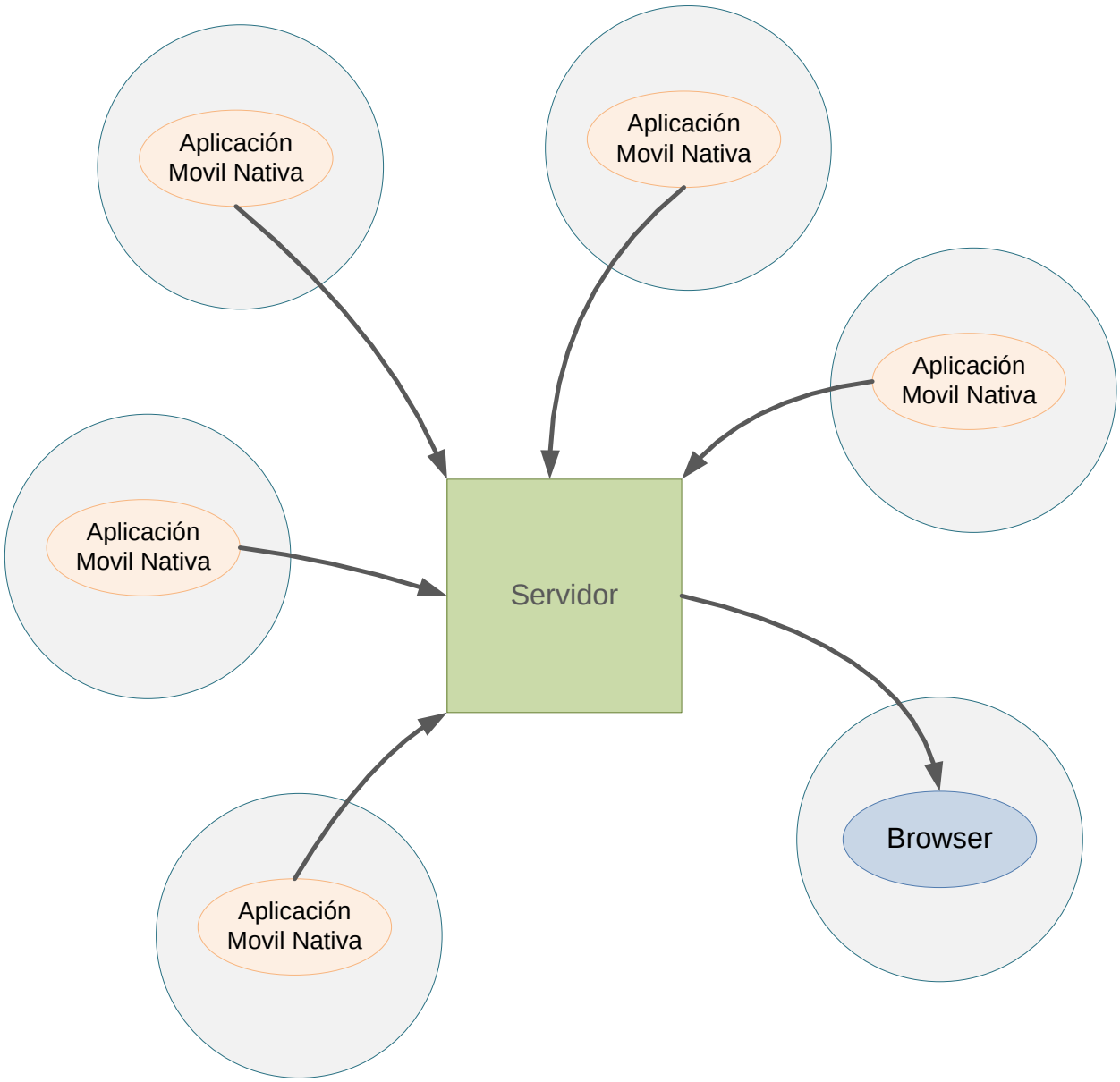


Figura 4: Simplificación de la solución propuesta.

A continuación se mostrará la interacción entre una aplicación móvil nativa de un dispositivo móvil, el servidor y un navegador web. En particular, nos vamos a focalizar en dos situaciones específicas:

- Enviar datos desde una aplicación móvil nativa al servidor
- El browser solicita explícitamente datos al servidor

Veamos a continuación el flujo que se establece para las dos situaciones antes descriptas mediante diagramas de secuencias.

a) Enviar datos desde una aplicación móvil nativa al servidor

Como se puede apreciar en la Figura 5, el inicio de comunicación se establece desde la aplicación móvil nativa del dispositivo móvil. La aplicación móvil nativa toma los datos de los sensores y los envía al servidor. El servidor será el encargado de mantener actualizados los datos de cada dispositivo. Cada vez que los datos recibidos por el servidor difieren de los que tiene almacenados, debe avisarle al navegador web (o los navegadores web) de estos cambios. Por último, el navegador web refresca el contenido HTML con los datos recibidos.

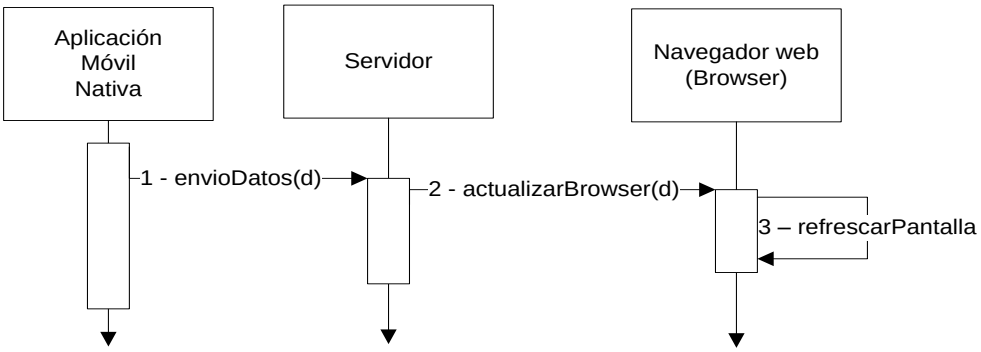


Figura 5: Enviar datos desde una aplicación móvil nativa al servidor

La secuencia de pasos de la Figura 5 son los siguientes:

1- envioDatos(d): El dispositivo envía los datos de alguno de los sensores (d = datos del sensor), al servidor web.
2- actualizarBrowser(d): El servidor web recibe los datos por parámetro y envía la nueva información (d) al browser (cliente) para que se actualice.
3- refrescarPantalla: El navegador web, con los nuevos datos, realiza su lógica correspondiente y refresca los datos en pantalla.

b) El browser solicita explícitamente datos al servidor

La primera vez que el navegador web realiza un request al servidor (ver Figura 6), éste último recorre los datos almacenados y envía una respuesta con la información disponible en ese momento. Dicha respuesta dependerá de que la información, del dispositivo o los dispositivos requeridos, exista. De lo contrario, el servidor enviará una respuesta indicando que los datos aún no han sido inicializados. Para lo cual, podría definirse un formato específico para contemplar estos casos. Por ejemplo, una página de error o un mensaje indicando la situación.

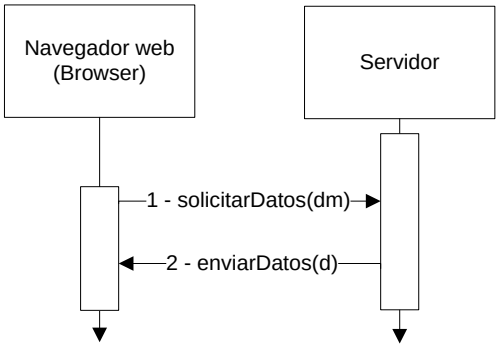


Figura 6: El browser solicita explícitamente datos al servidor

La secuencia de pasos de la Figura 6 son los siguientes:

1- solicitarDatos(dm): El navegador web se inicia y le pide los datos del o de los contextos de los dispositivos móviles al servidor (dm = datos del dispositivo móvil).
2- enviarDatos(d): El servidor, ante la petición del navegador web, envía al navegador web los datos correspondientes. Luego, el browser se actualiza con los datos recibidos desde el servidor.

En la Figura 7 se puede apreciar el caso de un intento fallido de comunicación iniciando desde el navegador web con dispositivo no listo/disponible.

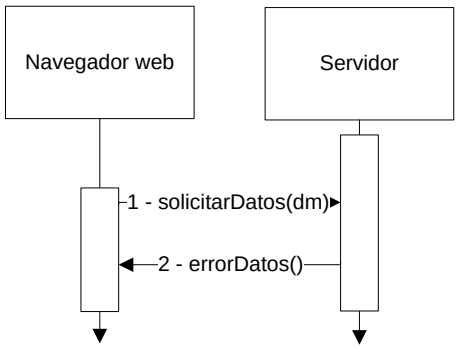


Figura 7: Intento fallido de comunicación iniciando desde el navegador web

La secuencia de pasos de la Figura 7 son los siguientes:

1- solicitarDatos(dm): El navegador web se inicia y le pide los datos del o de los contextos de los dispositivos móviles al servidor (dm = datos del dispositivo móvil).
2- enviarDatos(d): El servidor no dispone de los datos requeridos por el navegador web, esto puede deberse a que el dispositivo móvil no está disponible, no responde o dejó de enviar datos. El servidor entonces informa al navegador web de esta situación. Luego, el navegador web muestra el error en pantalla.

Los tipos de interacción descriptos pueden ampliarse tanto si se tienen más dispositivos móviles enviando datos de los sensores, o más browser consumiendo estos datos. Además, los datos de los sensores podrían ser consumidos por diferentes tipos de aplicaciones o cada browser personalizar la información que muestran.

Capítulo 4: Implementación del prototipo

4.1. Especificación general del prototipo

En el Capítulo 3 se presentó un esquema general para brindar una solución a la actualización de Aplicaciones Móviles Sensibles al Contexto, y además una simplificación de este esquema general, el cual se mostró en la Figura 4. Para el prototipo se decidió implementar tomando de base la solución simplificada pero acotando el prototipo a un solo dispositivo. Es decir, se contará con un dispositivo que cuenta con una aplicación móvil nativa que envía los datos al servidor acorde a los cambios de los mismos, por otro lado se contará con un servidor que recibe estos datos y le avisa al browser del dispositivo el cambio ocurrido (acorde a la naturaleza de la aplicación). Lo antes descripto se refleja en la Figura 8. Cabe destacar que hasta este momento el esquema de solución propuesta es general, más adelante se detallarán las características de la aplicación contemplada por el prototipo.

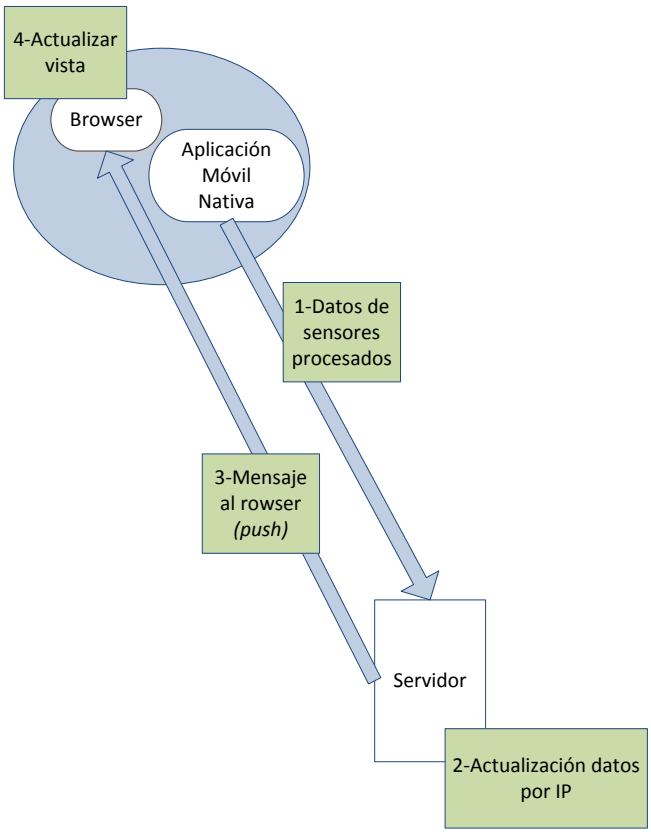


Figura 8: Diagrama que muestra el funcionamiento principal del prototipo cuando algún dato de sensor cambia.

Básicamente, la Figura 8 describe la Aplicación Móvil Nativa, encargada de obtener los datos de los sensores, procesarlos y enviarlos al servidor mediante web services

(1-Datos de los sensores procesados). El servidor recibe los datos procesados, guarda esa información identificando al dispositivo por su dirección IP *(2-Actualización datos por IP)*, y envía un mensaje al browser del dispositivo mediante el mecanismo *push (3-Mensaje al browser)*. Finalmente, el browser recibe la información del servidor y actualiza la vista de acuerdo a la orientación del dispositivo (vertical u horizontal) *(4-Actualizar vista)*.

Se puede apreciar de la Figura 8 que el prototipo consta de tres partes:

- La Aplicación Móvil Nativa
- El servidor
- Aplicación Web que se ejecuta en el browser

Estas partes serán presentadas con más detalle en las Secciones 4.2, 4.3 y 4.4 respectivamente.

A continuación se describirán las características de la Aplicación Móvil Sensible al Contexto implementada con el prototipo. Es decir, el domino a representar.

La funcionalidad principal del dominio de este prototipo es informar los lugares cercanos al dispositivo (puntos de interés), de acuerdo a dos "modos". Si el dispositivo está en forma vertical, se desplegará una lista de los puntos de interés (POI) que se encuentran en la dirección hacia donde apunta el dispositivo (ángulo de visión), además de mostrar información de luz y temperatura. En cambio, si el dispositivo se encuentra en forma horizontal, el browser debe mostrar un mapa con la posición del dispositivo mediante un marcador azul, y los puntos de interés cercanos alrededor de la posición del dispositivo con marcadores rojos.

El mapa que se muestra cuando el dispositivo se encuentra orientado horizontalmente, forma parte de la API provista por Google Maps [Google Maps] y es manipulada mediante código javascript.

Los puntos de interés (POI) se almacenan en el servidor y poseen la información posicional (latitud y longitud), y una descripción. Dichos puntos de interés fueron elegidos arbitrariamente referenciando distintos lugares destacados de la ciudad de La Plata.

Por ejemplo, los siguientes POI con su ubicación (latitud y longitud):

Descripción del POI	Latitud	Longitud
Hipódromo Ciudad de La Plata	-34.89896	-57.94636
Observatorio de la UNLP	-34.90682	-57.93248
Facultad de Ciencias Naturales y Museo UNLP	-34.90828	-57.92641

Facultad de Ciencias Medicas UNLP	-34.90946	-57.92774

De acuerdo a la posición de cada dispositivo, el servidor realizará el cálculo correspondiente para determinar los lugares cercanos a esa posición. Se toma como lugar cercano aquellos POI que se encuentren a una distancia menor a 1000 metros del dispositivo. En las Figuras 9 y 10 se muestra un diagrama con el ángulo y distancia de visión según la orientación del dispositivo (horizontal o vertical).

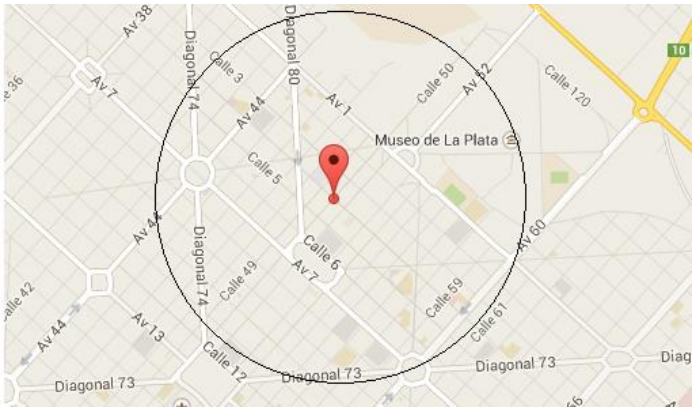


Figura 9: Radio de búsqueda de POI para cuando el dispositivo se encuentra en forma horizontal

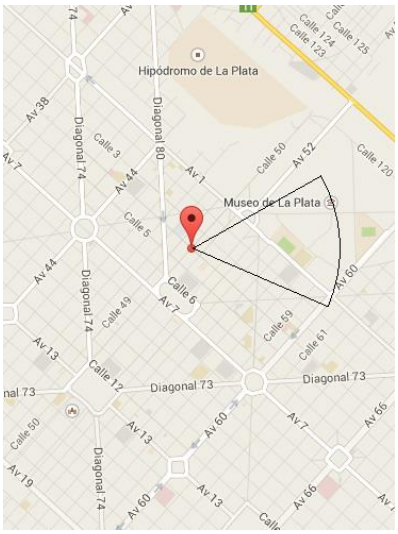


Figura 10: Ángulo y distancia de visión para cuando el dispositivo se encuentre orientado en forma vertical

La página principal de la aplicación web consta de dos componentes básicos que mostrarán al usuario la información actualizada en tiempo real. Por un lado se tiene una lista con los lugares cercanos si el dispositivo está en modo vertical. En la Figura 11 se puede ver un bosquejo de cómo se puede apreciar esta lista. Esta lista de lugares se irá actualizando a medida que el usuario se mueva de lugar o si cambia el ángulo de visión.

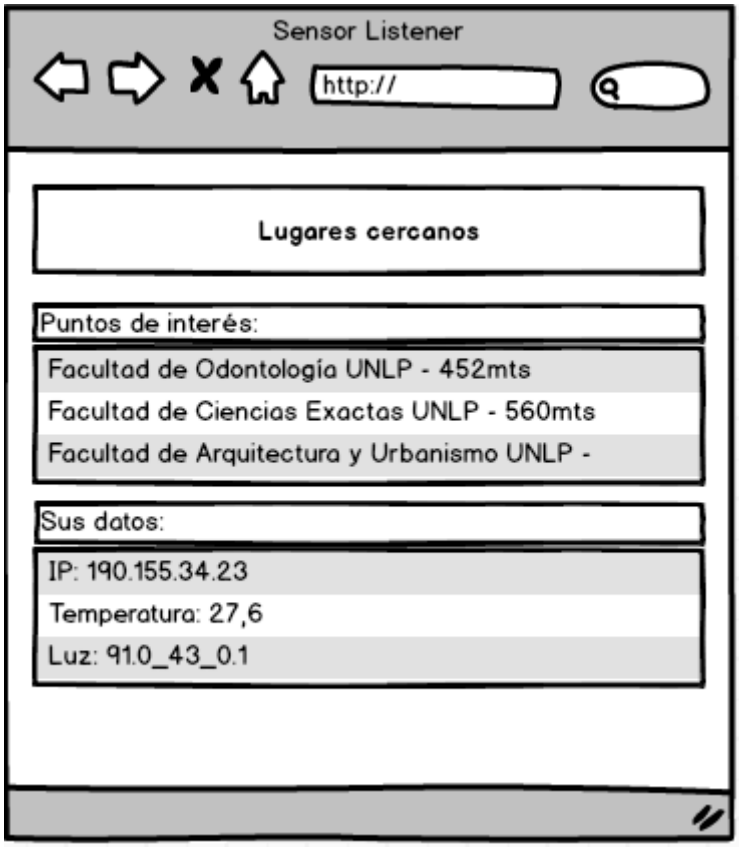


Figura 11: Esquema de ejemplo de la visualización en el browser cuando el dispositivo se encuentre orientado verticalmente.

En cambio si el dispositivo esta horizontal, muestra un mapa con los lugares cercanos, un bosquejo de esto se puede apreciar en la Figura 12. Se puede apreciar que en este modo se muestran todos los puntos de interés sin considerar el ángulo de visión, solo se considera el radio como se mostro en la Figura 9.

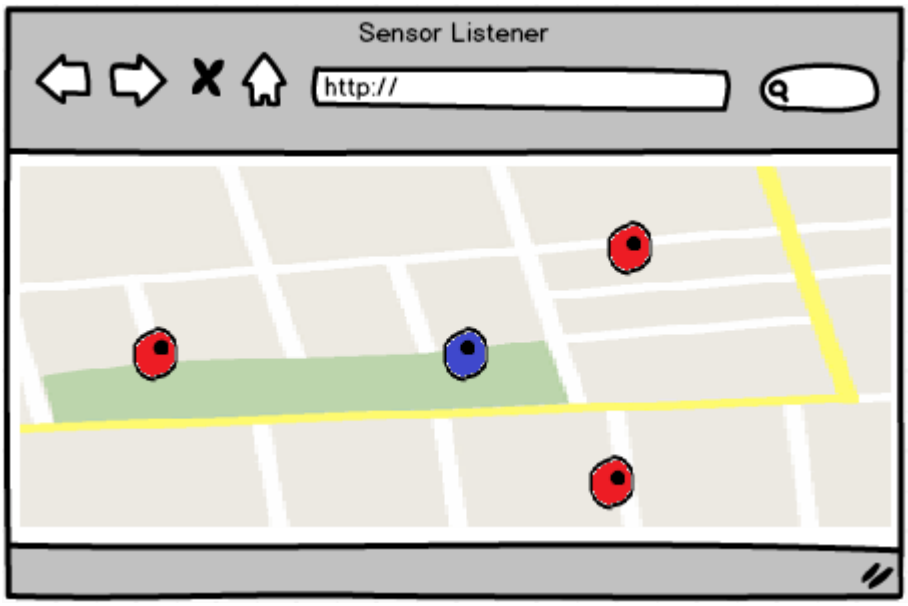


Figura 12: Esquema de ejemplo de la visualización en el browser cuando el dispositivo se encuentre orientado horizontalmente (mapa con marcador azul indicando la posición del dispositivo y en rojo los puntos de interés cercanos)

Cuando se describan cada una de las partes involucradas en el prototipo, se indicará como cada concepto mencionado anteriormente fue implementado.

4.2. Aplicación Móvil Nativa

El prototipo implementa una Aplicación Móvil Nativa que toma datos de algunos sensores del dispositivo, se decidió implementar esta aplicación usando Android (versión 4 o superior) para tomar los datos de los sensores. Esta decisión fue basada en el vasto mercado que utiliza dicha plataforma y aprovechando las ventajas que ya posee Android para el acceso a los datos de los sensores.

Android provee una API (*Application Program Interface*) [Android Sensor API] que permite utilizar los sensores disponibles en el dispositivo en el cual se ejecuta la aplicación. Básicamente, se tiene que indicar que la aplicación en cuestión debe implementar la interfaz Java *SensorEventListener*, y debe implementar del método `onSensorChanged`. En este método se debe programar la lógica necesaria que se ejecutará cada vez que cambia un dato de un sensor. El sensor que disparó este evento se recibe como parámetro del método, para así poder distinguirlo. Es decir, este método se comparte por todos los sensores, y en la lógica del método se debe desambiguar que acción tomar acorde a cuál fue el sensor que vario su valor.

El mecanismo para “escuchar” los eventos del GPS es diferente ya que la interfaz a implementar es otra llamada *LocationListener*, y la lógica relacionada a este evento debe indicarse en el método `onLocationChanged`. Cada vez que cambia el valor del GPS este método es invocado.

Si bien Android provee soporte para muchos tipos de sensores, en la implementación del prototipo se utilizarán solo los sensores que se listan a continuación:

- Acelerómetro
- Luz
- Giroscopio
- Magnetómetro
- Temperatura
- GPS

Para poder estar escuchando por los datos de los sensores, se decidió que la Aplicación Móvil Nativa este implementada como una aplicación Android, para esto una de las clases a implementar debe de extender de la clase *Activity* e implementar el método `onCreate`. Este método se invocará cuando se ejecute la aplicación. Para esto, se creó la clase ***SensorListener*** que extiende de *Activity* y se decidió implementar en el método `onCreate` la lógica necesaria para la configuración de los sensores. Esto implica configurar e inicializar el *SensorManager* y *LocationManager*

La aplicación Android estará escuchando los cambios que se producen en los sensores y, dependiendo del sensor que haya disparado el evento, crea una tarea (task) que se va a encargar de transmitir los datos al servidor. Para lograr escuchar los cambios de los sensores la clase *SensorListener* implementa las interfaces *SensorEventListener* y *LocationListener*, y los métodos correspondientes a las mismas para poder reaccionar a los cambios de los sensores (métodos `onSensorChanged y onLocationChanged`).

A continuación se especifican con más detalle lo descripto anteriormente.

a. *Configuración e inicialización del SensorManager*

En el método `onCreate` de la clase *SensorListener* se definió la configuración necesaria para el *SensorManager* como se muestra a continuación:

```
sensorManager =
            (SensorManager)
            this.getSystemService(SENSOR_SERVICE);
```

Con el código definido en la línea anterior queda configurado el *SensorManager*, luego hay que indicar que la clase *SensorListener* quiere “escuchar” los cambios de los datos del sensor. A continuación se muestra el código relacionado, por ejemplo, para escuchar por el acelerómetro. Se puede apreciar que se obtiene el sensor que se corresponde a `Sensor.TYPE_ACCELEROMETER`, hay que

corroborar que haya un sensor de este tipo, podría ser que el dispositivo no contará con dicho sensor, en cuyo caso no se puede “escuchar” por un sensor inexistente. En el caso de que haya un sensor de acelerómetro, se lo agrega a la lista de sensores que se quieren “escuchar” Esto se puede apreciar en la última línea del siguiente código (el `this` hace referencia a la clase *SensorListener*).

```
List<Sensor> accelerometerSensorList =
sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
if (!accelerometerSensorList.isEmpty()) {
    Sensor accelerometerSensor = accelerometerSensorList.get(0);
    sensorManager.registerListener(this, accelerometerSensor, 3);
}
```

Lo ejemplificado para el acelerómetro se realizó además para los sensores de *Luz, Giroscopio, Magnetómetro* y *Temperatura.*

b. *Configuración e inicialización del LocationManager*

El GPS es el único sensor que necesita de una inicialización diferente. Se agregaron las siguientes líneas de código en el método `onCreate` de la clase *SensorListener*. Primero se obtiene el *LocationManager,* y luego se indica que se quiere estar “escuchando” el GPS y la red, en este caso va a estar “escuchar” la clase *SensorListener* (esto está especificado con el `this` en el código). El aviso de cambio en este caso particular se realiza cada diez segundos.

```
LocationManager locManager =
(LocationManager) this.getSystemService(LOCATION_SERVICE);
//Envia cada 10 segundos el valor del GPS
locManager.requestLocationUpdates(
      LocationManager.GPS_PROVIDER, 10 * 1000, 0, this);
locManager.requestLocationUpdates(
      LocationManager.NETWORK_PROVIDER, 10 * 1000, 0, this);
```

Al indicar que se quiere “escuchar” por el GPS_PROVIDER y NETWORK_PROVIDER, se está expresando dos formas de obtener la posición del dispositivo, donde:

- GPS_PROVIDER: Utiliza el hardware especifico que posee el dispositivo y es el mecanismo que resuelve con mayor precisión.
- NETWORK_PROVIDER: Utiliza la red GSM a la que pertenece el dispositivo. Aunque si el dispositivo no está conectado a ningún proveedor, esta información no estará disponible.

c. *Definición del método onSensorChanged*

Para atender los cambios que se producen en los sensores, se debe implementar el método `onSensorChanged` que tiene la siguiente definición:

```
void onSensorChanged(SensorEvent pEvent) {…}
```

Donde en el parámetro `pEvent` se recibirá el tipo del sensor que disparo el evento para poder identificarlo y reaccionar de acuerdo a su valor. A continuación se especifica como determinar que el sensor que mando el cambio fue el `Sensor.TYPE_ACCELEROMETER`. Cuando este sensor envía un cambio, se llama al método `calculateOrientation` (el cual termina invocando a la tarea que manda los datos al servidor). Esto mismo se realizó con los otros sensores (de *Luz, Giroscopio, Magnetómetro* y *Temperatura*).

```
switch (pEvent.sensor.getType())
{
    ...

    case Sensor.TYPE_ACCELEROMETER:
        accelerometerValues = pEvent.values.clone();
        this.calculateOrientation();
        break;
}
```

d. Definición del método onLocationChanged

Para manejar los eventos de cambio de posición del GPS se debe implementar el método `onLocationChanged` como se muestra a continuación. Se puede apreciar que se crea una tarea (`ClientGPSTask`) se le setean los datos de la posición y luego se ejecuta dicha tarea, la cual enviará los datos al servidor.

```
public void onLocationChanged(Location location) {
   gpsTask = new ClientGPSTask();
   gpsTask.setLatitude(location.getLatitude());
   gpsTask.setLongitude(location.getLongitude());
   gpsTask.execute();
}
```

De este modo, cuando se detecten cambios en la ubicación o pasen 10 segundos desde la última vez que se accedió al posicionamiento del dispositivo, se ejecutará el método `onLocationChanged`.

e. Definición de Tareas (threads)

El paquete de servicios con operaciones básicas de Android (*android.os*), provee el tipo de tarea *AsyncTask*, mientras que el paquete util (*java.util*) provee el tipo *TimerTask*. El primero de ellos se utiliza para realizar tareas en segundo plano,

para luego esperar el resultado en forma asincrónica, si se tiene interés en su respuesta. El segundo tipo, representa una tarea que se ejecuta en un tiempo especificado, una vez o muchas veces repetidas.

En nuestra aplicación móvil nativa se definieron, para la transmisión de datos hacia el servidor, dos tipos de tareas:

- **ServiceAsyncTask**: Se implementó como una subclase de *AsyncTask* (Tarea asincrónica). Este tipo de tarea es utilizado para transmitir la orientación (Vertical u Horizontal), la dirección (Norte, Sur, Este u Oeste) y los datos del GPS al servidor.
- **ServiceTimerTask**: Se implementó como una subclase de *TimerTask* (Tarea con temporizador) y consiste en un hilo (o thread) que ejecuta su método principal llamado *run* cada cierto intervalo de tiempo, que puede configurarse. Este tipo de tareas es utilizado para transmitir los datos de los sensores de luz y temperatura.

Esta decisión de dividir la invocación a los servicios en dos tipos de tareas, surgió a raíz de la forma en que necesitan ser actualizados los datos de los sensores en el servidor. Por ejemplo el dato de la orientación o dirección, es necesario actualizarlos inmediatamente, a demanda, ya que de este valor depende la distribución de los componentes visuales en la aplicación web. En cambio los datos del sensor de luz y temperatura, se envían cada cierto intervalo de tiempo, para no sobrecargar al servidor con peticiones que poseen datos que varían constantemente en forma mínima, y no son parte fundamental de la vista de usuario de la aplicación web.

En la Figura 13 se pueden apreciar las sublases de *ServiceAsyncTask* definidas.

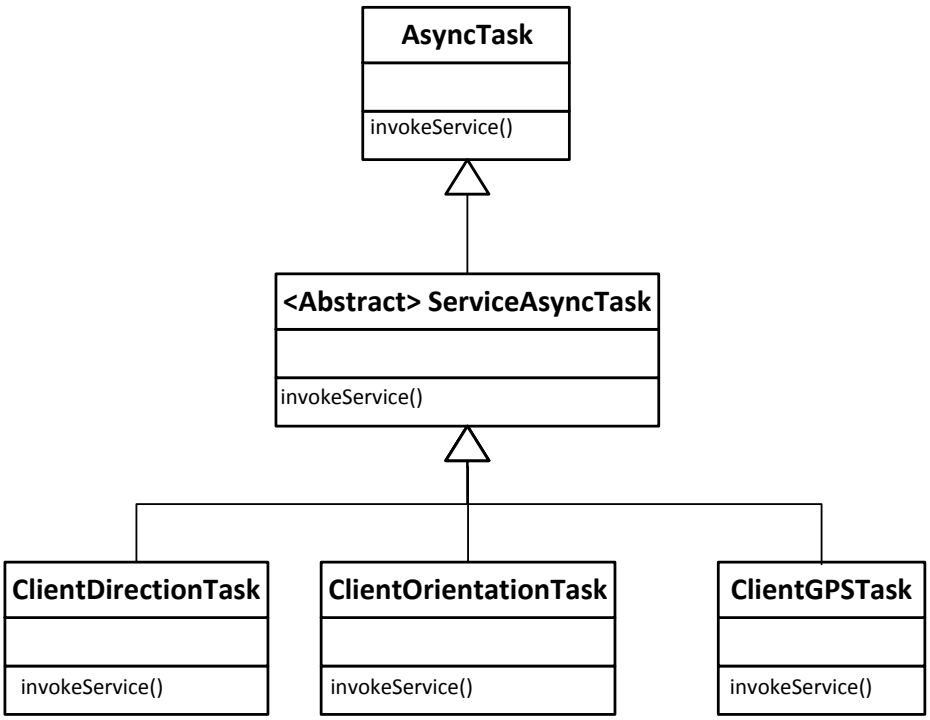


Figura 13: Jerarquía de clases de tareas asincrónicas

A continuación se define el propósito de cada clase concreta:

- **ClientDirectionTask**: Esta tarea es la encargada de transmitir al servidor los datos de la dirección (Norte, Sur, Este, Oeste, Noroeste, Noreste, Sudeste o Sudoeste).
- **ClientOrientationTask**: Esta tarea es la encargada de transmitir al servidor los datos de la orientación (Vertical u horizontal).
- **ClientGPSTask**: Esta tarea es la encargada de transmitir al servidor los datos del GPS (Latitud + Longitud).

En la Figura 14 se puede apreciar la clase *ClientLightTempetureTask* que es subclase de *ServiceTimerTask*. Esta clase es la encargada de transmitir al servidor la información de la luz y la temperatura obtenida por los sensores del dispositivo.

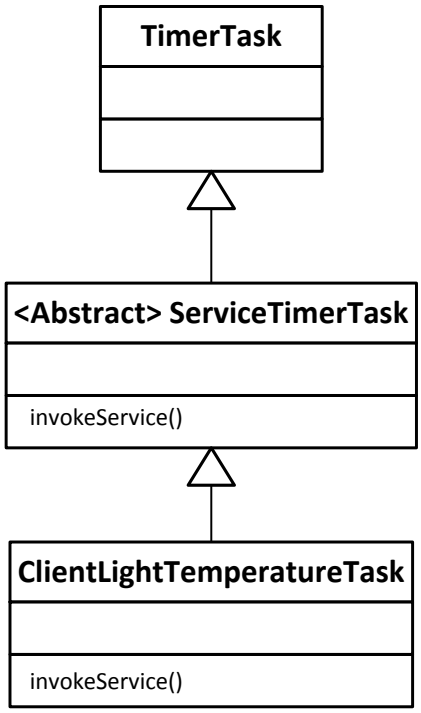


Figura 14: Jerarquía de clases de tareas temporizadas

La clase principal de nuestra aplicación, llamada *SensorListener*, es la encargada de crear e invocar las diferentes tareas de acuerdo al evento que se ha disparado. Por ejemplo, cuando la clase *SensorListener* detecta un evento del sensor de dirección, se invoca a la tarea *ClienteDirectionTask* que hereda de *ServiceAyncTask* y es la encargada de transmitir los nuevos datos.

A continuación se muestra la manipulación de una tarea de clase *ClientDirectionTask*. Donde para este caso particular, el método `execute` consiste en invocar al servicio web enviando los datos de la dirección.

```
directionTask = new ClientDirectionTask();
directionTask.setDirection(directionFromAzimuth);
directionTask.execute();
```

Cada una de las tareas termina enviando los datos de los sensores utilizando un servicio web (Web Services). Para esto, debe indicarse la dirección o URL donde están disponibles los Servicios Web que utilizan el protocolo *Hessian*. Este protocolo (en los Web Services) es ampliamente conocido por su velocidad en la tasa de transmisión y la simplicidad de su utilización (como se explico en más detalle en el Capítulo 2).

Como cliente de estos servicios, en cada tarea se utilizó la implementación de la librería provista por la marca *Exadel* llamada *Flamingo* [Flamingo] *(*la cual es de código fuente abierto y licencia Apache).

En las clases abstractas de las tareas (*ServiceAsyncTask* y *ServiceTimerTask*) se realiza un “casteo” desde la URL del servicio, a una interfaz que posee los métodos que pueden ejecutarse en forma remota.

La configuración se realiza de la siguiente manera (Donde *MobileDataTransferService* es la interfaz con los métodos expuestos por el servidor en la URL especificada):

```
final HessianProxyFactory factory = new HessianProxyFactory();
try {
      this.dataInterface = (MobileDataTransferService)
            factory.create(MobileDataTransferService.class,
            URL_SERVICE);
} catch (MalformedURLException e) {
      e.printStackTrace();
}
```

La URL del servicio es la siguiente:

```
 private static final String URL_SERVICE =
                        "http://.../turista/remoting/SensorService";
```

Luego, la tarea concreta encargada de transmitir los datos al servidor, puede invocar a un método de la interfaz, desconociendo que el método se ejecute en forma remota, esto se realiza de la siguiente manera:

```
/**
 * Actualiza la informacion de la dirección para el dispositivo
 * tal que su IP se recibe por parámetro.
 * @param pIp Numero de la IP
 * @param pDirection Dirección (Norte, Sur, Este, Oeste, etc)
 * */
void updateDirectionInfo(String pIp, String pDirection);
```

El método updateDirectionInfo necesita dos parámetros: la IP del dispositivo móvil y la dirección (NORTE, SUR, ESTE, OESTE, NOR-ESTE, NOR-OESTE, SUD-ESTE, SUD-OESTE).

Una vez realizada todas las creaciones y configuraciones, veamos cómo funciona la recepción del cambio en tiempo real. A continuación se presentarán dos diagramas de secuencias que muestran como es la interacción desde que se comunica que un sensor cambio su valor, hasta que dicho cambio es comunicado al servidor.

En la Figura 15 se puede apreciar el diagrama de secuencias que se genera cuando se realiza un cambio en el acelerómetro. Esto es acorde a como está implementada la Aplicación Móvil Nativa del prototipo. Se puede apreciar que el sensor avisa al *SensorListener* que hubo un cambio en un dato de sensor, y en particular se determina por el tipo de evento que es un cambio en la orientación. Acorde a esto,

se ejecuta la tarea *ClientOrientationTask*, la cual envía la actualización correspondiente al servidor.

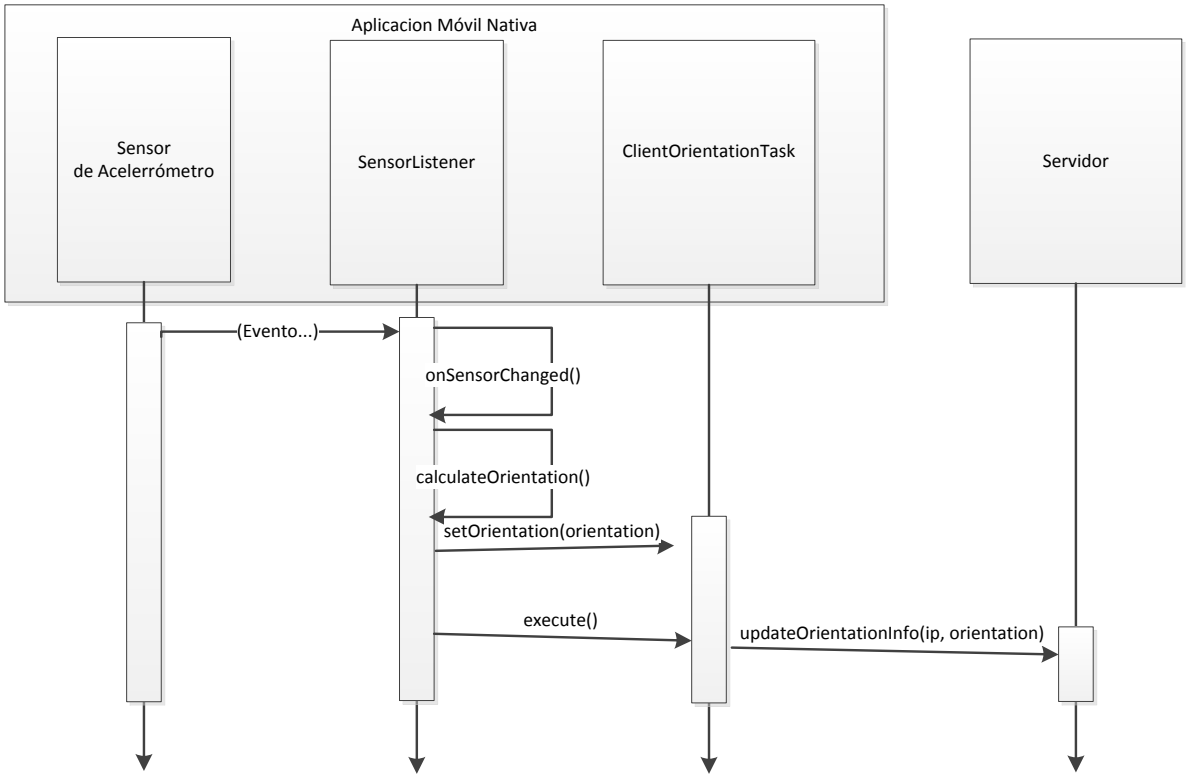


Figura 15: Diagrama de secuencia cuando se dispara un evento en el acelerómetro

La Figura 16 describe el flujo de ejecución para el caso que se registre un cambio en el sensor de GPS. El sensor avisa al *LocationListener* que hubo un cambio del GPS, y acorde a esto se ejecuta la tarea *ClientGPSTask*, la cual envía la actualización correspondiente al servidor.

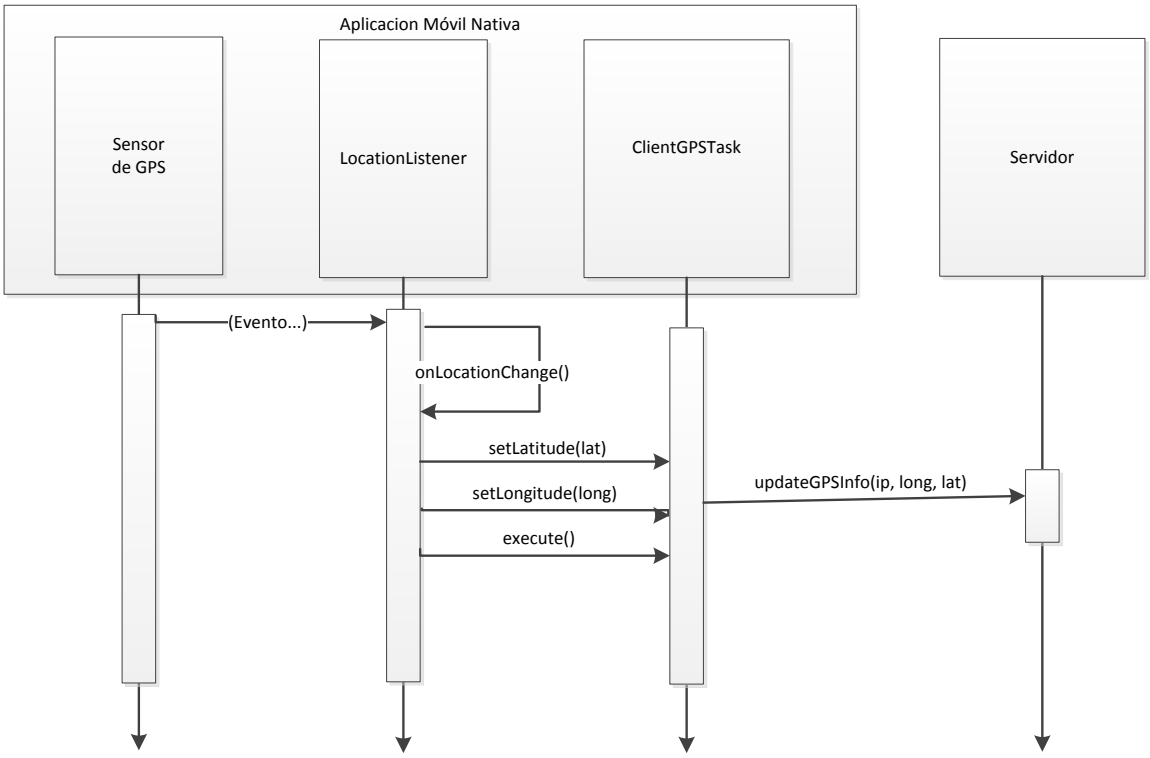


Figura 16: Diagrama de secuencia cuando se dispara un evento en el GPS

Como se pudo apreciar en las Figuras 15 y 16, ante cada evento de cambio de un sensor, se le comunica al listener adecuado (ya sea *SensorListener* o *LocationListener*) y este ejecuta la tarea adecuada. Notar que cada tarea es la encargada de mandar los datos del sensor al servidor.

Procesamiento de datos de los Sensores

Los datos de luz y temperatura se envían sin procesamiento alguno al servidor para que se muestren en el “modo” vertical de la aplicación web. Básicamente, para la luz se reciben desde el sensor 3 (tres) valores reales, se concatenan transformándolos al tipo string y se envía al servidor como un único valor. Mientras que para la temperatura, se recibe solo un valor real que se transforma a string para enviar al servidor.

Los datos detectados por el sensor de GPS se reciben en un objeto (clase *android.location.Location*) y contiene dos métodos para obtener la “nueva” posición (longitud y latitud). Estos datos son pasados a la tarea encargada de transmitir los datos al servidor, donde la tarea envía al servidor la IP del dispositivo, la latitud y la longitud. El resto de los datos almacenados en el objeto *Location* son descartados.

En cambio, los datos recibidos por los sensores del acelerómetro y magnetómetro son interpretados en la aplicación móvil nativa para calcular su orientación y dirección respectivamente, y se envían al servidor.

De los valores obtenidos del acelerómetro (matriz de enteros) se calcula la orientación: Vertical u horizontal, y se envía este dato como String.

De los valores obtenidos del magnetómetro (matriz de números reales) se calcula la dirección: Norte, Sur, Este, Oeste, Nor-Este, Nor-Oeste, Sud-Este o Sud-Oeste, y se envía al servidor como un String.

Estos cálculos en la aplicación nativa móvil minimizan las tareas del servidor y lo provee de información más simple de interpretar y procesar, optimizando su performance.

4.3. Servidor

Como se especificó previamente, el prototipo, además de la aplicación móvil nativa consta de la parte servidor. También, como se menciona en el comienzo de este capítulo, el prototipo implementa el modelo de arquitectura que se muestra en la Figura 3 del Capítulo 3. Por lo que el servidor está preparado para recibir información de uno o varios dispositivos que envían constantemente sus datos de sus sensores. Luego, con dicha información, el servidor envía al navegador correspondiente, la actualización de estos datos mediante la técnica o mecanismo denominado *push*. Este mecanismo será descrito en detalle posteriormente. Sin embargo, para el prototipo solo se probará el envió de un solo dispositivo (como se mostró en la Figura 8).

Se decidió implementar el lado servidor de la aplicación en el lenguaje de programación Java, específicamente utilizando la versión Java Enterprise Edition [JEE]. Esta aplicación se ejecuta en un servidor Apache Tomcat [Tomcat].

Básicamente, se pueden distinguir dos tareas principales en el servidor:

- Recibir los datos sensados de cada dispositivo (servicios)
- Actualizar el navegador correspondiente al dispositivo que envió esos datos (mecanismo *push*)

A continuación se detalla cómo se implementaron cada una de estas partes.

a) Servicios

La parte encargada de los servicios, se refiere al Web Service que expone los métodos que serán llamados por cada aplicación móvil nativa, necesarios para obtener la información de cada dispositivo. En el servidor implementado para este prototipo, se definieron cuatro métodos que serán los encargados de recolectar dicha información, estos son:

- Método encargado de recibir la información de luz y temperatura de cada dispositivo
- Método encargado de recibir la información geoposicional de cada dispositivo (GPS)
- Método encargado de recibir la orientación del dispositivo (vertical u horizontal)
- Método encargado de recibir la dirección o punto cardinal al que apunta el dispositivo

La Figura 17 muestra la interface (servicio), que define los métodos mencionados anteriormente, encargados de recibir la información a través de los parámetros.

```
public interface MobileDataTransferService {

    /**
     * Actualiza la informacion de luz y temperatura para el dispositivo tal que su IP se recibe por parámetro.
     * @param pIp Numero de la IP
     * @param pSensorData Datos de los sensores
     * */
    void updateLightTemperatureInfo(String pIp, Map<String, String> pSensorData);

    /**
     * Actualiza la informacion del GPS para el dispositivo tal que su IP se recibe por parámetro.
     * @param pIp Numero de la IP
     * @param pLatitude Latitud
     * @param pLongitude Longitud
     * */
    void updateGPSInfo(String pIp, Double pLatitude, Double pLongitude);

    /**
     * Actualiza la informacion de orientación para el dispositivo tal que su IP se recibe por parámetro.
     * @param pIp Numero de la IP
     * @param pOrientation Orientación (Vertical, Horizontal)
     * */
    void updateOrientationInfo(String pIp, String pOrientation);

    /**
     * Actualiza la informacion de la dirección para el dispositivo tal que su IP se recibe por parámetro.
     * @param pIp Numero de la IP
     * @param pDirection Dirección (Norte, Sur, Este, Oeste)
     * */
    void updateDirectionInfo(String pIp, String pDirection);
}
```

Figura 17: Los cuatro métodos del servicio que son llamados por las aplicaciones móviles nativas

Como se puede observar en la Figura 17, en cada método es necesario que se envíe la dirección IP. Ésta servirá al servidor para identificar unívocamente cada dispositivo, para luego poder enviar la respuesta al browser correspondiente. En el prototipo implementado, el mecanismo para manipular las peticiones de cada

dispositivo a través de la dirección IP, está implementado mediante un diccionario de datos. El diccionario de datos puede entenderse como una estructura que define como *clave* la IP del dispositivo y como *valor*, una clase llamada *StateData* que contiene en sus propiedades la información de los sensores de un dispositivo específico. La clase que posee la información sensada es de donde el servidor obtiene y setea los valores correspondientes de cada sensor.

La clase *StateData* tiene las siguientes propiedades que serán accedidas y modificadas por el servidor a medida que vaya recibiendo la información de los sensores de cada dispositivo. Los valores que se guardan son los siguientes:

- Orientación (información que indica si el dispositivo está en una posición vertical u horizontal)
- Luz
- Dirección (información que muestra hacia dónde está apuntando el dispositivo, los posibles valores son: Norte, Sur, Este, Oeste, Nor-Este, Nor-Oeste, Sud-Este, Sur-Oeste)
- Temperatura
- Latitud (información geoposicional)
- Longitud (información geoposicional)

La Figura 18 muestra de manera visual como son almacenados los datos de cada dispositivo asociados a cada sensor. De esta manera, luego cada navegador puede estar interesado en mostrar datos diferentes o de diferentes dispositivos. Se puede apreciar en dicha figura que hay datos de prueba cargados para dos dispositivos distintos.

CLAVE: Dirección IP	VALOR: Objeto *StateData* con la información de cada sensor
191.190.28.3	**StateData** ***Propiedad*** / ***Valor*** Luz: 1 Temperatura: 28 Latitud (GPS): -34.4322341 Longitud (GPS): -57.4564451 Orientación: Vertical Dirección: Norte
191.169.12.4	**StateData** ***Propiedad*** / ***Valor*** Luz: 0.5 Temperatura: 28 Latitud (GPS): -33.3211122 Longitud (GPS): -54.3691881 Orientación: Horizontal Dirección: Nor-Oeste

Figura 18: Esquema de ejemplo de la estructura de datos utilizada por el servidor para el manejo de la información de los sensores.

La implementación de esta estructura de datos en el servidor implica que sea una estructura manejada en memoria, lo que la convierte en *volátil*. Esto quiere decir que la información será accesible y visible únicamente mientras el servidor se esté ejecutando. Luego, cuando el servidor no se encuentre disponible (por ejemplo está apagado), toda información registrada anteriormente se pierde. Una posibilidad, si se quisiera persistir la información manipulada y enviada por los dispositivos, es utilizar una base de datos o archivos para almacenar dicha información. Se eligió esta solución volátil por simplicidad a la hora de implementar el prototipo ya que el foco estaba puesto en mostrar la actualización de los datos contextuales.

A continuación se describe en detalle el funcionamiento de cada método presentado en la Figura 17:

- **Información de luz y temperatura**
  El dispositivo móvil recibe la información actualizada de luz y temperatura cada 5 segundos. Para Android, la temperatura consta de un número real y los datos de la luz están formados por 3 números reales.

  La aplicación nativa transforma el dato de temperatura a un string, mientras que con los números de la luz, la aplicación conforma un string separando los valores mediante el carácter “|”. Luego, la aplicación arma un diccionario para enviar estos datos utilizando como claves los strings

"Light" y "Temperature" y como valor los string anteriormente creados, y lo envía al servidor mediante un web service.

En el servidor, el método encargado de obtener la información de luz y temperatura sensada y enviada por cada dispositivo, recibe como parámetro: La IP del dispositivo y el diccionario de datos con la información de luz y temperatura.
El servidor actualiza los datos obtenidos del diccionario de datos en el objeto *StateData* (actualiza la estructura de datos correspondiente a la IP del dispositivo que envió la información).

Luego, mediante el mecanismo push, actualiza los datos referentes a la luz y temperatura en el browser, verificando primeramente si el dispositivo se encuentra en forma vertical. De lo contrario, la información será actualizada cuando el dispositivo pase a orientación vertical.

- **Información geoposicional (GPS)**
  El método encargado de obtener la información referente a la posición geográfica en la que se encuentra el dispositivo, recibe como parámetro además de la IP, la longitud y la latitud. Éstos son los datos que indican las coordenadas geográficas de cada dispositivo.

  Cuando el servidor recibe la información, en principio actualiza el objeto *StateData* del dispositivo con la nueva longitud y latitud. Segundo, verifica la orientación del dispositivo; si ésta es horizontal, obtiene los puntos de interés más cercanos a la posición actual a un radio de 1000 metros. Si está orientado verticalmente, consulta al objeto *StateData* solicitando la dirección a la que apunta el dispositivo (norte, sur, este, oeste, noreste, sudeste, sudoeste y noroeste), y obtiene los puntos de interés ubicados en esa dirección y a una distancia no mayor a 1000 metros. Por último realiza un *push* al browser del dispositivo con la orientación, la lista de POI y, de ser necesario, la información de luz y temperatura.

- **Información de la orientación**
  Éste método será el encargado de obtener la información referente a la orientación del dispositivo. El parámetro pOrientation es una cadena que contiene el valor Landscape (Horizontal) o Portrait (Vertical).
  Lo primero que realiza el servidor al recibir la nueva información es verificar en el objeto *StateData* si la orientación del dispositivo cambió, de ser así debe realizar la lógica correspondiente en el navegador y actualiza en el *StateData* del dispositivo, la nueva orientación.
  Cualquiera sea la nueva orientación, el servidor obtiene una lista con los puntos de interés más cercanos y hace un push para que el navegador se actualice con estos datos. La diferencia radica en que si la nueva

orientación es vertical, además de la lista de interés, debe enviar al navegador los datos de luz y temperatura.

- **Información de la dirección**
  Esta información solamente será tenida en cuenta cuando el dispositivo se encuentre orientado verticalmente.
  El método además de la IP, obtiene la dirección o punto cardinal al que apunta ahora el dispositivo. Primero, el servidor verifica que el dispositivo se encuentre en orientación vertical, luego obtiene los puntos de interés más cercanos en esa dirección y los envía al navegador mediante *push* para que se actualice. Por último, actualiza el nuevo valor en el objeto *StateData* correspondiente al dispositivo.

b) Mecanismo push

Cuando el servidor recibe información de los sensores de un dispositivo, luego de almacenar esta información indexándola por IP del dispositivo (identificador único), debe notificar al navegador Web que debe actualizarse enviando la nueva información a través de un canal (channel). En el navegador Web, al recibir esa información, se disparará un evento en lenguaje javascript y recibirá por parámetro la información a actualizar.

Existe otro mecanismo para poder actualizar navegador Web llamado *polling*. Este método requiere que el cliente, en este caso la aplicación web, solicite al servidor la información actualizada cada “x” periodo de tiempo. Si bien, es un método más fácil de implementar, tiene como desventaja que es poco eficiente, ya que la aplicación web está constantemente solicitando información, que quizás, aún no ha cambiado. Esto podría generar una sobrecarga en el servidor que es el encargado de atender requerimientos tanto de la aplicación nativa, como de la aplicación web. Por esta razón, se decidió optar por la tecnología *push* en lugar de *polling*.

En la Figura 19 y 20 se puede apreciar el mecanismos push que hace el Servidor al navegador para enviar el nuevo dato sensado. Se puede apreciar que en ambos diagramas el método enviado al browser es el mismo.

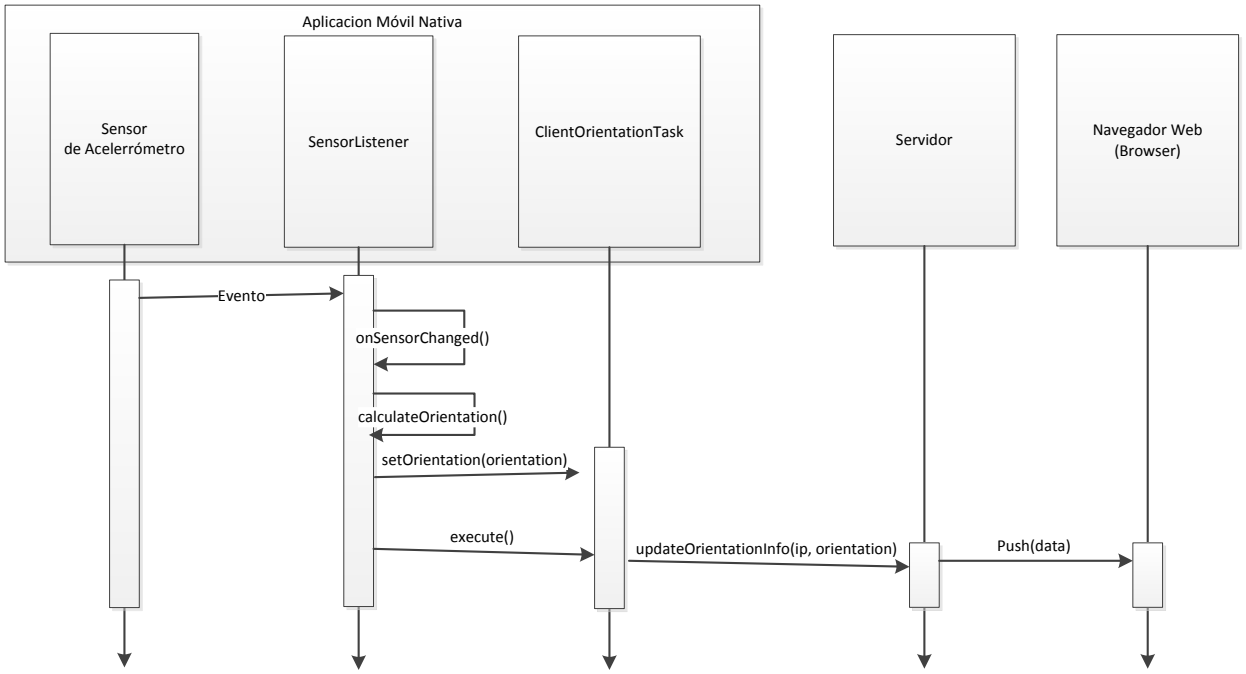


Figura 19: Diagrama de secuencia cuando se dispara un evento en el acelerómetro y cómo llegan estos datos al navegador web.

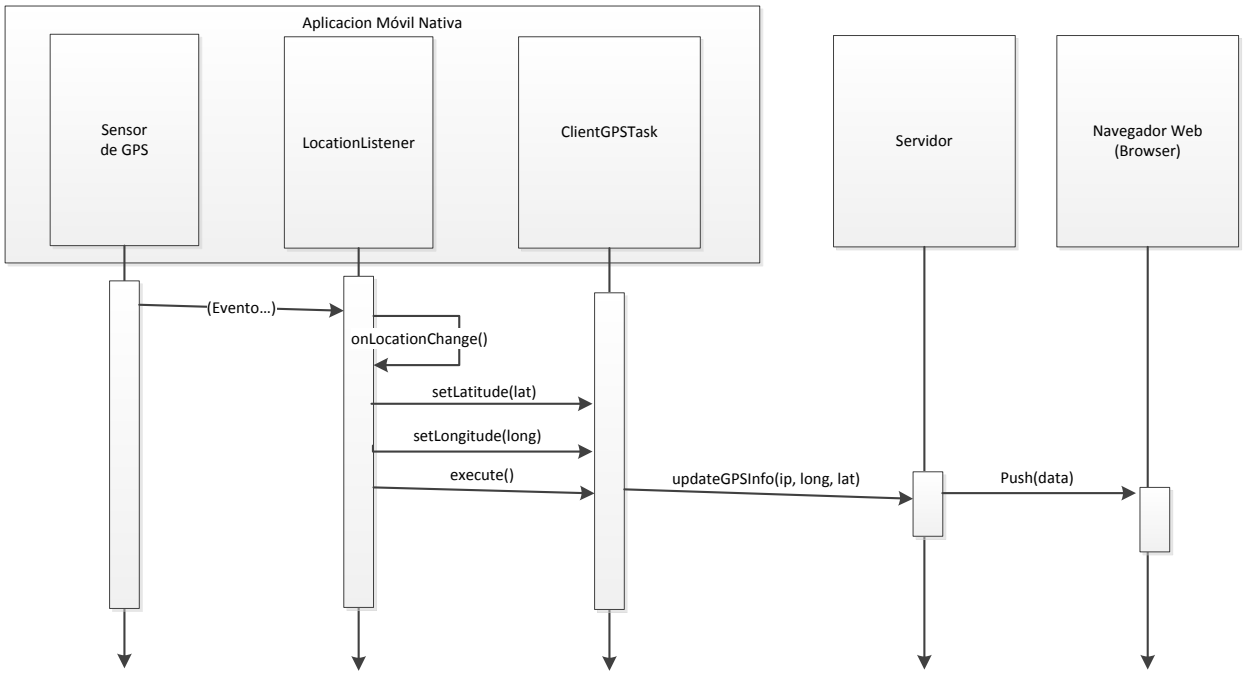


Figura 20: Diagrama de secuencia cuando se dispara un evento en el GPS y cómo llegan estos datos al navegador web.

4.4. Browser o navegador web (Aplicación Web)

La aplicación web se ejecuta sobre un browser (navegador web) tradicional y es, en definitiva, aquello que interactúa con el usuario final. Su implementación está basada sobre JSF (*Java Server Faces*) [JSF] para los componentes visuales. Específicamente, se utilizan los componentes de la librería Primefaces [Primefaces]. También se hace uso del framework Spring WebFlows [Spring], el cual permite utilizar el patrón MVC mediante la configuración de XML's. Tanto las paginas implementadas en JSF (parte visual), como los servicios implementados en Java, residen y se ejecutan en un servidor de aplicaciones JEE, como por ejemplo Apache Tomcat, JBoss, GlassFish, WebSphere Application Server, etc. En nuestro caso, para probar la implementación del prototipo, se utilizó Apache Tomcat.

Cuando se intenta acceder a la página principal de la aplicación, Spring WebFlows realizará una validación para este requerimiento. Si el usuario (que se identificara con su IP) posee información asociada en el servidor, entonces la respuesta al requerimiento será la página principal. En cambio, si el usuario no posee información asociada en el servidor, se redireccionará hacia una página de error informando la situación. En el Capítulo 3, en la Sección 3.3, se pueden observar los diagramas de secuencia que describen estas situaciones (Figuras 6 y 7).

La vista principal posee un canal push mediante el cual "escuchará" los mensajes que envía el servidor. Este canal necesita tener definida una función javascript encargada de "interpretar" los datos recibidos en el mensaje y realizar las acciones necesarias. De esta forma, cuando el canal recibe un mensaje, se ejecuta la función con los datos recibidos como parámetro; y de acuerdo a cómo se encuentre orientado el dispositivo (vertical u horizontal) muestra/oculta distintas secciones de la vista. Esto fue mostrado en la Figuras 19 y 20, donde este mecanismo fue usado para notificar el cambio de los datos de los sensores de GPS y orientación.

En la Figura 21 se muestra un ejemplo de cómo se muestra la aplicación web en modo horizontal. El ícono azul representa la posición actual del dispositivo, mientras que los puntos en rojo representan los lugares cercanos (POI). Estos puntos mostrados se encuentran dentro del radio de alcance (mencionado anteriormente cuando se presento la Figura 9). Es decir, a una distancia menor a 1000 metros de la posición actual del usuario.

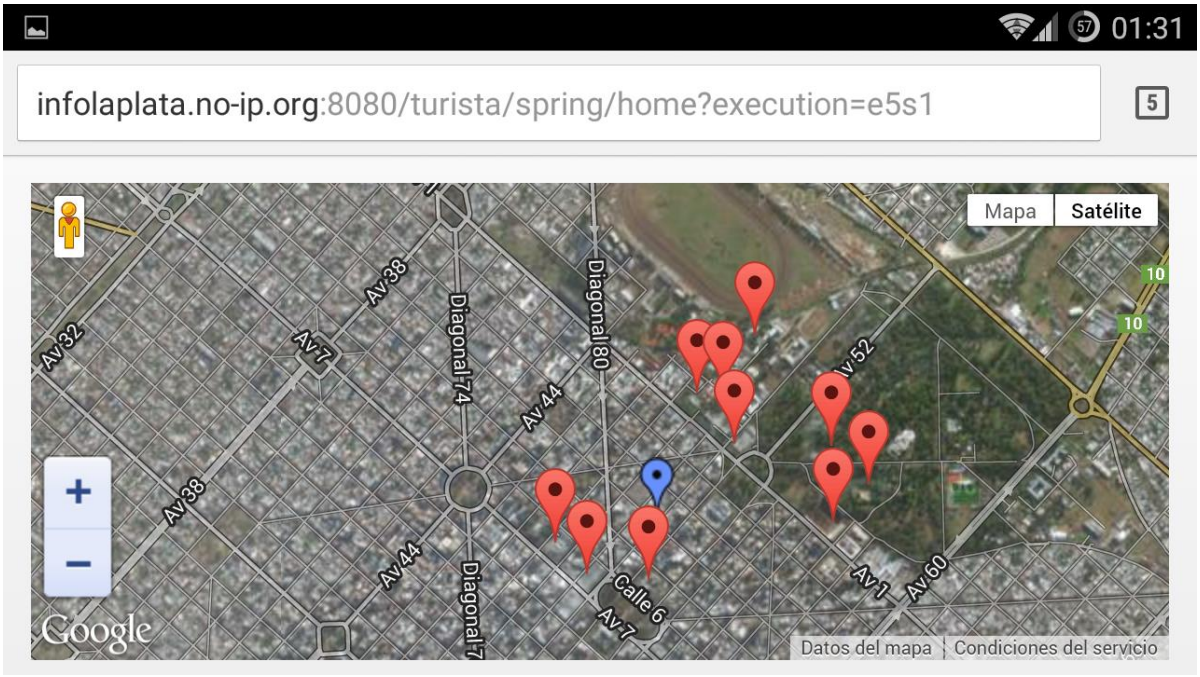


Figura 21: Dispositivo orientado en forma horizontal

En la Figura 22, se muestra un ejemplo de la aplicación funcionando en modo vertical. En la parte superior se encuentra el detalle de los lugares cercanos, en la dirección hacia donde apunta el dispositivo (descripción del punto de interés y distancia en línea recta). En la parte inferior se observan los datos correspondientes al dispositivo (IP), temperatura y luz. Notar que estos POI varían acorde al ángulo de visión del usuario (orientación).

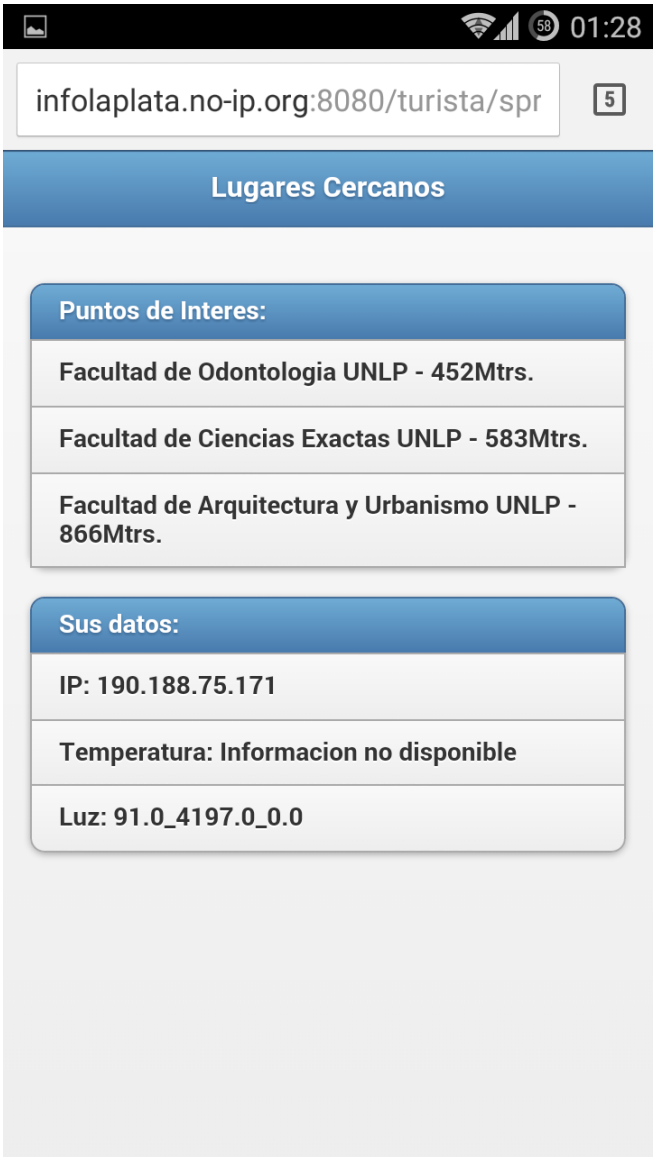


Figura 22: Dispositivo orientado en forma vertical.

Capítulo 5: Simulación del prototipo

En este capítulo se mostrarán diferentes ejemplos para ejemplificar el funcionamiento del prototipo. Cuando la aplicación nativa esté ejecutándose y enviando información de los sensores, al acceder a la aplicación web, su contenido dependerá de la orientación del dispositivo. Como fue explicado en los capítulos anteriores, si está orientado horizontalmente, se mostrará un mapa con la posición actual del dispositivo mediante un marcador azul, y se indicará con marcadores rojos los puntos de interés cercanos a su posición. En cambio, cuando el dispositivo está orientado de manera vertical, se muestra la lista con los puntos de interés cercanos en dirección a donde está apuntando el dispositivo.

5.1. Aplicación Web Funcionando

Para demostrar el completo funcionamiento de la aplicación y su respectivo comportamiento cuando el valor de cada sensor se modifica, se utilizaran como ejemplo distintos casos de prueba. Por ejemplo, dadas dos ubicaciones arbitrarias “A” y “B”, dentro de la ciudad de La Plata, se mostraran diferentes escenarios y el comportamiento de la aplicación de acuerdo a su orientación (vertical u horizontal) y dirección (norte, sur, este, oeste, sudeste, sudoeste, noreste, noroeste).

La Aplicación Web se accede desde cualquier navegador que tenga instalado el dispositivo móvil (celular), en el navegador se debe escribir la siguiente url:

http://www.infolaplata.no-ip.org:8080/turista

Supongamos que el usuario se instala y deja ejecutando la Aplicación Nativa Móvil (se describen más detalles en la Sección 5.2). A continuación se presentan diferentes escenarios que muestran la información que recibe el usuario de la Aplicación Web mediante el bvowser, una vez que está funcionando dicha Aplicación Nativa Móvil.

Escenario 1

Veamos cómo se comporta la Aplicación Web cuando se tiene la siguiente información:

Ubicación del usuario: Posicionado en la intersección de las calles 4 y 50
Orientación del dispositivo: Horizontalmente
Angulo de Visión: Noreste

La aplicación detecta que la orientación es horizontalmente, por lo tanto, desplegará un mapa indicando mediante un ícono azul la posición actual del

dispositivo, y en rojo, los puntos de interés cercanos a esa posición (esto es, a no más de 1000 metros de distancia de la ubicación actual). En este caso el ángulo de visión no es considerado para brindar la información en el mapa.
Las Figuras 23 y 24 reflejan el escenario descripto anteriormente. Se puede apreciar que la Figura 23 posee más zoom respecto de la Figura 24.

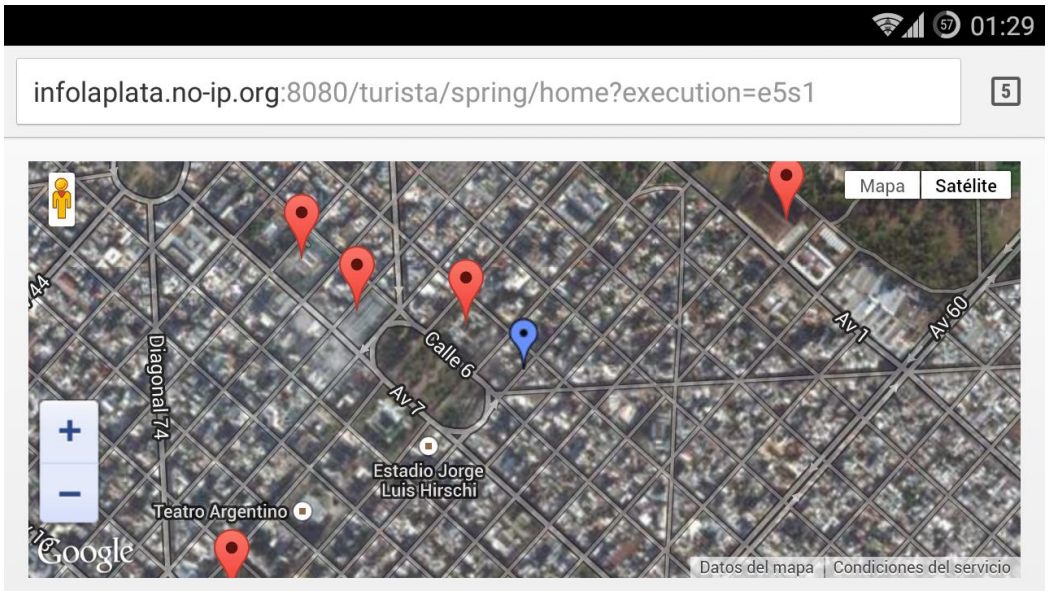


Figura 23: Ejemplo cuando el dispositivo está orientado en forma horizontal en la intersección de las calles 4 y 50.

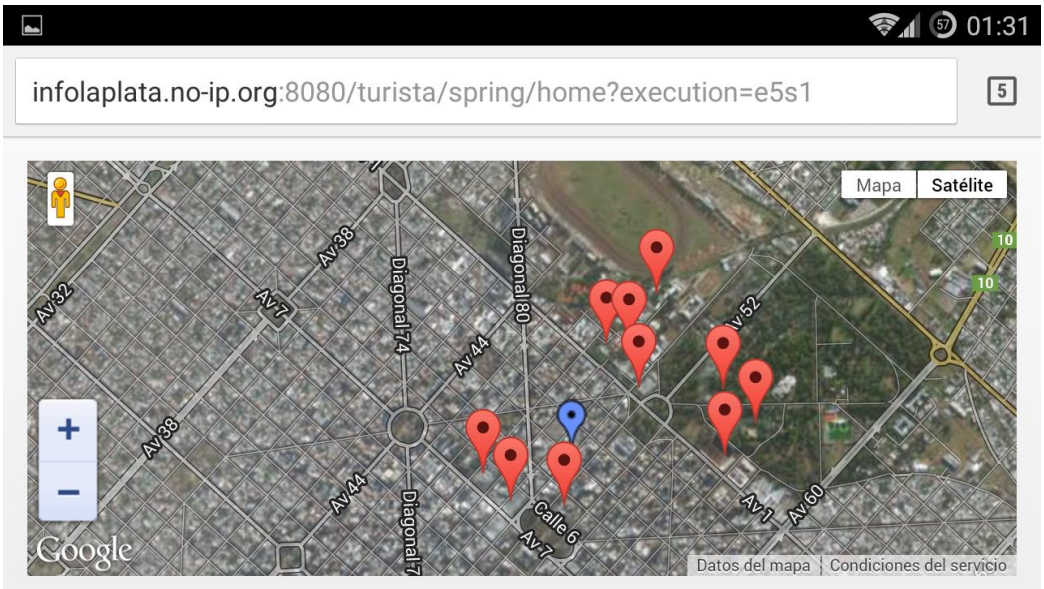


Figura 24: Mismo ejemplo cuando el dispositivo está orientado en forma horizontal en la intersección de las calles 4 y 50 (con menos zoom respecto de la Figura 23).

Escenario 2

Supongamos que luego de tener el *Escenario 1*, el usuario decide cambiar la orientación del dispositivo a vertical, en este caso pasa a tener la siguiente información:

Ubicación del usuario: Posicionado en la intersección de las calles 4 y 50
Orientación del dispositivo: Verticalmente
Angulo de Visión: Noreste (la Figura 25 ejemplifica este ángulo)

Figura 25: Ejemplificación del ángulo de visión considerado para este escenario.

Como ya se había mencionado anteriormente, cuando la orientación es vertical la Aplicación Web despliega una lista con los puntos de interés cercanos (a no más de 1000 metros de distancia), respecto de la posición y el ángulo de visión.

Supongamos que actualmente la información de los sensores con los que se cuenta es la siguiente:

Temperatura: Información no disponible *(la información no disponible se debe a que el dispositivo de prueba no cuenta con dicho sensor)*
Luz: 91.0_4197.0_0.0

Las Figura 26 muestra la información que visualiza el usuario en la Aplicación Web. Se puede apreciar tres puntos de interés cercanos y en su ángulo de visión y los datos de los sensores.

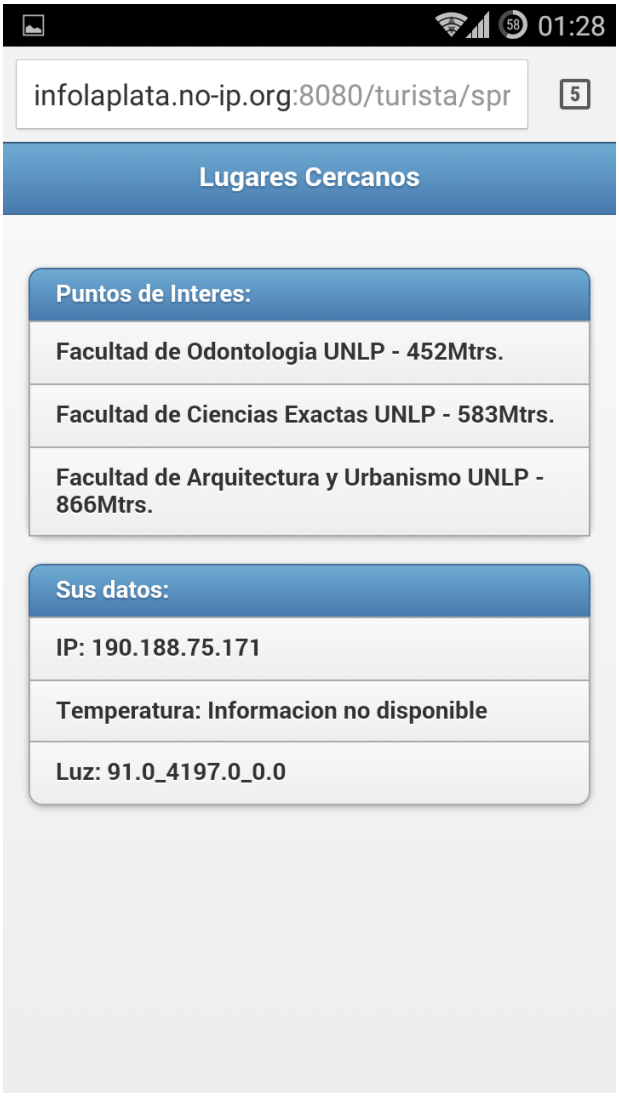


Figura 26: Ejemplo de cuando el dispositivo está orientado en forma vertical, apuntando hacia el noreste y posicionado en la intersección de las calles 4 y 50.

Escenario 3

Supongamos que teniendo el *Escenario 2*, el usuario cambia su ángulo de visión a “este”, es decir, se tienen los siguientes datos:

Ubicación del usuario: Posicionado en la intersección de las calles 4 y 50
Orientación del dispositivo: Verticalmente
Angulo de Visión: Este (la Figura 27 ejemplifica este ángulo)

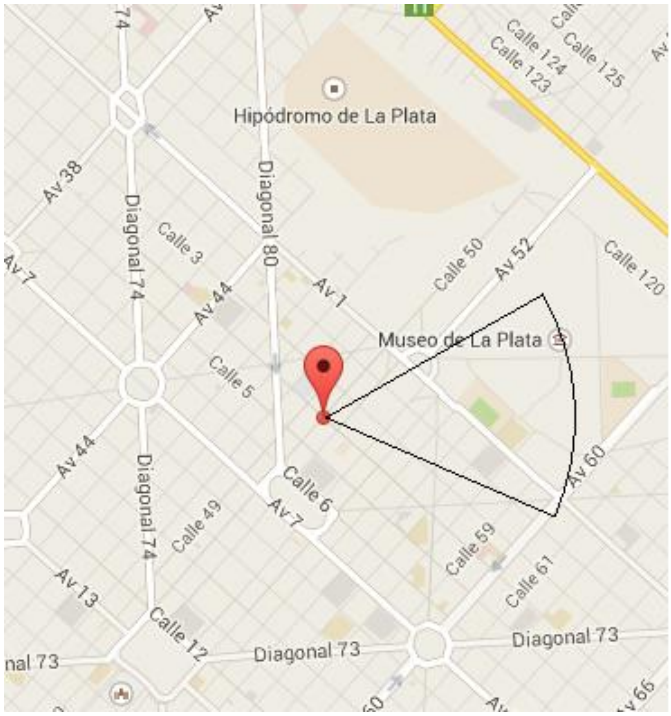


Figura 27: Ángulo de visión del dispositivo apuntando hacia el este y posicionado en la intersección de las calles 4 y 50.

Al cambiar el ángulo de visión cambia también la luz que detecta el dispositivo quedando la siguiente información:

Luz: 64.0_4224.0_0.0

Como ya se había mencionado anteriormente, cuando la orientación es vertical la Aplicación Web despliega una lista con los puntos de interés cercanos (a no más de 1000 metros de distancia), respecto de la posición y el ángulo de visión. En la Figura 28 se puede apreciar la vista que recibe el usuario acorde a este escenario descripto. Se puede apreciar que los puntos de interés desplegados cambiaron respecto a los mostrados en el *Escenario 2*, esto se debe al cambio en el ángulo de visión. Además, se puede apreciar el cambio en la información de la luz. Internamente estos datos se actualizan a través de la aplicación nativa instalada en el dispositivo que envía constantemente la información que detecta, en este caso la luz.

Figura 28: Ejemplo de cuando el dispositivo está orientado en forma vertical, apuntando hacia el este y posicionado en la intersección de las calles 4 y 50.

Escenario 4

Supongamos que otro usuario recorre la ciudad y en un momento dado se posiciona en 1 y 50 orientando el dispositivo de manera horizontal. En este caso se tiene la siguiente información:

Ubicación del usuario: Posicionado en la intersección de las calles 1 y 50
Orientación del dispositivo: Horizontal
Angulo de Visión: Este

Acorde a estos datos, como se mencionó anteriormente al tener una orientación horizontal, la Aplicación Web muestra un mapa indicando mediante un ícono azul la posición actual del dispositivo, y en rojo, los puntos de interés cercanos a la posición (esto es, a no más de 1000 metros de distancia de la ubicación actual). Las Figura 29 refleja este escenario.

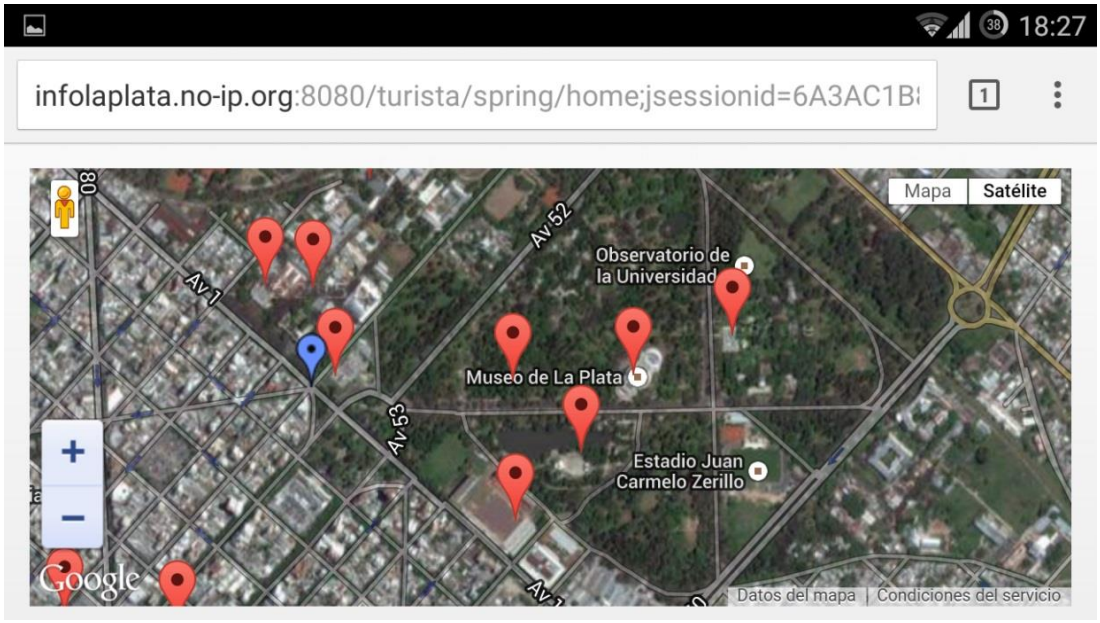


Figura 29: Ejemplo cuando el dispositivo está orientado en forma horizontal en la intersección de las calles 1 y 50.

Escenario 5

Luego del *Escenario 4*, supongamos que el usuario cambia la orientación del dispositivo a vertical, en este caso se tendría la siguiente información:

Ubicación del usuario: Posicionado en la intersección de las calles 1 y 50
Orientación del dispositivo: Vertical
Angulo de Visión: Este (la Figura 30 ejemplifica este ángulo)

Figura 30: Ángulo de visión del dispositivo apuntando hacia el este y posicionado en la intersección de las calles 1 y 50.

Al cambiar la orientación cambia también la luz que detecta el dispositivo quedando la siguiente información:

Luz: 59.0_3412.0_0.0

La Figura 31 muestra la vista de la Aplicación Web acorde a los datos mencionados para este escenario. Se puede apreciar que se actualizaron los datos de los puntos de interés cercanos como así también la temperatura. Se puede apreciar que la IP cambio respecto del *Escenario 3*.

Figura 31: Ejemplo de cuando el dispositivo está orientado en forma vertical, apuntando hacia el este y posicionado en la intersección de las calles 1 y 50.

Escenario 6

Supongamos que estando en el *Escenario 5*, el usuario cambia de ángulo de visión, y ahora mira hacia el sudeste. En este caso la información que se tiene es la siguiente:

Ubicación del usuario: Posicionado en la intersección de las calles 1 y 50
Orientación del dispositivo: Vertical
Angulo de Visión: Sudeste (la Figura 32 ejemplifica este ángulo)

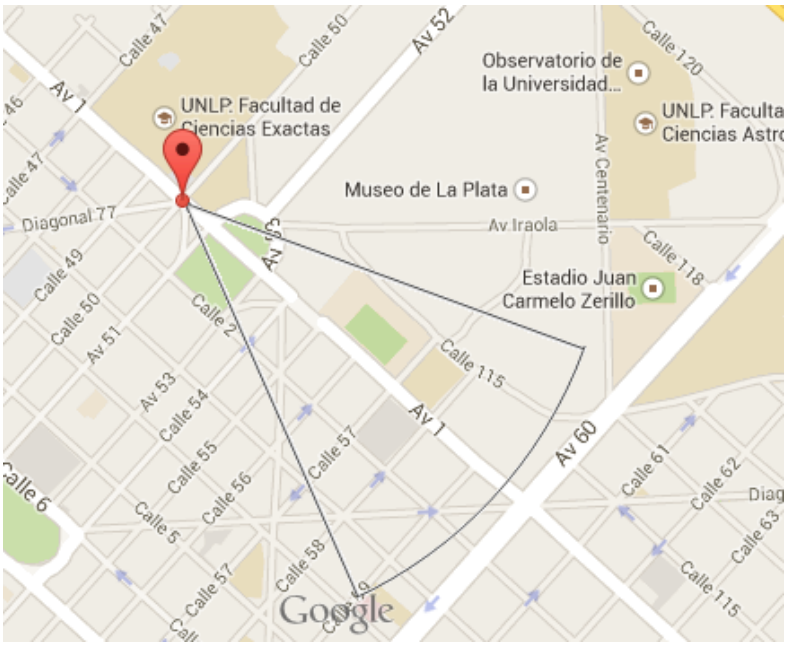


Figura 32: Ángulo de visión del dispositivo apuntando hacia el sudeste y posicionado en la intersección de las calles 1 y 50.

Como viene pasando en los casos anteriores, al cambiar el ángulo de visión la luz se ve afectada, en este caso el valor que se detecta es:

Luz: 41.0_5875.0_0.0

En la Figura 33 se puede apreciar que solo se detecta un punto de interés en el ángulo de visión y que se ha actualizado el dato de la luz.

Figura 33: Ejemplo de cuando el dispositivo está orientado en forma vertical, apuntando hacia el sudeste y posicionado en la intersección de las calles 1 y 50.

De esta manera se pudieron apreciar diferentes escenarios de uso de la Aplicación Web y cómo la misma reacciona para mostrarle al usuario diferente tipo de información acorde a su posición, orientación del dispositivo y ángulo de visión. También se pudo apreciar cómo dos usuarios distintos pueden utilizar la aplicación.

5.2. Aplicación Nativa Android

Luego de instalar la Aplicación Nativa Android aparecerá en el menú de aplicaciones el acceso directo que permitirá ejecutar la aplicación. Este se puede apreciar en la Figura 34, el cual queda instalado con el nombre de *Sensor Listener.* Esta aplicación está instalada en el celular y como ya se explica anteriormente es la encargada de mandar la actualización de los datos de los sensores y el gps.

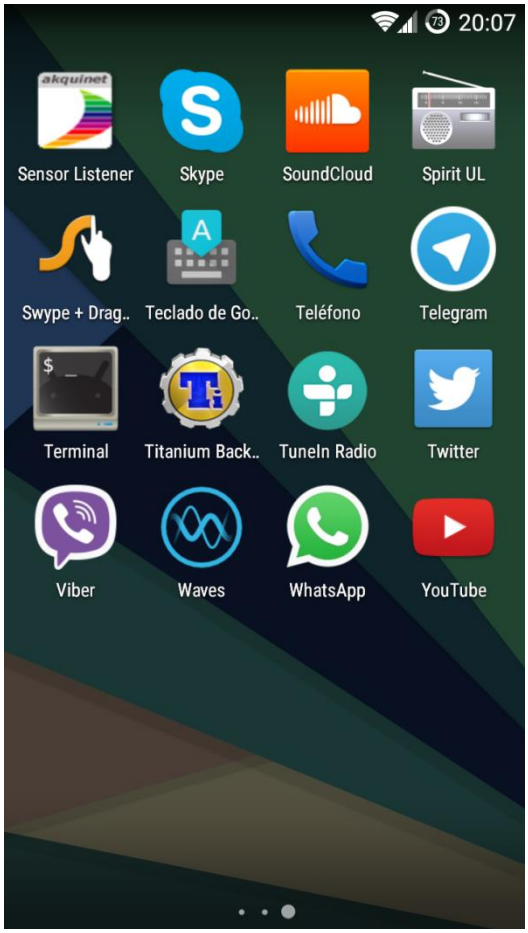


Figura 34: Menú de aplicaciones con el acceso directo de la aplicación nativa (Sensor Listener).

En el caso de acceder a la Aplicación Web y aún no se está ejecutando la Aplicación Nativa Móvil, en el navegador aparecerá el mensaje mostrado en la Figura 35 donde se le indica al usuario que dicha aplicación no está instalada. Esto se debe a que no se cuenta aun con datos del usuario, por ejemplo, datos del GPS para determinar donde está ubicado y acorde a eso brindarle información.

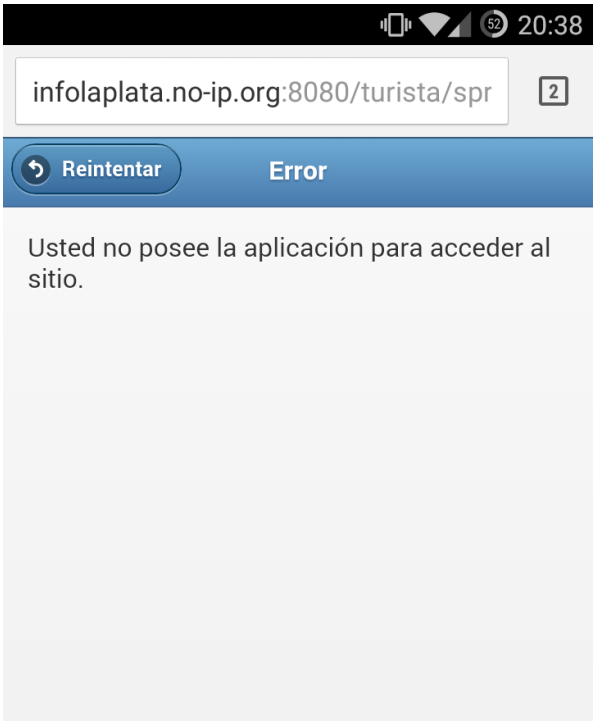


Figura 35: Mensaje que se muestra en el navegador cuando aún no se recibieron datos de los sensores para ese cliente.

Capítulo 6: Conclusiones y Trabajos Futuros

La investigación realizada en esta tesis trata de explicar al lector, en principio, lo que es una aplicación móvil sensible al contexto, la motivación por la cual se realiza este trabajo y una descripción general. Luego lo provee de la información técnica básica (Capítulo 2: Background), para poder abordar y entender de forma más clara los siguientes capítulos. Posteriormente se comienza por describir cuál fue la arquitectura implementada para poder llevar a cabo el prototipo y se la compara con otras soluciones similares. Por último se explica y se muestra mediante imágenes el funcionamiento del prototipo implementado.

Todo esto fue realizado para brindar una solución eficiente a la hora de crear y obtener un contexto real del dispositivo del usuario que está ejecutando la aplicación, a través de los sensores del mismo. A la vez que es una arquitectura que tiene la ventaja de poder adaptarse a los cambios tecnológicos que vayan surgiendo con el paso de los años. Es importante aclarar esto último ya que se trata de una tecnología que está en plena evolución actualmente y se está llevando a una estandarización (a través de HTML) para poder acceder directamente desde la web a los sensores de un dispositivo.

En un principio, el desarrollo de tales aplicaciones, significaba para los desarrolladores un trabajo engorroso ya que cada dispositivo era en cuestión un *mundo* distinto, distintos sistemas operativos (si es que tenían), distintas interfaces de programación, diseño adaptable al dispositivo, etc. Por ejemplo, en el caso del hardware, dispositivos con poca capacidad de procesamiento, poca memoria, carencia de sensores, etc. Un gran avance se fue produciendo en este campo en cuanto a la facilidad para programar, luego de que se produzcan cambios a nivel comercial y la aparición de los Smartphones, donde ya prácticamente su uso es mundial y la mayoría de los dispositivos se ven abarcados por los sistemas operativos Android y IPhone. De esta manera, la programación de aplicaciones se vio afectada para bien, ya que estos sistemas proveen interfaces de programación simples de utilizar para crear aplicaciones.

El prototipo implementado, si bien fue creado para representar un contexto y servir como ejemplo de un sistema que utiliza la arquitectura propuesta, es simplemente eso, un solo tipo de contexto creado para que el sistema responda de tal o cual manera a los cambios de ciertos sensores de un dispositivo. Esto quiere decir que podrían existir muchos tipos de contexto, y estos contextos deben adaptarse a las necesidades de los sistemas. En nuestro prototipo, solamente utilizamos los sensores y la información del usuario que consideramos necesarios para mostrar la información correspondiente en la aplicación web. Si necesitáramos más información, probablemente podríamos utilizar otros sensores que nos brinden dicha información, o por ejemplo entrada de datos del usuario para poder obtener información personal, etc. En todo caso, sería otro tipo de contexto, y los contextos se crean con información que puede provenir de distintas fuentes. En [Emmanouilidis et al., 2013], se presenta una taxonomía de contexto que describe diferentes tipos de contexto dentro de las guías móviles, esta taxonomía se puede observar en la Figura 36

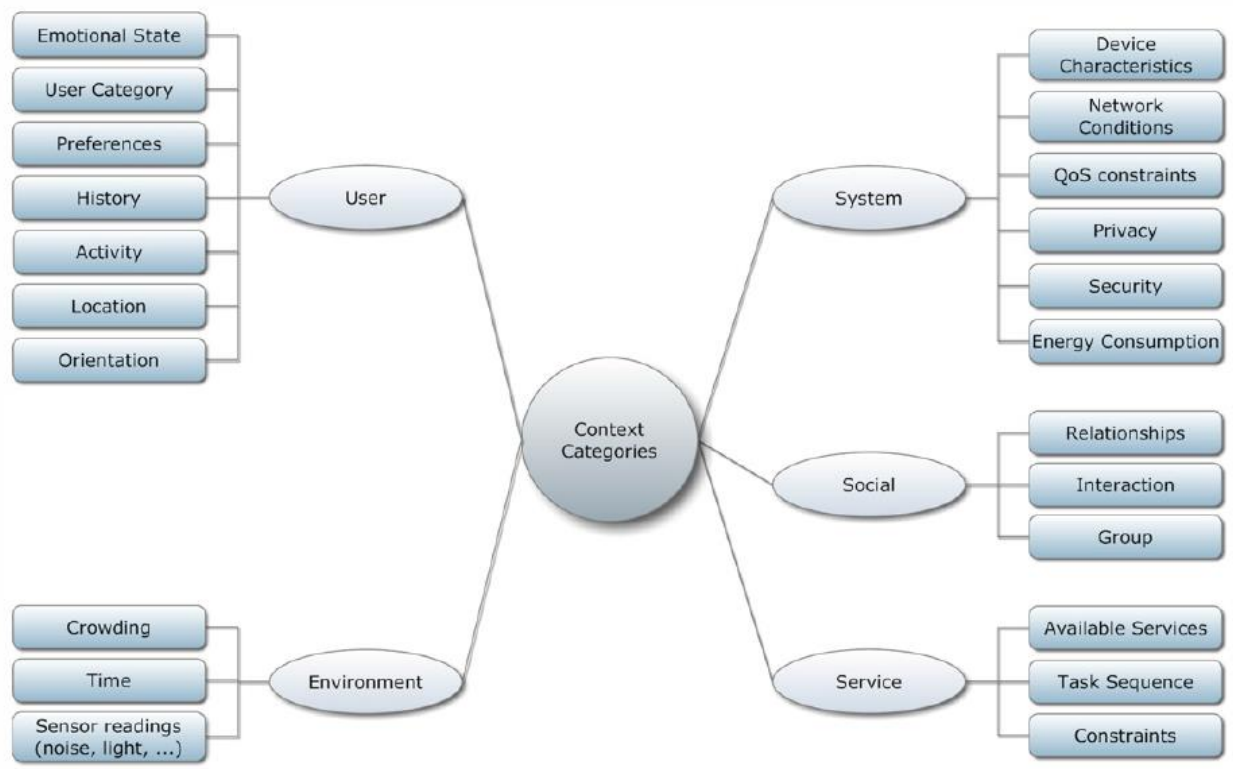


Figura 36: Taxonomía de contexto que describe diferentes tipos de contexto [Emmanouilidis et al., 2013].

Como se puede observar en la Figura 36, existe una diversidad importante de información referente a cada categoría de contexto (usuario, sistema, entorno, social o servicios), por lo que combinando esta información se podrían crear múltiples contextos para que se adapte cada uno a la aplicación que lo requiera. Se puede observar también que algunos datos se pueden sacar de los sensores de un dispositivo, o entrada de datos del usuario, o del sistema operativo, etc. En detalle se puede ver que esta información también fue utilizada en nuestro prototipo, por ejemplo la ubicación y la orientación del usuario, que es información sensada, al igual que la luz o temperatura, que también proviene de los sensores pero se observa que corresponden a otra categoría, la del entorno del usuario. Por el contrario, se puede apreciar que mucha información que podría estar disponible, no es tenida en cuenta por el prototipo, como se explicó anteriormente, esto se debe a que cada aplicación usa los datos necesarios para crear el contexto correspondiente. Sin embargo, la arquitectura propuesta podría soportar cualquier combinación de los mismos.

Para tener una idea más real de los contextos que actualmente están en uso en las distintas aplicaciones móviles disponibles, en [Emmanouilidis et al., 2013] se hace un relevamiento de las mismas. En la Figura 37 se puede observar el uso de cada uno de los contextos. Se puede apreciar que si bien hay una taxonomía extensa de contexto, en la realidad las aplicaciones no los utilizan. El contexto más utilizado es el de posicionamiento. Esto da una idea de que en esta área todavía falta mucho por explorar y analizar para que este tipo de aplicaciones contextuales se vuelvan comunes.

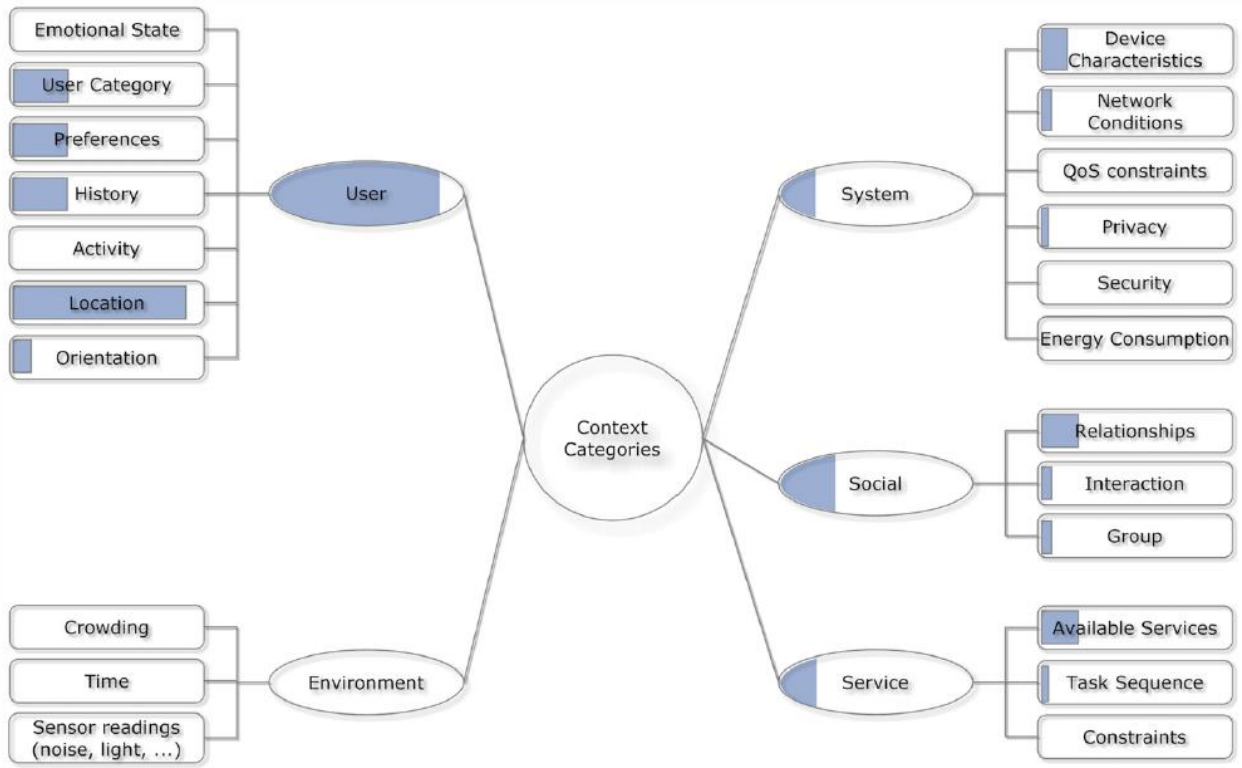


Figura 37: Tipos de contexto más utilizados según [Emmanouilidis et al., 2013].

Como se menciono anteriormente, la Figura 37 muestra los tipos de contexto más utilizados en la actualidad, esto se debe no solo a las aplicaciones que existen, sino que también hay que tener en cuenta que existe información que sería bastante útil pero que en ciertas ocasiones o al día de hoy, no existe soporte para adquirirla (falta de sensores, interface de programación escasa, poca estandarización, diversidad de dispositivos y sistemas operativos, etc).

Cabe destacar que la arquitectura propuesta fue pensada para que pueda adaptarse a cualquier contexto, es decir, si el día de mañana, surge una nueva manera de obtener la información de un dispositivo, o se implementa una estandarización en HTML. La arquitectura es independiente de la manera en que se obtienen los datos, simplemente la aplicación nativa debe enviar estos datos al servidor, y luego el servidor actualizar la web.

Un tema planteado en esta tesis es el avance del uso de sensores en HTML5 [HTML5-mobile-problems]. Es decir, disponer dentro del browser el acceso a sensores para no tener la necesitad de tener una aplicación nativa. Existen en HTML5 numerosas API's para la manipulación de dispositivos definidas por la W3C para su implementación y estandarización. Entre ellas podemos mencionar:

- Red
- Filesystem
- Administración de la energía
- Wifi
- Vibración

- Geolocalización
- Cámara

Pero en la actualidad, el número de navegadores (browsers) que implementan estas API's es muy reducido. Tal es así, que la única API implementada en casi todos los navegadores y que se encuentra en un estado lo suficientemente madura para ser utilizada es la API de geolocalización (GPS) mediante internet o el hardware del dispositivo.

Según las fuentes citadas en el artículo referenciado en [HTML5-mobile-problems], el principal problema para la adopción de las diferentes API's tiene que ver con las políticas de las principales empresas desarrolladoras de navegadores, que a su vez, también son proveedores de sistemas operativos y están muy preocupadas por "canalizar" las aplicaciones a las respectivas tiendas. De este modo, Google alienta las aplicaciones nativas Chrome o Android y Apple hace lo propio y, aunque parece preocupada en la implementación de últimos estándares HTML5, está "*dejando de lado las API's relacionadas con el rendimiento, por ejemplo WebGL*".

También se menciona en [HTML5-mobile-problems] que si bien la diferencia entre la implementación de las API's por HTML5 no afecta el rendimiento con respecto a implementaciones nativas, su utilización en HTML5 también se ha vuelvo más compleja ya que no existen herramientas adecuadas para "*debuggear*" y manipular el código fuente.

La Figura 38 muestra en cuadro comparativo la evolución de las diferentes API's y la cantidad de navegadores que las han adoptado.

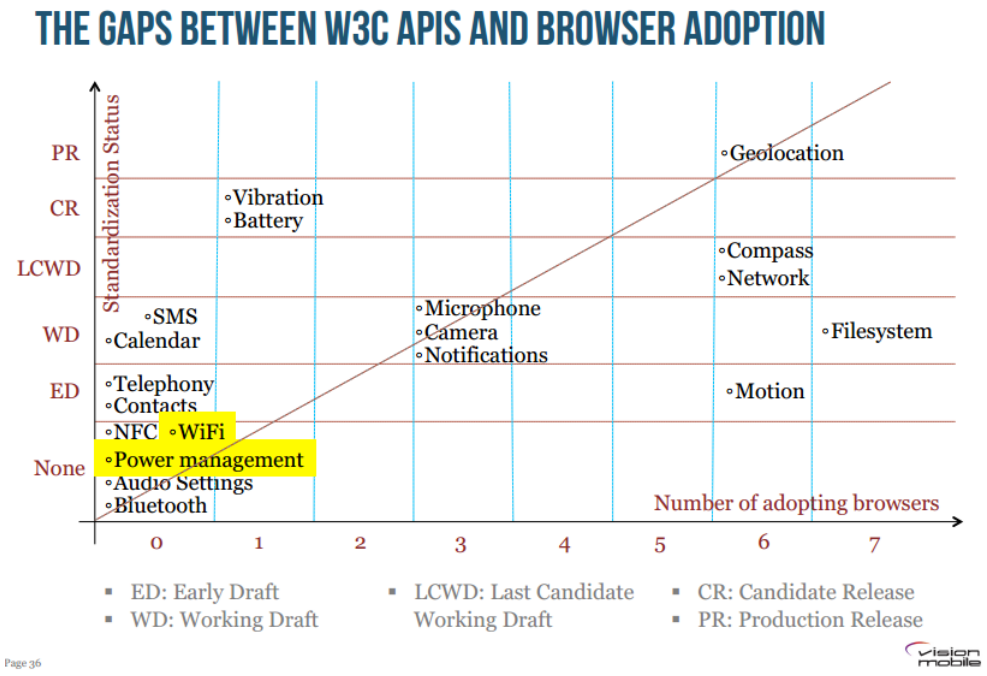


Figura 38: API's de W3C y su implementación por los browsers [HTML5-mobile-problems].

Ante esta situación planteada en la Figura 38, se volvió determinante la necesidad de contar con una aplicación nativa para obtener los datos de los sensores como temperatura, orientación, luz y dirección. En nuestro caso, la implementación fue para el sistema

operativo Android. Acorde al estado actual de HTML5, la solución propuesta en esta tesis sigue siendo la más apropiada, hasta que el mismo brinde el soporte adecuado a los sensores.

A partir de la tesis planteada se pueden proponer varios trabajos futuros. A continuación se mencionan algunos.

- En el prototipo planteado se consideraron algunos contextos provenientes de los sensores del dispositivo y solo afectaban a un usuario. Algunas extensiones respecto de este tema podrían ser considerar otros contextos, por ejemplo:

    a. poder mostrar las posiciones de otros usuarios del sistema. Esto involucra contextos que afectan a más de un usuario. Por ejemplo, visualizar la posición de mis amigos en un mapa y cuando estos cambian la posición reflejar este cambio en todos los interesados.

    b. considerar aspectos del ambiente para brindar información, por ejemplo, servicios contextuales del ambiente.

    c. considerar aspectos del dispositivo, por ejemplo, que dependiendo de la batería del usuario se reduzcan los servicios brindados.

    d. si bien los historiales del usuario son utilizados para guardar sus actividades, estos se podría utilizar como información contextual para enriquecer los servicios de la aplicación web. Por ejemplo, acomodar la interfaz acode a sus accesos frecuentes.

    e. las preferencias del usuario podría ser otro contexto a incorporar, por ejemplo, para las búsquedas de caminos. Estas preferencias podrían llegar a variar, desde el camino más corto, más rápido, etc.

    Estas extensiones son posibles con la arquitectura propuesta ya que usarían la capa de servicios que está en el servidor para mandar estos datos, y los mismos se propagarían a los usuarios interesados.

- Otro trabajo futuro seria probar el funcionamiento del prototipo con más usuarios, para poder detectar el funcionamiento y la eficiencia del servidor respecto al tiempo de respuesta entre que recibe un cambio y lo refleja en el usuario.

- Si bien el prototipo fue desarrollado para un dominio turístico, la arquitectura general detallada en esta tesis puede ser aplicada a cualquier dominio. Acorde a esto se podría tener como trabajo futuro realizar prototipos más complejos o con dominios que requieran más procesamiento. Por ejemplo, juegos móviles basados en posicionamiento. En este caso, la información se tiene que actualizar todos al mismo tiempo, ya que en caso contrario algunos usuarios podrían ser beneficiados por estos retrasos.

- Otro trabajo futuro para brindar soporte a aplicaciones más complejas podría ser implementar algún mecanismo para la sincronización de varios contextos. Al tener aplicaciones más complejas hay que analizar cómo trabajar con varios contextos al mismo tiempo y brindar una respuesta cuando la misma depende de varios contextos al mismo tiempo. Se podría investigar soluciones como la propuesta en [Fortier et al., 2010].

Bibliografía

[Albaladejo Da Silva, 2011] Albaladejo Da Silva, J.M.: Adaptación de un Smart-Phone para la toma de datos. Universidad Politécnica de Cartagena. 2011

[Alonso et al., 2004] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications – Capítulo 4. 2004.

[Android] http://source.android.com/

[Aplicaciones Móviles] http://www.alertaenlinea.gov/articulos/s0018-aplicaciones-m%C3%B3viles-qu%C3%A9-son-y-c%C3%B3mo-funcionan

[Dey,2000] Dey, A. K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, Georgia Institute of Technology, 2000. Director: G. D. Abowd

[Fortier et al., 2010] Fortier, Andrés, Gustavo Rossi, Silvia E. Gordillo, and Cecilia Challiol. Dealing with variability in context-aware mobile software. *Journal of Systems and Software* 83, no. 6 (2010): 915-936.

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

[Espada et al., 2012] Espada J. P., Martínez O. S., Pelayo G-Bustelo B. C., Cueva Lovelle J. M., Ordoñez de Pablos P, Journal of Universal Computer Science, vol. 18, 2012.

[Espada et al., 2012b] Espada J. P., González Crespo P., Martínez O. S., Pelayo G-Bustelo B. C., Cueva Lovelle J. M., Extensible architecture for context-aware mobile web, applications, Department of Computer Science, University of Oviedo, 2012.

[Flamingo] http://blog.exadel.com/flamingo-for-android/

[Gossweiler et al., 2011] Gossweiler, R., McDonough, C., Lin, J., Want, R.: Argos: Building a Web-Centric Application Platform on Top of Android. Pervasive Computing, IEEE, 10(4), pp 10-14, 2011.

[Gu et al., 2004] Tao Gu, Hung Keng Pung, Da Qing Zhang: A middleware for building context-aware mobile services. Department of computer science, National University of Singapore. 2004.

[Hessian] http://hessian.caucho.com/

[Hessian-Comparación] http://javaaa.wordpress.com/tag/hessian/

[HTML5] http://www.w3schools.com/html/html5_intro.asp

[HTML5Spec] http://www.w3.org/2009/dap/

[HTML5-mobile-problems] http://www.infoq.com/news/2013/11/mobile-html5

[JEE] http://www.oracle.com/technetwork/java/javaee/overview/index.html

[JSF] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[Maven] http://maven.apache.org/

[Parsons, 1989] Parsons J. D., Gardiner J. G.: Mobile Communication Systems. New York, NY, USA. 1989

[Primefaces] http://www.primefaces.org/whyprimefaces

[Primefaces-Concepto] http://infociberland.comxa.com/primefaces/

[Push] http://www.w3.org/TR/push-api/

[Schilit, 1994] Schilit, B.: A System Architecture for Context-Aware Mobile Computing. PhD thesis, Columbia University, 1994.

[Spring] http://spring.io

[Stüber, 2012] Stüber, Gordon L: Principles of Mobile Communication. Georgia Institute of Technology. 2012 3rd ed.

[Tomcat] http://tomcat.apache.org/