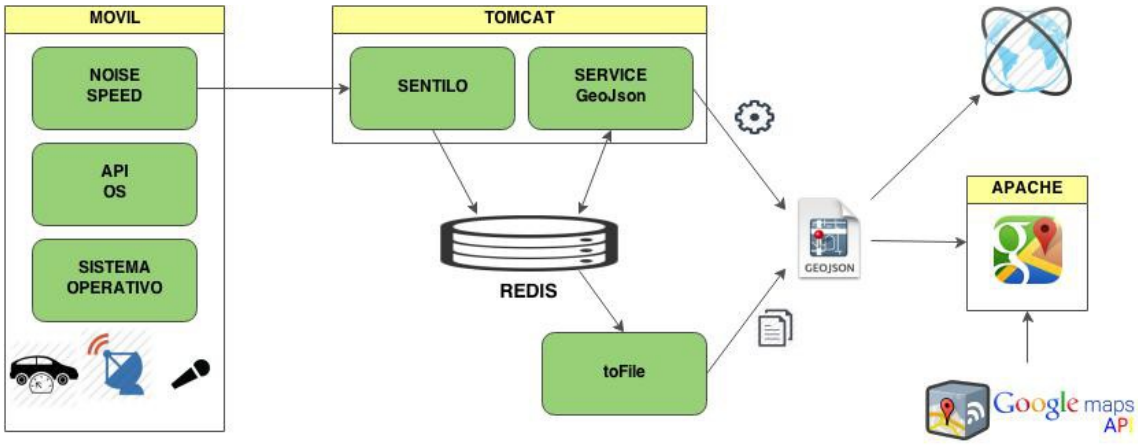
UNIVERSITAT OBERTA DE CATALUNYA
PROYECTO FIN DE MASTER
SOFTWARE LIBRE

LA CIUDAD EN EL CONTEXTO DE LA INTERNET DE LOS OBJETOS



Fecha de entrega: 9 de Junio de 2014
Autor: Esteban Román Castellanos
Tutor de la UOC: Carles Pairot Gavaldà
Tutor del Ayuntamiento: José Ignacio González Varona
Área de especialización:
Sistemas Distribuidos y Desarrollo de Aplicaciones Web.

ÍNDICE DE CONTENIDOS

1 INTRODUCCIÓN

El siguiente trabajo responde al Proyecto de finalización de los estudios de Máster en Software Libre de la Universitat Oberta de Catalunya. Se enmarca en las áreas de especialización de Sistemas Distribuidos y Desarrollo de Aplicaciones Web y su denominación es:

"La ciudad en el contexto de la Internet de los objetos"

El Proyecto Final de Master (PFM) tiene una orientación profesional y el responsable es el *Institut Municipal d'Informatica del Ajuntament de Barcelona* conjuntamente con la *Universitat Oberta de Catalunya*. Dicho PFM forma parte del desarrollo de la **SmartCity**[1][1] de la Ciudad de Barcelona cuyo núcleo de control es el **City OS**[2] al que llega información desde diferentes elementos como la **PSAB**[3], plataformas de scadas, cámaras, etc. De él se extrae información hacia la SmartCity compuesta de Situation Room[4], Aplicaciones Web, app's, etc. En la figura 1 se muestra la interrelación entre los elementos citados:

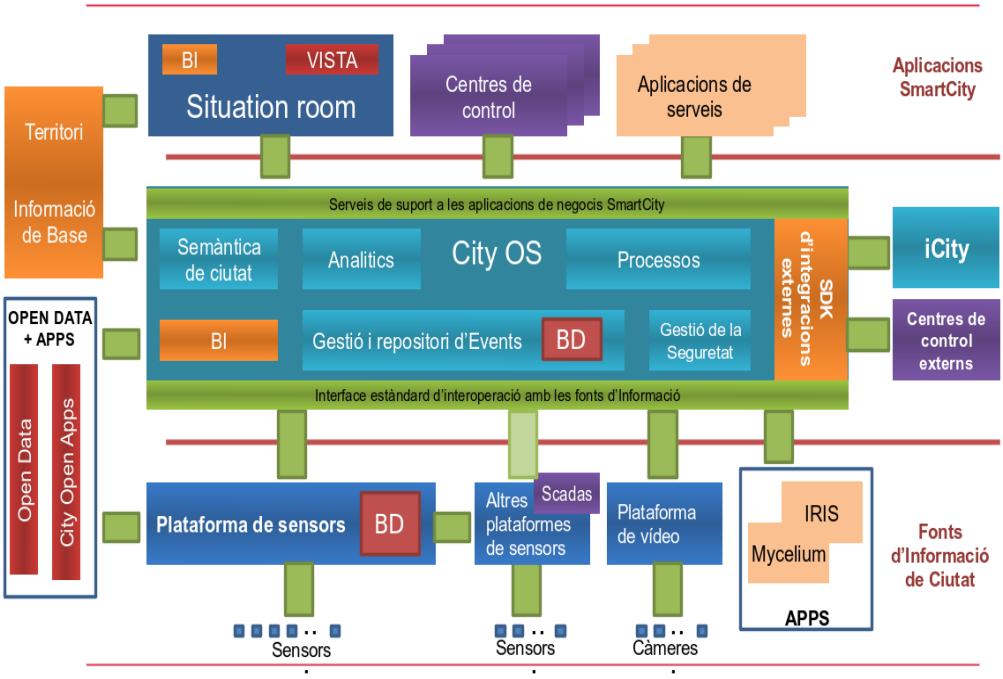


Ilustración 1: Arquitectura del City OS

1 Ciudades con TIC como soporte y herramienta para la provisión de servicios sostenibles.
2 The City Opetating System. http://www.urbiotica.com empresa colaboradora con el BCN
3 Plataforma de Recogida de datos de Sensores y Actuadores de Barcelona: http://connecta.bcn.cat
4 Sala de emergencias o de control del ayuntamiento.

El proyecto de la Smart City está íntimamente relacionado con “La Internet de las cosas”[2] que describe las técnicas bajo las cuales los sensores alojados en todo tipo de dispositivos pueden comunicarse con algunos procesos del mundo real para ofrecer servicios que mejoren la habitabilidad y sostenibilidad de las ciudades.

Los **objetivos** del proyecto son el análisis, diseño e implementación de un sistema completo de recogida de observaciones procedentes de dispositivos dotados con sensores (como móviles), su publicación en forma de texto y la posterior generación de mapas en tiempo real con los diferentes valores observados. Dentro de los requisitos iniciales no está determinada la magnitud física a enviar, sino que se considera de forma genérica para cualquier parámetro observable. Los parámetros elegidos como ejemplo son las medidas de ruido ambiental y la velocidad de desplazamiento del dispositivo que ilustrarán cómo el sistema se puede adaptar a las peculiaridades del parámetro medido. El sistema tendrá cuatro elementos principales:

- Aplicaciones que se ejecutarán sobre dispositivos móviles o fijos encargadas de medir, procesar y enviar las observaciones a una plataforma.
- Plataforma encargada de recibir y contener de forma eficiente y segura las observaciones.
- Proceso de publicación en tiempo real y través de Internet de toda la información requerida por un usuario en forma de texto.
- Proceso de publicación de datos históricos.
- Mapas Web de publicación de las observaciones en tiempo real e históricos.

Gracias a este sistema los ciudadanos de Barcelona y los técnicos del Ayuntamiento tendrán accesible dicha información en tiempo real y diferido para poder actuar en consecuencia.

Las áreas de actuación del proyecto serán:

- **Administración informática**: será necesario administrar y mantener la comunicación entre los distintos componentes del sistema e instalar y mantener diversos servicios sobre un servidor.
- **Desarrollo de aplicaciones**: hay que diseñar y desarrollar la aplicación para los distintos dispositivos de toma de datos. También desarrollaremos una web sobre la que se mostrarán los mapas web.
- **Seguridad informática**: el servidor estará expuesto a Internet por lo que habrá que tomar las medidas necesarias para su seguridad.
- **Marketing**: habrá que publicitar la aplicación para que los ciudadanos puedan y quieran instalarla en sus dispositivos.

Para el desarrollo del presente proyecto vamos a utilizar la metodología ágil **Scrum**[3] que establece el **Product Backlog** en el que están incluidos todos los requisitos iniciales. El **Product Backlog** puede variar para adaptarse a nuevas circunstancias tanto del cliente como de los desarrolladores. Hemos escogido esta metodología ya que inicialmente los objetivos y recursos no están del todo definidos y pueden evolucionar a lo largo del tiempo de desarrollo.

Las dificultades que a priori se presentan son:

- La gran cantidad de dispositivos con diferente hardware y con distintos sistemas operativos. Además la oferta sigue creciendo y la cuota de mercado varía, por lo que será necesario hacer un análisis de estas cuotas de mercado para el entorno de uso de la aplicación.
- Los dispositivos en diferentes momentos tendrán activos diferentes sensores y conexiones por lo que será necesario programar sobre la app comportamientos para adaptarse a estos cambios.
- Los proveedores de localización geográfica incluidos en los dispositivos móviles son muy variados, por lo que habrá que adaptar el método usado al parámetro que queremos medir en cada momento. Habrá que tener en cuenta la precisión de la medida, el tiempo de respuesta y el consumo de batería.
- Las medidas de ruido, temperatura, etc. presentan dificultades como la calibración de los dispositivos y las buenas prácticas de medida de los usuarios.

2 ESPECIFICACIÓN DE REQUERIMIENTOS GENERALES

Inicialmente el Ayuntamiento propone una serie de requisitos que después de un acuerdo entre los tres elementos del equipo Scrum (Ayuntamiento, Universidad y equipo de desarrollo) conformarán el **Product Backlog**. Estos requisitos pueden ser clasificados en los siguientes grupos: política estratégica, métodos y procedimientos y requisitos funcionales.

Política estratégica

- Debe ofrecer una plataforma/aplicación funcional y estándar que permita el envío de parámetros desde cualquier tipo de dispositivos: móviles, tabletas, estaciones fijas, etc.
- La información obtenida estará disponible de forma universal, gratuita y en tiempo real a través de Internet.
- Todo el software usado en el proyecto será software libre. El software generado, si se licencia, también se hará bajo alguna licencia de software libre.
- En todo momento debe cumplir con la Ley Orgánica de Protección de Datos y en ningún caso se enviarán datos personales de los usuarios ni identificativos de los dispositivos.
- No tendrá limitación geográfica, las medidas se podrán enviar desde cualquier parte del mundo.
- El proyecto debe ser extensible, adaptable y de alta disponibilidad. Pudiendo adaptarse de una forma sencilla al envío de cualquier dato observable desde uno de los dispositivos citados anteriormente.

Métodos y procedimientos

- Habrá pautas de uso del sistema de envío para que las mediciones realizadas por el usuario sean lo más fidedignas posible.
- Se organizarán las copias de seguridad de las observaciones almacenadas en la plataforma y se optimizará el rendimiento de los servicios.

Requisitos funcionales

- El sistema contendrá todos los elementos necesarios: aplicaciones para el envío de diferentes observaciones, plataforma para recibir y almacenar, un proceso de extracción y publicación en formato de texto y un conjunto de mapas web que muestren todas las observaciones.
- La información debe estar siempre disponible para cualquier usuario anónimo tanto en forma de texto como en forma de mapas.
- Las aplicaciones enviarán a la plataforma observaciones procedentes de sus sensores como medidas del volumen de ruido, velocidad, etc.
- Cada observación quedará almacenada con, como mínimo, los datos de localización geográfica, marca de tiempo y la observación correspondiente.
- La aplicación podrá activarse y desactivarse de una forma sencilla.
- La plataforma será escalable y estará diseñada para la recepción simultánea de grandes cantidades de datos en un formato determinado.
- El sistema analizará la información almacenada en la plataforma y representará los niveles de cada parámetro en tiempo real sobre un mapa de acceso WEB. También representará los registros históricos de la misma forma.

3 PLANIFICACIÓN DEL PROYECTO

3.1 Metodología de desarrollo

Para la realización de este proyecto vamos a utilizar una metodología ágil. En concreto tendrá como marco de trabajo una versión reducida del método **Scrum**. Por tanto, el proceso de desarrollo estará compuesto por una serie de sprints en los que el resultado ya es potencialmente entregable, pero cada vez será más completo y detallado. Las iteraciones se realizarán de forma secuencial y se adaptarán a las entregas de las distintas partes del proyecto a la Universidad.

El principio fundamental del método Scrum es que el proyecto no está determinado desde el principio, los clientes pueden cambiar los requerimientos y además se parte de la premisa de que el problema no puede ser completamente comprendido y definido a priori. Este método se adapta perfectamente a este proyecto debido a la incertidumbre inicial sobre los requisitos a cumplir, amplitud y medidos con los que se cuenta.

Este método de trabajo tiene los siguientes roles:

- **Product Owner**: representa la voz del cliente y proporciona las historias de usuario, las prioriza y las coloca en el **Product Backlog** que es el conjunto de requisitos de alto nivel del proyecto. Este rol lo asumirá el tutor del Ayuntamiento de Barcelona.
- **Scrum Máster**: es el que decide qué requisitos de alto nivel serán incluidos en cada **sprint** y quien hace cumplir las normas. Este rol lo asumirá el tutor de la Universidad.
- **Desarrolladores**: son los responsables de hacer el diseño, desarrollo y pruebas. Este papel lo asumirá el estudiante del Máster.

En el método Scrum estándar la comunicación entre estos tres elementos está perfectamente sistematizada en varios tipos de reuniones: **Daily Scrum** (diaria), **Scrum de Scrum** (entre los representantes de cada uno de los grupos), reunión de planificación del sprint, reunión de **revisión del sprint** y la **retrospectiva** del sprint.

Además de los roles y formas de comunicación descritos, dicho método propone tres tipos de documentos para definir cada una de las fases del proceso:

- **Product Backlog**: ya comentado anteriormente, este documento es genérico y contiene todos los requerimientos y funcionalidades del proyecto priorizados.
- **Sprint Backlog**: es un documento detallado que describe los requerimientos que serán atendidos en cada sprint y cómo se van a implementar. Se describen las tareas a realizar y el número de horas de cada una.
- **Burndown Chart**: este documento muestra gráficamente el número de requerimientos del Product Backlog que aún están pendientes en cada inicio del sprint.

Como se puede ver en la ilustración 3, el método de trabajo siempre comienza seleccionando los requisitos del Product Backlog y generando el Sprint Backlog, el cual contiene las tareas a cumplir en esa iteración (sprint).

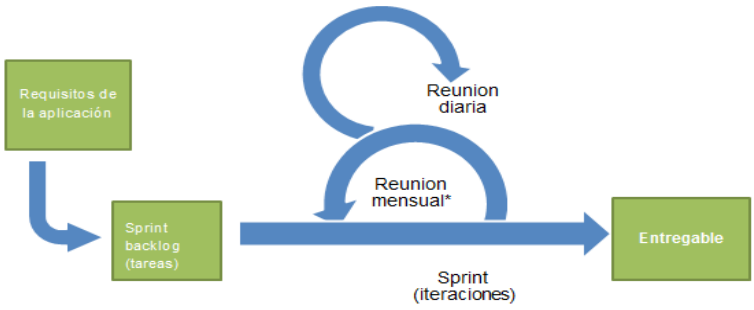


Ilustración 2: Flujo de trabajo de Scrum

En este proyecto se va a simplificar el sistema de reuniones dejando solo establecidas las reuniones de planificación del sprint, la revisión y retrospectiva, y se realizarán mediante correo electrónico o por teléfono.

3.2 Tareas y subtareas

El proyecto estará compuesto por la definición inicial y tres sprint que vienen determinados por la entrega de cada una de las Pruebas de Evaluación Continua (PEC). En esta definición se determinó el **Product Backlog** inicial con todas las historias de usuario que determinarán las funcionalidades específicas que debe tener la solución. En cada uno de estos sprint el nivel de definición y funcionalidad de la entrega irán en aumento. Las fechas de las entregas son las siguientes:

(1) 01/10/2013-12/10/2013: Definición del proyecto, Product Backlog y entrega de la PEC1.
(2) 13/10/2013-26/12/2013: Entrega de la PEC2 y sprint1.
(3) 27/12/2013-20/03/2014: Entrega de la PEC3 y sprint2.
(4) 21/03/2014-09/06/2014: Entrega de la PEC4 y sprint3.
(5) 17/06/2014-31/06/2014: Defensa del proyecto.

Utilizando el método Scrum el cronograma del proyecto puede variar significativamente ya que el conjunto de tareas a desarrollar se decide en el inicio de cada uno de los sprint. Inicialmente se plantearon cuatro grandes grupos de tareas: viabilidad del proyecto, estado del arte y diseño, desarrollo e implantación. Cada una de estas grandes tareas se subdividen a su vez en subtareas más pequeñas que se reparten entre los tres sprint del proyecto.

En la ilustración 3 se puede ver el desglose de tareas y subtareas con la carga horaria de cada una de ellas. Este diagrama ha sido diseñado en el **inicio** del proyecto y como se verá a continuación será modificado sustancialmente a lo largo de su desarrollo.

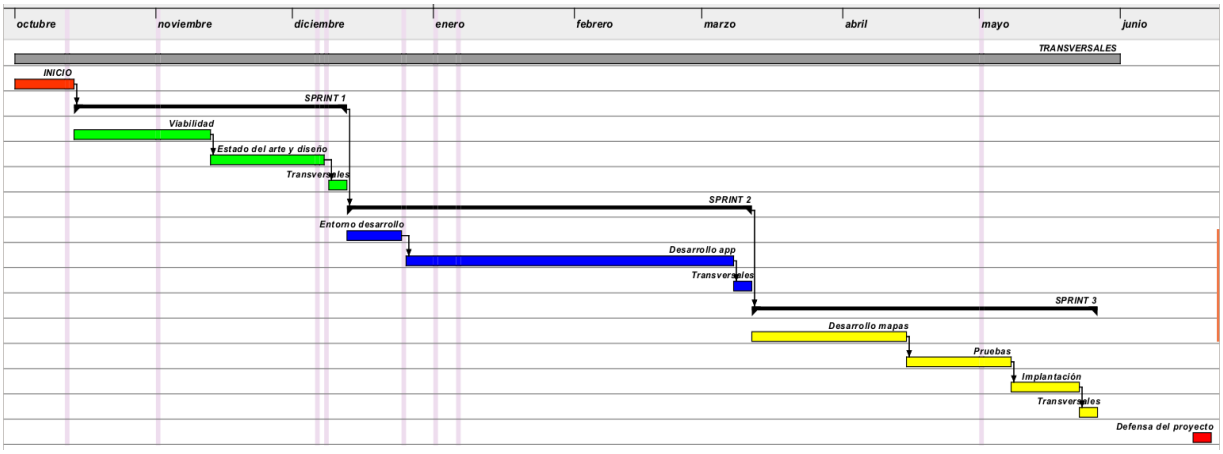


Ilustración 3: Cronograma de los sprint

Al finalizar cada fase se entregará una memoria con todos los pormenores del proyecto incluyendo todos los detalles técnicos de la solución propuesta y las relaciones entre los distintos elementos. La unión de los tres trabajos anteriores es la que da pie a esta memoria final.

Cuando se implemente la solución también se entregará todo el código fuente, toda la documentación necesaria para su instalación y toda la documentación para el acceso web a los mapas.

Para atender a todos los requerimientos del proyecto habrá que llevar a cabo una serie de tareas y subtareas. El orden de realización lo decidirá el **Scrum Máster** que atendiendo a la prioridad del cliente generará el Sprint Backlog en el inicio de cada iteración. Cada tarea tiene una estimación del tiempo aproximado necesario para desempeñarla. Finalmente las tareas definitivas realizadas durante todo el proyecto son las siguientes:

Tareas transversales

Una de las tareas realizadas durante todo el proyecto serán las reuniones “ON-Line” Scrum. En el inicio del Sprint se efectuará la reunión de planificación en la que se determinará el Sprint Backlog, al finalizar se procederá a la revisión y retrospectiva del Sprint.

En la revisión se tiene que validar si la solución propuesta o desarrollada cumple con los requisitos iniciales y tiene la conformidad del cliente. En la retrospectiva hay que describir los problemas y soluciones que ha habido durante el Sprint además de los acuerdos a los que se ha llegado.

Además de estas reuniones se elaborará la memoria del desarrollo del Sprint que contendrá la validación de los requisitos y el desarrollo del Burndown Chart que especifica los requisitos que quedan por atender, además de toda la documentación necesaria.

De forma transversal también estará la adaptación del código desarrollado a las nuevas circunstancias como el montaje de la plataforma y la representación en forma de mapas, de forma que en cada entrega el producto sea mejor.

Tarea 1: Viabilidad

Antes de acometer el proyecto tenemos que realizar un estudio de viabilidad para extraer información sobre el estado del arte y el alcance real del objetivo propuesto. Con este estudio tendremos la información necesaria para llegar a acuerdos entre los integrantes del equipo Scrum: Universidad, Ayuntamiento y alumno.

Dicho estudio de viabilidad incluirá las siguientes tareas priorizadas:

(1) Establecimiento de los requisitos y necesidades del cliente (P1).

(2) Análisis de la situación actual de las aplicaciones móviles, los servicios y la plataforma de publicación web del Ayuntamiento (P1).

(3) Estudio del estado del arte de todas las técnicas a utilizar (P1).

(4) Estudio, valoración y elección de las alternativas de solución (P1).

Tarea 2: Análisis y diseño de las aplicaciones móviles de envío de observaciones

(1) Análisis de las distintas plataformas y sistemas operativos sobre los que desarrollar app's y su cuota de mercado (P1).
(2) Documentación sobre las formas y tipos de medidas de parámetros como el ruido ambiental o la velocidad de desplazamiento.
(3) Análisis de las técnicas utilizadas en la actualidad para la localización geográfica con dispositivos móviles (P1).
(4) Análisis de las interfaces de la aplicación (P1).
(5) Análisis de la API del sistema operativo sobre el que se desarrollará (P1).
(6) Diseño de la arquitectura y de los subsistemas de las aplicaciones (P1).

Tarea 3: Entorno de desarrollo (no forman parte de las historias de usuario**)**

(1) Análisis y estudio de los tipos de entornos de desarrollo sobre Android.
(2) Configuración del entorno de desarrollo sobre Android (P1).
(3) Generación de los emuladores de terminal (P1).
(4) Configuración del repositorio de desarrollo sobre ***Git*** (P1).
(5) Análisis y estudio de los entornos de desarrollo para la programación de servicios web sobre ***Tomcat*** y su empaquetado en formato war.
(6) Configuración de Eclipse para el desarrollo y empaquetado Android, uso del repositorio, desarrollo y empaquetado de servicios web, etc. (P1).

Tarea 4: Desarrollo de las app's y pruebas unitarias

(1) Desarrollo, instalación y pruebas unitarias del envío automático de información. Tanto de velocidad como de medidas de ruido (P2).
(2) Pruebas de integración entre los distintos módulos y resolución de problemas (P2).
(3) Desarrollo, instalación y pruebas unitarias de la visualización del nivel de ruido y velocidad medidos por el dispositivo (P2).
(4) Posterior a la instalación de la plataforma habrá que adaptar la app y realizar las pruebas de integración.

Tarea 5: Instalación de la plataforma de recogida de datos

(1) Análisis y estudio de las características, funcionalidades y servicios de la herramienta Sentilo.
(2) Estudio de la forma de instalación y configuración.
(3) Contratación y configuración de un servidor virtual para el montaje de dicha plataforma y montaje de los servicios de soporte del proyecto: Apache2, ssh, bind, etc.
(4) Instalación y comprobación del funcionamiento de Tomcat.
(5) Estudio, instalación y configuración de Redis, MySQL y MongoDB. Servidores sobre los que funciona Sentilo.

(6) Instalación y configuración de Maven2.
(7) Instalación y configuración de la herramienta propiamente dicha. Este proceso ha llevado más tiempo del que inicialmente se pudiera pensar debido a ciertos errores en la documentación del software Sentilo que fueron subsanados por el equipo de soporte una vez advertidos.
(8) Generación del proveedor, componente y sensores a utilizar durante el proyecto dentro de la herramienta Sentilo.
(9) Test de instalación, pruebas de integración con las app's y resolución de problemas.

Tarea 6: Publicación de la información

(1) Análisis de las alternativas para la realización de mapas Web que puedan representar cualquier tipo de parámetro con información de localización geográfica (P3).
(2) Análisis de la forma de implementar los mapas en el Ayuntamiento (P3).
(3) Elección de la plataforma de publicación y análisis de la API de la plataforma de publicación elegida (P3).
(4) Diseño e implementación de la API (servicio web) de extracción de la información para los mapas en tiempo real (P3).
(5) Diseño e implementación del servicio de generación de ficheros para la publicación de los mapas históricos (P5).
(6) Diseño, desarrollo e implantación de la web del proyecto (P5).

Tarea 7: Implantación y Entregas

(1) Pruebas de integración, resolución de problemas y mejora del producto.
(2) Empaquetado y distribución controlada de la aplicación (P6).
(3) Recogida de los feedback, resolución de problemas y mejora del producto (P6).
(4) Distribución de la aplicación y aceptación por parte del cliente (P6).

La duración máxima del proyecto es de 240 días a lo largo de 8 meses con una carga horaria teórica de unas 370 horas, aunque en la realidad sobrepasa holgadamente las 400 horas. La siguiente imagen muestra un diagrama de Gantt real del proyecto que incluye todas las tareas realizadas y el tiempo empleado en número de días.

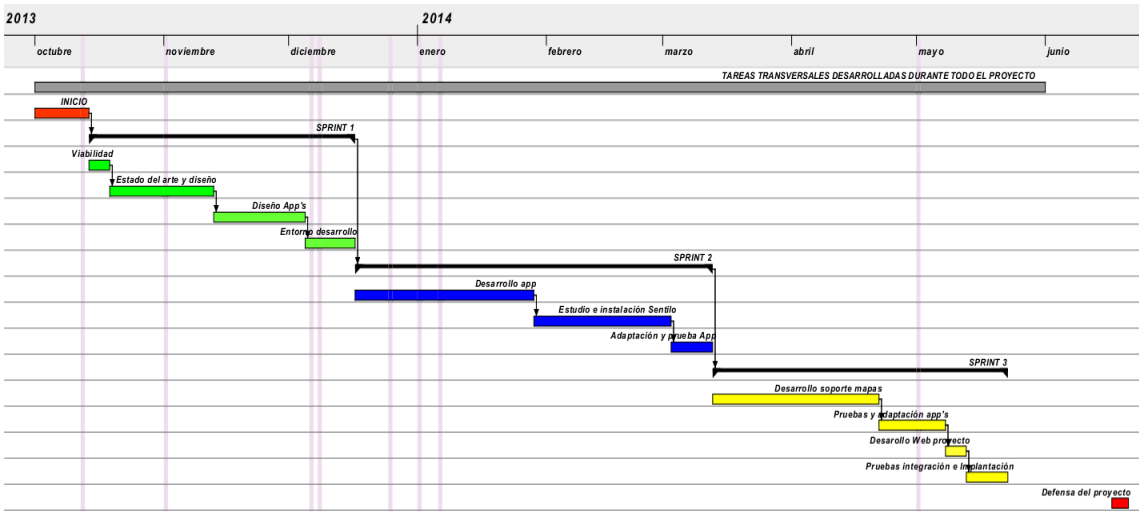


Ilustración 4: Diagrama de Gantt final

3.3 Recursos y coste del proyecto

Todo el software utilizado para la realización del proyecto será Open Source al igual que todos los productos obtenidos. Para las pruebas se recurrirá a emuladores de móviles y varios SmartPhone con una línea de datos. Como equipo de trabajo se contará con un ordenador portátil, una línea ADSL y un servidor virtual contratado.

Los elementos que incluyen costos en este proyecto son:

- Recursos humanos: una hora a la semana por cada uno de los tutores (Ingeniero Superior en Informática) y 415 horas en total del estudiante.
- Recursos software: el software utilizado es exclusivamente software libre por lo que el gasto en licencias es nulo.
- Líneas de comunicación en las que se incluirían las ADSL y las líneas de datos contratadas para realizar las pruebas con los móviles.
- Servidor virtual sobre el que se montará la plataforma y todos los servicios necesarios como: DNS, servidor web, contenedores de aplicaciones, bases de datos, etc.

La hora de un Ingeniero Superior en Informática tiene una retribución alrededor de 60€ y la hora de un Técnico en Informática alrededor de 35€. Hemos considerado diez meses de trabajo ya que el proyecto comenzó en Octubre y terminará en Junio. Y vamos a suponer una hora de trabajo a la semana para cada Ingeniero Superior.

En la imagen 6 se puede ver el presupuesto desglosado:

HARDWARE:	CANT	PRECIO UNI (€)	TOTAL	DESCRIPCIÓN
EQUIPOS SOBREMESA	1	800	800	Portátil
SERVIDOR VIRTUAL	12	7	84	VPS para soportar la plataforma y el repositorio
MOVILES DE PRUEBAS	2	300	600	Móviles de varias plataformas
	SUB TOTAL HARDWARE		1484	
OTROS	CANT	PRECIO UNI (€)	TOTAL	DESCRIPCIÓN
LINEAS DATOS MOVILES	1	100	100	Lineas de datos para las pruebas durante 3 meses
LINEAS ADSL	3	400	1200	Lineas ADSL por participante y para 10 meses
	SUB TOTAL OTROS		1300	
PERSONAL	CANT	PRECIO UNI (€)	TOTAL	DESCRIPCIÓN
TUTOR UNIVERSIDAD	50	60	3000	Horas de asesoramiento y corrección de trabajos
TUTOR AYUNTAMIENTO	50	60	3000	Horas de asesoramiento
ESTUDIANTE	415	35	14525	Desarrollo de las actividades y software
	SUB TOTAL PERSONAL		20525	
	TOTAL		23309	

Ilustración 5: Presupuesto del proyecto

La carga horaria de cada subtarea la podemos ver en la siguiente imagen:

		DESCRIPCIÓN	Historias De cliente	tareas Técnicas		
TAREA 0	0.1	Requisitos iniciales y definición del proyecto	16			
	0.2	Tareas transversales: redacción de entregas, reuniones, etc	33			
					TOT	49
TAREA 1 Viabilidad	1.1	Análisis situación actual del ayuntamiento	16			
	1.2	Sistemas operativos para APP	6			
	1.3	Medidas magnitudes físicas	6			
	1.4	Localización geográfica	12			
	1.5	Servicios web	8			
	1.6	Mapas web	8			
	1.7	Propuesta de la solución	4			
					TOT	60
TAREA 2 Diseño APP	2.1	Estudio API Android	6			
	2.2	Medidas magnitudes físicas	6			
	2.3	Localización geográfica	12			
	2.4	Interfaces	2			
	2.6	Arquitectura	6			
					TOT	32
TAREA 3 Entorno Desarrollo	3.1	Análisis de entornos desarrollo Android		16		
	3.2	Configuración del entorno SDK		8		
	3.3	Generación de Emuladores		2		
	3.4	Repositorios Git		6		
	3.5	Configuración de Eclipse: Android, S. web, Git, etc		6		
					TOT	38
TAREA 4 Desarrollo App's	4.1	Localización geográfica	20			
	4.2	Interfaz de usuario	14			
	4.3	Medidas ruido				
	4.4	Proceso calculo				
	4.5	Formateo en json	10			
	4.6	Envío por la conexión activa	10			
	4.7	Pruebas unitarias y resolución de problemas		16		
					TOT	70
TAREA 5 Sentilo	6.1	Análisis y estudio de Sentilo y servidores.	24			
	6.2	Contratación y configuración del servidor virtual	8			
	6.3	Estudio e instalación: Tomcat, Redis, MongoDB, etc	18			
	6.4	Instalación de Sentilo: Sentilo	12			
	6.5	Configuración y pruebas: Sentilo	6			
					TOT	68
TAREA 6 Mapas Web	7.1	Análisis de la API de JavaScript de Google	8			
	7.2	Diseño e implementación del servicio web	24			
	7.3	Diseño e implementación de ficheros históricos	16			
	7.4	Realización de la web y mapas del proyecto	34			
					TOT	82
TAREA 7 Integración	8.1	Pruebas de integración y resolución de problemas	16			
	8.2	Empaquetado y distribución controlado	10			
	8.3	Feedback, adaptaciones y mejora del producto	30			
					TOT	56

TOTAL PROYECTO CLIENTE	401
TOTAL PROYECTO (cliente y horas técnicas)	455

Ilustración 6: Carga horaria por tarea y subtarea

Barcelona impulsa junto con otras ciudades[5] la definición e implementación de servicios ofrecidos a los ciudadanos basados en la SmartCity. Este desarrollo se apoya sobre una plataforma tecnológica capaz de tratar la información procedente de una gran cantidad de dispositivos heterogéneos, de manera eficiente, rápida y segura. Esta plataforma es el llamado City OS.

Desde un punto de vista genérico el proyecto debe cumplir ciertos requisitos establecidos en el inicio:

- **Seguridad:** la aplicación nunca enviará información personal del usuario del dispositivo móvil y siempre se utilizarán métodos de localización basados en cliente para no revelar la posición del usuario. Por otro lado cumplirá con los estándares de seguridad establecidos por el Ayuntamiento.
- **Propiedad intelectual**: todo el software generado en el proyecto será licenciado bajo alguna licencia de software libre.
- **Distribución de la app**: la distribución será realizada de forma gratuita y estará a la disposición de cualquiera en la web del Ayuntamiento.
- **Prestaciones**: el consumo de recursos, especialmente de batería, será el mínimo posible.

Los **requisitos funcionales exactos:**

- Las aplicaciones podrán disponer de varios métodos de localización geográfica. Cuando el GPS no esté disponible y la precisión requerida por magnitud medida no sea muy alta, se utilizará la localización por Wi-Fi o en tercer lugar usando las antenas de telefonía cercanas. La adaptación a cada una de ellas será automática.
- Cuando la localización geográfica no esté disponible se detendrá el envío de datos observados.
- Enviará a la plataforma las medidas observadas de forma automática utilizando la conexión Wi-Fi en primer lugar o la línea de datos que esté disponible en ese momento. La adaptación al modo de envío también será automática.
- Como mínimo recogerá los datos de localización geográfica, observación (ruido o velocidad) y una marca de tiempo.
- El protocolo para el envío será http y el formato de los datos será Json. Se tendrá que contemplar la posibilidad del envío futuro de datos en formato XML.
- La plataforma almacenará de forma eficiente y ordenada todas las observaciones.
- Los datos de la plataforma estarán disponibles en formato de texto y en forma de mapas WEB. Estos datos serán en tiempo real e históricos.

5 http://www.redciudadesinteligentes.es/

4.1 Situación actual.

En la actualidad, el Ayuntamiento de Barcelona dispone de un sistema completo en funcionamiento constituido por varios subsistemas: Plataforma de Recogida de datos de Sensores y Actuadores de Barcelona (PSAB)[6], aplicaciones móviles que ofrecen y aportan información a la plataforma, una sala de control (Situation Room) que muestra la información de la PSAB y una Web donde se puede acceder a gráficos y mapas sobre diversos parámetros recogidos por la plataforma. Todos estos subsistemas están controlados por el CityOS

La Plataforma de Sensores y Actuadores del Ayuntamiento de Barcelona está montada utilizando la herramienta **Sentilo** que proporciona un sistema de recogida y almacenamiento de datos a través de una API de tipo REST[7][4]. Esta herramienta ha sido publicada bajo una licencia dual EUPL 1.1 y LGPL3 pocos días antes de su utilización en este proyecto.

Los elementos que componen la infraestructura son: Redis, MongoDB, MySQL y Tomcat. Para la compilación y generación de los script que la controlan se utiliza Maven. Los recursos ofrecidos por la PSAB son:

- **Sensores**: elementos hardware o software con la capacidad de generar una observación. Son los sensores contenidos en los dispositivos móviles que nos ofrecerán información como niveles de ruido, localización geográfica, etc.
- **Component**: elementos móviles o fijos que pueden contener varios sensores, estos serán los SmartPhone, tabletas, etc.
- **Proveedor**: es una entidad que representa una agrupación de componentes y que permite a estos la comunicación con la PSAB. Una de las características de esta entidad es la persona de contacto y la clave de autenticación que tendrán que tener todas las observaciones procedentes de los sensores que pertenecen a este proveedor.
- **Aplicación cliente**: entidad que permite representar los valores.

El formato (url) para identificar cada uno de los recursos existentes en la plataforma tiene como elementos: protocolo (http o https), dominio (connecta.bcn.cat), puerto, servicio usado, proveedor, sensor y parámetros.

```
protocol://domini:port/servei/event/id_proveidor/id_sensor/valor?parametre=valor
```

El envío de información a la plataforma siempre se realiza utilizando el método put en formato Json y el cliente tiene que tener permiso de escritura. Es posible utilizar dos métodos: uno abreviado en el que se incluye la información directamente en la URL y el método “normal” en el que se incluye la información en el cuerpo del mensaje con el formato siguiente :

```
{"observations":[{"value":"12.3","timestamp":"17/09/2012T12:34:45"}]}
```

6 http://connecta.bcn.cat/xwiki/bin/view/APIDocs/WebHome
7 Representational State Transfer

En el formato anterior podemos ver que hay dos parámetros: observación del sensor y timestamp. Este último no está presente en la observación enviada a la plataforma ya que es la propia plataforma quien la incluye antes de almacenarla. Además de estos dos, también podemos incluir de forma opcional la localización del dispositivo, que será necesario para nuestro proyecto.

La plataforma ofrece a los posibles usuarios los siguientes servicios:

- **Alertas:** nos da la posibilidad de publicar y recuperar alertas sobre distintos fenómenos.
- **Catalog**: permite dar de alta, actualizar, recuperar la lista y borrar componentes/sensores. Este catálogo tiene una amplia serie de componentes como: temperatura, ruido, viento, humedad, etc. También tiene una amplia serie de sensores registrados como anemómetros, temperatura, parcking, air_quality_*, etc.
- **Data**: este servicio permite publicar (enviar), recibir y borrar observaciones de un sensor o proveedor. Será el servicio que utilizará la App para el envió de medidas de ruido.
- **Subscripcio**: gracias a él los clientes de la plataforma (aplicaciones, proveedores o sensores) pueden subscribirse para recibir notificaciones cuando se producen eventos como recepción de observaciones, recepción de órdenes o alarmas.
- **Ordre**: permite enviar órdenes hacia los sensores/actuadores y recibir la respuesta.

La plataforma está provista de un sistema de autenticación y autorización incluyendo en la cabecera de las peticiones http la clave **IDENTIFY_KEY**, que como hemos comentado anteriormente pertenece al proveedor. Una vez identificado el emisor de la petición, la plataforma comprueba sobre un sistema de permisos si dicha petición se puede atender o tiene que ser rechazada.

El Ayuntamiento actualmente tiene disponibles una gran cantidad de aplicaciones para dispositivos móviles para varias plataformas: iPhone, iPad, Android, BlackBerry, Tablets Android, Symbian y Windows Phone. De entre todas ellas voy a destacar algunas de las que envían datos desde el dispositivo móvil hasta la plataforma :

- **Farmaguia**: es una aplicación que partiendo de nuestra ubicación nos da la dirección e indicaciones para llegar a la farmacia de guardia más cercana. Solo está disponible para iPhone, iPad y Android.
- **Bicing**: nos da la disponibilidad de todas las estaciones partiendo de la ubicación del usuario. Disponible para todas las plataformas menos para Windows Phone.
- **Bustia Ciudatana**: permite a los ciudadanos de Barcelona enviar incidencias de un tipo predeterminado. Solo se distribuye para Android y iPhone y iPad.

Además, en la actualidad hay varios proyectos en desarrollo: sensorización de calles, monitorización del ruido en la vía pública, telecontrol del alumbrado, etc.

4.2 Estado del arte

Este proyecto recoge una gran cantidad de aspectos a tener en cuenta sobre los que hay que documentarse de forma previa al diseño de una posible solución para los requerimientos iniciales.

Entre todos estos aspectos los más destacables son las plataformas que pueden ofrecer servicios web, los distintos sistemas operativos de los dispositivos móviles, las técnicas de localización geográfica disponibles, las técnicas de medidas de ruido y otros parámetros y las distintas formas de publicar observaciones con la localización geográfica sobre un mapa Web.

4.2.1 Servicios Web

El consorcio 3WC define los Servicios Web como sistemas de software diseñados para soportar una interacción interoperable máquina a máquina sobre la red. Suelen ser API's que ofrecen servicios a procesos alojados en equipos remotos. Hay varios estilos que son implementados actualmente:

- Arquitectura orientada a servicios SOA, donde la unidad básica es el mensaje más que la operación y que añade más capas y funcionalidades por encima del protocolo de transporte.
- Arquitectura REST, que trata de emular al protocolo HTTP en el que la interfaz de acceso al servicio está restringida a un conjunto de operaciones conocidas. REST ofrece sus servicios sin una capa añadida como es el caso de SOAP.

Entre las dos arquitecturas, la más adecuada para este proyecto es la de tipo REST ya que es más simple y requiere menos recursos por lo que se ajusta mejor a los dispositivos móviles.

Las API's de tipo REST al estar basadas en el protocolo http tienen un conjunto de **operaciones bien definidas** (get, post, put, delete, etc), una sintaxis universal para la **identificación de recursos** y la utilización de hiperenlaces para ir de un recurso a otro. Esta API nos permite crear servicios y aplicaciones que pueden ser usados por cualquier entidad que entienda html. Hay que definir los recursos o elementos de información que serán intercambiados entre las diferentes entidades del sistema e identificarlos por su URL[8]. Esta arquitectura es la que usa la herramienta Sentilo y la que tendrá el servicio de extracción de observaciones implementado en este proyecto.

Sentilo es una plataforma Open Source de recogida de datos diseñada para encajar en la arquitectura de una SmartCity. Está basada en una arquitectura de tipo REST y para su funcionamiento necesita de una serie de servidores que describimos a continuación:

- **Tomcat**[9] es un servidor web con soporte para servlet de Java y JSPs licenciado utilizando la Apache License 2.0. Para su funcionamiento necesita la máquina virtual Java. Se puede utilizar conjuntamente con Apache o de forma independiente. En la actualidad está la versión 8.0.1 (beta) y la última versión estable es la 7.0.50. Será el contenedor sobre el que se ejecutará la

8 Uniform Resource Identifier
9 http://tomcat.apache.org/

aplicación de recogida de observaciones.

- **Redis**[10] es un motor de almacenamiento clave/valor en memoria principal gracias a lo cual el rendimiento es mucho mayor en comparación con los sistemas gestores de bases de datos tradicionales. Las observaciones recogidas por la plataforma se almacenarán usando este gestor. Este sistema se puede hacer persistente de varias formas a costa de perder rendimiento. Está licenciado bajo BSD y en este proyecto utilizaremos la versión 2.8.4.
- **MongoDB**[11] es un sistema gestor de bases de datos orientado a la documentación y noSQL. Este sistema no utiliza tablas para el almacenamiento de datos, sino que los almacena en formato Json que facilita y agiliza la integración de los datos en ciertas aplicaciones. MongoDB es utilizado para almacenar toda la información referente a la plataforma web sobre la que podemos generar, borrar y modificar todos los artefactos pertenecientes a Sentilo. Está licenciado bajo GNU AGPL V3.0.
- **MySQL** es un sistema gestor de bases de datos relacional y SQL. Actualmente pertenece a Oracle, que lo distribuye con una licencia dual: GPL GNU cuando no se va a incluir en un software propietario y cuando sí va a ser incluido se tendrá que pagar una licencia para este uso. Todas las observaciones se almacenan en Redis, pero existe un agente software que cuando está activo envía estas observaciones también a MySQL. Por tanto este sistema gestor no es una parte fundamental de la plataforma pero nos podría ofrecer un sistema de almacenamiento persistente.

Para la instalación de Sentilo se utiliza **Maven** que es un Framework para la construcción y gestión de proyectos Java. Puede compilar, ejecutar test, generar ficheros .jar, instalar, desplegar, generar zip para instalación, etc. Al igual que Tomcat, pertenece a la Apache Software Foundation y está licenciado bajo la Licencia Apache. En este proyecto se utilizará la versión 2.2.

4.2.2 Sistemas operativos para dispositivos móviles

En la actualidad hay una gran cantidad de posibles plataformas sobre las que podemos programar aplicaciones para dispositivos móviles, inicialmente podemos clasificarlas en programación nativa o programación híbrida. En el mercado hay algunos framework que nos permiten programar en HTML5/JavaScript/CSS3 y después compilarlo online para la plataforma que deseemos como es el caso de PhoneGap. Otro es Velmobc, que con su framework, nos permite desarrollar utilizando XML para después ejecutarlo sobre cualquier plataforma de forma “nativa”.

Si queremos implementar la aplicación de forma nativa, una de las primeras cuestiones que nos tenemos que plantear es la elección de la plataforma y por tanto del lenguaje de programación con el que implementaremos la APP. En el momento de realizar este proyecto estas son las plataformas más representativas:

10 http://Redis.io/
11 http://www.mongodb.org/

- **Android**: es un sistema operativo para dispositivos móviles desarrollado por Android Inc, posteriormente esta empresa fue comprada por Google a quien pertenece en estos momentos. Está desarrollado bajo la licencia Apache 2.0 y GPL2[12]. Las aplicaciones se programan habitualmente en Java con el Android SDK[13], pero también hay otras alternativas como C, C++, Basic4Android, Mono para Android, App Inventor, LiveCode, In Design CS6, etc. En la actualidad esta es la plataforma más extendida en el mundo y en especial en España.
- **iOS**: es el sistema operativo de la empresa Apple Inc inicialmente diseñado para Iphone pero en la actualidad también corre sobre iPod Touch, iPad y el Appel TV. Está licenciado bajo APSL y Apple EULA. Tiene el kit de desarrollo SDK, el iPhone simulator y el lenguaje de programación que se utiliza es Objective C.
- **Windows Phone**: ha sido desarrollado por Microsoft como sucesor de Windows mobile. Las aplicaciones se programan con .NET y está licenciado bajo Microsoft CLUF.
- **BlackBerry OS**: este sistema operativo fue desarrollado por la empresa canadiense de teléfonos móviles BlackBerry. Para desarrollar aplicaciones tenemos que utilizar la BlackBerry Java Development Environment o un Pluging para Eclipse. El lenguaje de programación es Java.
- **Firefox OS**: está desarrollado por Mozilla Corporation con la colaboración de otras empresas como Telefónica bajo licencia MPL[14]. Tiene un Kernel Linux sobre el que podemos programar aplicaciones utilizando HTML5.

4.2.3 Técnicas de localización geográfica

La localización geográfica es un proceso mediante el cual se obtiene la latitud y longitud de la posición de un dispositivo. En el caso de los dispositivos móviles podemos tener dos formas genéricas de obtenerla: GPS y LBS.

Satélites (GPS): es la forma tradicional para calcular las coordenadas de localización geográfica de un punto (longitud, latitud y altura). Utiliza 32 satélites equidistantes y antes de poder dar la posición, el sensor del dispositivo necesita localizar a cuatro de ellos como mínimo. El tiempo requerido en este proceso es el TTFF[15] y dependiendo de las condiciones puede ser bastante largo (hasta 15 minutos), consume una gran cantidad de energía y en recintos cerrados o zonas con edificios muy altos la recepción de señal es muy limitada o nula, por lo que funciona mejor en espacios abiertos. La precisión, según Bertoli [5], es muy alta entre 10m y 1m dependiendo de las circunstancias. La tecnología A-GPS[16] disminuye el TTFF usando una red de dispositivos fijos de referencia que provee a los dispositivos GPS clientes la posición aproximada, posición de los satélites, efemérides, etc[6].

12 Licencia Pública General GNU: http://www.gnu.org/licenses/gpl-2.0.html
13 Kit de desarrollo de software.
14 Licencia Pública de Mozilla: http://www.mozilla.org/MPL/
15 Time To First Fix
16 Servicio de posicionamiento global asistido

Los servicios de localización **LBS**[17] ofrecen de forma personalizada y en tiempo real la situación geográfica de dispositivos móviles sin la utilización del GPS. Este servicio puede ser ofrecido por los proveedores de telefonía o por el software del propio móvil utilizando un proceso de cálculo llamado trilateración inversa. El proceso de posicionamiento puede utilizar dos elementos[5]:

- **Estaciones WI-FI**: este método utiliza los puntos de acceso WI-FI que se pueden reconocer alrededor de la posición móvil. Como elemento de identificación de la WI-FI se utiliza la dirección MAC del PA. Es un método bastante preciso ya que el alcance de estas redes es de unos 60m [5], además el TTFF es muy pequeño y el consumo de batería también es menor. Hay técnicas pasivas en las que es el punto de acceso el que calcula la posición, y activas en las que es el propio cliente el que lo calcula. Tres de estas son: Sensores de proximidad (posición del PA=móvil), trilateración de las estaciones cercanas y Fingerprint, donde se utiliza un mapa del nivel de ponencia de cada una de ellas.

  En todos los casos es necesario disponer de un mapa de localización de las distintas redes WI-FI que ha sido realizado por grandes empresas como Google, Apple y Microsoft. Cuando autorizamos el servicio de ubicación anónimo de Google (por ejemplo) estamos permitiendo que cuando nuestro teléfono tenga una localización geográfica envíe la MAC de los puntos Wi-Fi detectados a su alrededor. De esta forma se tiene una base de datos de la ubicación aproximada de una gran cantidad de puntos de acceso. Para utilizar este servicio no hace falta estar asociado a ninguna red Wi-Fi, basta con recibir la señal y enviar una consulta sobre la base de datos para conocer la localización aproximada.
- **Red de telefonía móvil**: Para esta técnica se utiliza el "**cell ID**" de las antenas de telefonía más cercanas al dispositivo. En la actualidad existen dos redes celulares: GSM (2G) y UMTS (3G) y existe una cuarta generación que ya se está implantando LTE (4G). En GSM existen tres técnicas de localización [5]: basadas en terminal móvil (MB), basadas en la red móvil (NB) e híbridas. Para este proyecto la más adecuada sería la MB ya que es el propio móvil el que calcula la posición y por tanto la red no tiene esta información, no depende de las capacidades que nos ofrece la red y se pueden hacer tantas medidas como se deseen. Usando esta técnica, y en circunstancias muy favorables, la precisión podría llegar a estar entre 50 y 150m según [5]. Solo en zonas sin cobertura no estará disponible. Esta forma y la anterior están incluidas en la llamada Network Location Provider de Android[18].

Para determinar el método a seguir tenemos que tener en cuenta aspectos como la **precisión**, la **velocidad del proceso de localización** y el **consumo** de batería. La mejor forma de optimizar el resultado es la combinación de varias técnicas adaptándolas a los requerimientos de cada situación y propiedades de la magnitud física que queremos enviar.

17 Location Based Services
18 http://developer.android.com/guide/topics/location/strategies.html

4.2.4 Medición de magnitudes físicas

El objetivo de este proyecto es generar un sistema de envío, tratamiento y publicación de cualquier parámetro que sea susceptible de ser medido por un dispositivo móvil o fijo. Por este motivo, los parámetros escogidos carecen de importancia. Uno de estos posibles parámetros podría ser el ruido ambiental o contaminación acústica medida por el micrófono del dispositivo.

La definición de contaminación acústica y los procedimientos de medida están regulados por normativas europeas, españolas y catalanas. Hay tres normas legislativas sobre el ruido a nivel europeo y español[7] y otras tantas que regulan la contaminación acústica en Catalunya[8].

Para su medición se utilizan varios índices de medición basados todos ellos en el Nivel de Presión Sonora (LP). Estos índices se diferencian por el intervalo de medida y por ser datos directos o promedios calculados. Entre ellos tenemos los siguientes: SEL, LAmax, Lkeq y el Nivel equivalente día-noche. Los primeros índices utilizan un intervalo de un segundo para la medición y el último es un valor promediado.

Aparte de los índices de medida hay que tener en cuenta la ponderación que determina la relación entre la intensidad del ruido, su frecuencia y la sensibilidad del oído humano a esa frecuencia. Intensidades iguales de sonidos con distintas frecuencias no son percibidos de la misma forma por el oído humano. Por esto se utilizan las curvas de ponderación y se obtienen los niveles sonoros ponderados que son los que deberían ser usados para determinar el nivel de contaminación acústica.

En cuanto a la medición de la velocidad con dispositivos móviles utilizando el GPS hay que tener en cuenta que calculan la velocidad con los datos puntuales recibidos desde los satélites que son visibles. Hay dos formas de calcular la velocidad:

- Teniendo en cuenta dos observaciones y midiendo la distancia en línea recta entre ellas y dividiéndolo entre el tiempo. Este método se ajusta muy bien a movimientos con velocidades uniformes, pero puede no ser muy real si el móvil no va en línea recta de un punto de medida a otro, si las medias tienen cierto retardo o si hay puntos en los que no se obtiene una medida.
- También se utiliza el efecto Doppler, con el que se puede calcular la velocidad a partir del cambio de frecuencia de las señales recibidas desde los satélites. La precisión de este método depende del tiempo que dediquemos a su cálculo, por lo que cuando queremos medidas en "tiempo real" no podemos esperar mucha precisión.

En el caso general de cualquier medida de alguna magnitud física habrá que tener en cuenta la **calibración de los dispositivos**. La calibración es el proceso mediante el cual se comparan los valores obtenidos por los instrumentos de medida con las medidas correspondientes a algún patrón de referencia. Como ejemplo podemos poner que distintos teléfonos móviles dieran la misma medida de ruido cuando están sometidos al mismo nivel de ruido.

4.2.5 Representación de observaciones sobre mapas WEB.

Una parte importante del proyecto es la publicación de las observaciones sobre un mapa accesible desde la WEB. En este sistema de publicación tenemos que distinguir varios elementos generales: el servidor web que aloja el mapa, el propio mapa, las observaciones añadidas como una capa más y la plataforma web desde donde se cargará el mapa.

Hay varios proveedores de cartografía digital que ponen a nuestra disposición una API con la que cargar mapas y modificar sus características de forma gratuita o de pago. Entre ellos, el más utilizado en la actualidad es Google Maps. Cada uno de estos proveedores nos permite cargar las observaciones con localización geográfica utilizando alguno de los siguientes formatos:

- **Kml**: es un lenguaje de marcado basado en xml para la representación de datos geográficos en tres dimensiones.
- **Gml**: es un sub-lenguaje XML para el modelaje, transporte y almacenamiento de información geográfica.
- **Shp**: es un formato propietario para el intercambio de información geográfica de la empresa ESRI Shapefile. Es un estándar de facto.
- **Csv**: formato abierto para la representación sencilla de datos en forma de tabla.
- **GeoJson**: Este formato abierto permite generar ficheros cumpliendo una serie de requisitos para después ser incluidos como una capa superpuesta sobre un mapa.

En el Ayuntamiento, en la actualidad, se utilizan dos plataformas para generar los mapas que después serán insertados en su web:

- **https://developers.google.com/maps/**: permite subir archivos con datos geográficos de dos tipos: vectoriales y de imágenes. Dentro de los vectoriales podemos subir como tipos: Kml[19] o kmz, csv, shp. Los archivos de imágenes pueden ir en formatos jpg, tif, tiff, png, etc. Los límites asociados a estos archivos los podemos ver en la página oficial[20]. En este proyecto utilizaremos el formato GeoJson para importar las observaciones.
- **http://cartodb.com**: utiliza formatos de archivos como shp (propietario), csv, excell, ESRI Shapefiles y GPX files.

Utilizando la API de alguna de estas plataformas y JavaScript[21] podemos importar un mapa genérico sobre nuestra web y después superponer una capa con los datos obtenidos en alguno de los formatos explicados anteriormente. Estos datos pueden venir en forma de texto desde un fichero o desde un servicio Web.

19 https://developers.google.com/kml/documentation/kml_tut?hl=es
20 https://support.google.com/mapsengine/answer/1272933?hl=es
21 https://developers.google.com/maps/documentation/Javascript/?hl=es

4.3 Propuesta de solución

La propuesta de solución es un sistema completo constituido por varios elementos activos comunicados entre si usando Internet. Los elementos principales son los que describimos a continuación:

- Aplicaciones móviles (App) para el envío de observaciones en formato Json. Noise como aplicación de uso urbano que utiliza las tres técnicas de localización geográfica y Speed que solo utiliza el GPS.
- Plataforma de recogida de observaciones con arquitectura de tipo REST. Se usará Sentilo que almacena dichas medidas sobre Redis.
- Servicio web, con arquitectura tipo REST, de extracción de observaciones en tiempo real en formato GeoJson programada en Java.
- Aplicación Java para la generación de ficheros históricos en formato GeoJson que se ejecute periódicamente de forma automática.
- Plataforma de generación de mapas y su API. En concreto se usará la versión 3 del API de JavaScript de Google Mapas[22].
- Web para la publicación del proyecto y de los mapas en tiempo real e históricos.

La comunicación entre ellos se realizará a través de interfaces que intercambiarán información con un formato determinado y abierto. A continuación se muestra de forma gráfica la estructura del sistema propuesto.

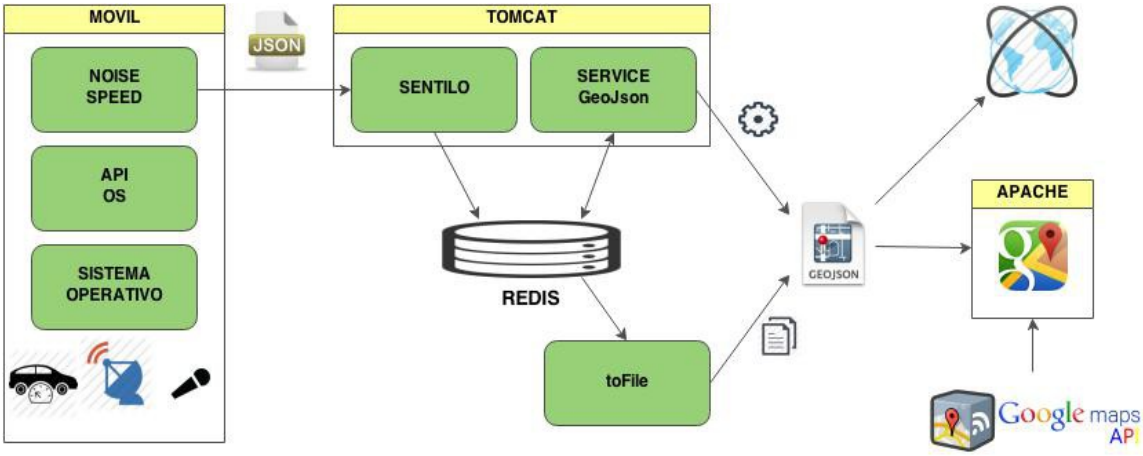


Ilustración 7: Elementos del sistema

22 https://developers.google.com/maps/documentation/Javascript/tutorial?hl=es

En la actualidad, según la consultora Kantar Worldpanel[23], la cuota de mercado de los distintos sistemas operativos de móviles en España es la siguiente:

%Cuota Sistemas Operativos sobre nuevos *Smartphones*	May - Jul 2012	May - Jul 2013
Android	82,5%	89,9%
iOS	2,9%	6,1%
Symbian	2,1%	0,7%
BlackBerry	9,4%	0,7%
Windows	1,7%	1,8%
Otros	1,4%	0,8%
Total *Smartphones*	100%	100%

Fuente: Worldpanel ComTech

Ilustración 8: Cuota de mercado

Por consiguiente, el equipo Scrum ha decidido que los primeros prototipos de las App's serán realizados sobre Android ya que tiene una mayor cuota de mercado en España y Catalunya. Por tanto tendremos como referencia la web para desarrolladores de Android[24].

El tipo de servicio web será de tipo REST ya que requiere menos recursos y es más fácil de implementar que una arquitectura con más capas como la que utiliza el protocolo SOAP. El lenguaje de programación será Java coincidiendo con el lenguaje de las app, este lenguaje también será utilizado para la implementación de la aplicación de generación de históricos.

La plataforma de mapas elegida será Google por ofrecer una API muy versátil y con una gran cantidad de documentación. La web de soporte del proyecto se implementará utilizando HTML y JavaScript como lenguaje cliente.

5 ANÁLISIS Y DISEÑO DE LA APP'S

5.1 Análisis de la API de Android

La app tendrá que comunicarse a través de una serie de Interfaces con tres elementos: el hardware del dispositivo a través de la API del sistema operativo, con el usuario a través de una interfaz gráfica y con la plataforma de recogida de datos utilizando la tecnología ***REST*** con el formato de envío ***Json.*** El elemento que más va a condicionar la aplicación móvil es la API sobre la que tenemos que programar, que en este caso es **Android**.

Android comenzó con la versión 1.1 y en la actualidad está disponible la 4.4. Ofrece distintos Framework API que se identifican por un número entero que indica la revisión ofrecida por cada una de las versiones de la plataforma de Android. A este número entero se le denomina “API Level” y va desde el 1 en la primera versión hasta el 19 actual[25]. Aparte de estas API internas (con almacenamiento persistente) también contamos con un repositorio de de API's externas.

Las API's nativas de Android tienen *packages* que contienen: *Interfaces, Classes y Exceptions.* Cada uno de estos packages nos ofrecerán una serie de servicios que utilizaremos para obtener:

23 http://www.kantarworldpanel.com/es/Noticias/El-28-de-los-mviles-en-Espaa-son-Nokia
24 http://developer.android.com/reference/packages.html.
25 http://developer.android.com/guide/appendix/api-levels.html

LOCALIZACIÓN GEOGRÁFICA:

Para obtener la localización geográfica podemos utilizar la API android.location de Android o la Google Play Service (más potente y con un mayor nivel de configuración)[26]. La API de Android nos ofrece las tres técnicas de servicios de localización: GPS más preciso, lento y con mayor consumo de batería, los puntos de acceso Wi-Fi cercanos y las antenas de telefonía, ambos métodos más rápidos, con menor consumo y más imprecisos.

La clase principal que vamos a utilizar para la localización geográfica es el ***LocationManager*** que nos da acceso al Location Service de Android y nos ayuda a escoger en cada momento el mejor proveedor. Tiene como uno de sus métodos el *getAllProviders* que nos proporciona los posibles proveedores de localización disponibles en el dispositivo aunque no estén activos o permitidos. Estos proveedores pueden ser tres: **network** (Wi-Fi y antenas de telefonía), **passive** y **gps**. Una vez tenemos las posibles opciones podemos analizar las características del servicio que ofrecen en cada momento para seleccionar el más adecuado. Por ejemplo ***getAccuracy*** es un método que nos da la precisión o ***getPowerRequirement***t para saber el consumo de recursos.

Android identifica a cada uno de estos proveedores con GPS_PROVIDER, NETWORK_PROVIDER y PASSIVE_PROVIDER. El modo passive no activa los sensores sino que nos ofrece la última medida realizada por la petición de otra aplicación. Por tanto puede estar muy desactualizada. Todos estos elementos están contenidos en el Network Location Provider (NLP) dentro del package ***android.location***.

Otra posibilidad para seleccionar proveedores que cumplan una serie de condiciones es ***Criteria,*** que tiene un método que nos da el mejor proveedor según un conjunto de criterios seleccionados ***getBestProviders***. Para saber si un proveedor concreto está activo o no contamos con ***isProvidersEnabled().***

Una vez seleccionado el proveedor tenemos que tener en cuenta que el TTFF del GPS, como puede ser muy largo, no se interroga directamente al proveedor sino que se busca la última posición conocida con ***getLastKnownLocation()***. Esta última medida hay que ir actualizándola utilizando una suscripción a las notificaciones que nuestro proveedor emite cuando hay cambios de posición usando ***requestLocationUpdates()***. Para estas suscripciones necesitamos una instancia de un objeto llamado ***LocationListener*** con el que definimos las acciones a realizar cuando hay un cambio en el proveedor. Este nos puede informar, por ejemplo, cuando el proveedor es deshabilitado, habilitado o cuando hay un cambio en la posición.

Para obtener la información del ***ManagerLocation*** utilizamos la clase ***Location*** que es una clase abstracta que contiene como mínimo la latitud, longitud y marca de tiempo, pero también puede contener la velocidad, la altura, etc. Para mantener el ***Location*** actualizado usamos el ***LocationListener*** mencionado anteriormente que usa el método ***requestLocationUpdates().***

Hay que adaptar las propiedades de cada proveedor a la naturaleza de las observaciones que queremos localizar geográficamente y a las limitaciones de cada zona geográfica. Algunas limitaciones son: la velocidad solo está disponible cuando utilizamos el GPS en el objeto location, el círculo de error de la localización cuando

26 http://developer.android.com/guide/topics/location/strategies.html#Updates

utilizamos las antenas de telefonía puede ser mayor a 1 Km, es imposible usar las antenas Wi-Fi en zonas no muy pobladas, el GPS no funciona en ubicaciones cerradas o entre edificios muy altos, etc.

MONITORIZACIÓN DEL ESTADO DE LA CONEXIÓN:

El dispositivo deberá estar conectado en todo momento, ya sea mediante la conexión de "datos" asociada con el proveedor de telefonía o mediante Wi-Fi. La aplicación estará informada sobre el estado y los cambios de dicha conexión gracias a la clase **ConnectivityManager**. Esta clase monitoriza las conexiones: Wi-Fi, GPRS, UMTS, etc, permitiendo a la aplicación adaptarse a las circunstancias. Se encuentra dentro del paquete android.net.ConnectivityManager.

ACCESO AL MICRÓFONO:

Android.media nos proporciona acceso a los dispositivos multimedia del teléfono. Entre ellos está el micrófono que utilizaremos para hacer las medidas de presión sonora. Algunas clases relacionadas con los dispositivos multimedia del teléfono son: *android.media.AudioFormat, AudioManager, AudioTrack y otros* [9].

ENVÍO Y EMPAQUETADO DE LA INFORMACIÓN:

El envío de la información se realiza en dos pasos, en primer lugar tenemos que generar el paquete en formato Json para enviarlo y en segundo lugar tenemos que realizar el envío. Para el parseo disponemos de la clase ***JsonObject*** (extends Object)**,** perteneciente al ***packaje org.Json***, que generará los los valores en formato Json. La creación de un método de este tipo puede generar una excepción por lo que tendremos que tratarla.

Para el envío de la información vamos a utilizar la clase HttpClient que será la encargada de realizar la comunicación con la plataforma. Una vez tenemos una instancia de nuestra clase creamos otra instancia de la clase HttpPost (o get) que contiene la URL de la plataforma. Para finalizar tenemos que asociar el objeto JsonObjsect con el HttpClient y para ello lo tenemos que convertir en un StringEntity y finalmente incluirlo en el httpPost con un setEntity.

Contamos con el ***package org.apache.http.cliente*** para la utilización de los métodos http: put, get, post, etc. La clase principal es el HttpURLConnection.

EJECUCIÓN DE TAREAS PESADAS:

Inicialmente en Android todo se ejecuta en el mismo hilo principal: actividades, servicios y gestión de la interfaz. Cuando tenemos que ejecutar una operación muy pesada puede generar una ralentización de toda la aplicación (no respuesta). Hay varias soluciones para esto: crear nuestros propios hilos secundarios (new Thread) o utilizar la clase AsynkTask. Con la primera forma es complicado usar los parámetros del hilo principal en el secundario y viceversa, por tanto usaremos el segundo método.

En el momento de decidir entre todas las funcionalidades tenemos que tener en cuenta que no todas están incluidas en todos los niveles API, por lo que tendremos que establecer un **nivel mínimo de API** y utilizar solo aquellas funcionalidades contenidas en él ya que las versiones de mayor nivel tienen todo lo anterior además de las nuevas características.

5.2 Noise: Arquitectura y comportamientos

Esta app es un ejemplo genérico para el envío de cualquier parámetro susceptible de ser medido desde un teléfono móvil. Las medidas de ruido que enviará son ficticias y generadas de forma aleatoria.

Será programada pensando en entornos urbanos donde tenemos una gran cantidad de puntos de acceso disponibles para la localización geográfica y en algunas situaciones habrá problemas con la localización de satélites cuando la densidad de edificios muy altos sea alta. Por este motivo, utilizará los tres proveedores disponibles de localización geográfica: puntos de acceso Wi-Fi, antenas de telefonía y GPS. Gracias a esto también será posible utilizarla en interiores. Como desventaja tenemos que cuando la medida provenga de las antenas de telefonía la precisión será muy baja.

La app está compuesta por una serie de subsistemas que se comunican entre sí a través de interfaces y su arquitectura tiene 4 capas:

- **Hardware del dispositivo** al cual accederemos utilizando los servicios ofrecidos por la API. La cantidad y calidad de los sensores depende de cada dispositivo. El hardware que tenemos disponible depende del dispositivo sobre el que se instala la aplicación. Tenemos que distinguir entre los sensores (GPS), arquitectura del microprocesador (ARM mayoritariamente), tipo y modelo de micrófono, formas de conexión y si dispone o no de tarjeta SD.
- **API: l**a API depende del sistema operativo que se ejecuta sobre el dispositivo. La arquitectura sería la misma para cualquier plataforma, solo cambiaría la API sobre la que desarrollamos la aplicación.
- **Aplicación móvil** que recibe, procesa y envía los datos de ruido, tiempo y localización.
- **Interfaz con el usuario** a través de la cual el propietario del dispositivo podrá activar y desactivar la aplicación, configurarla, etc. También podrá ver los valores de las observaciones que está enviando, además de las alarmas y mensajes necesarios para el funcionamiento.

La siguiente imagen muestra la arquitectura citada:

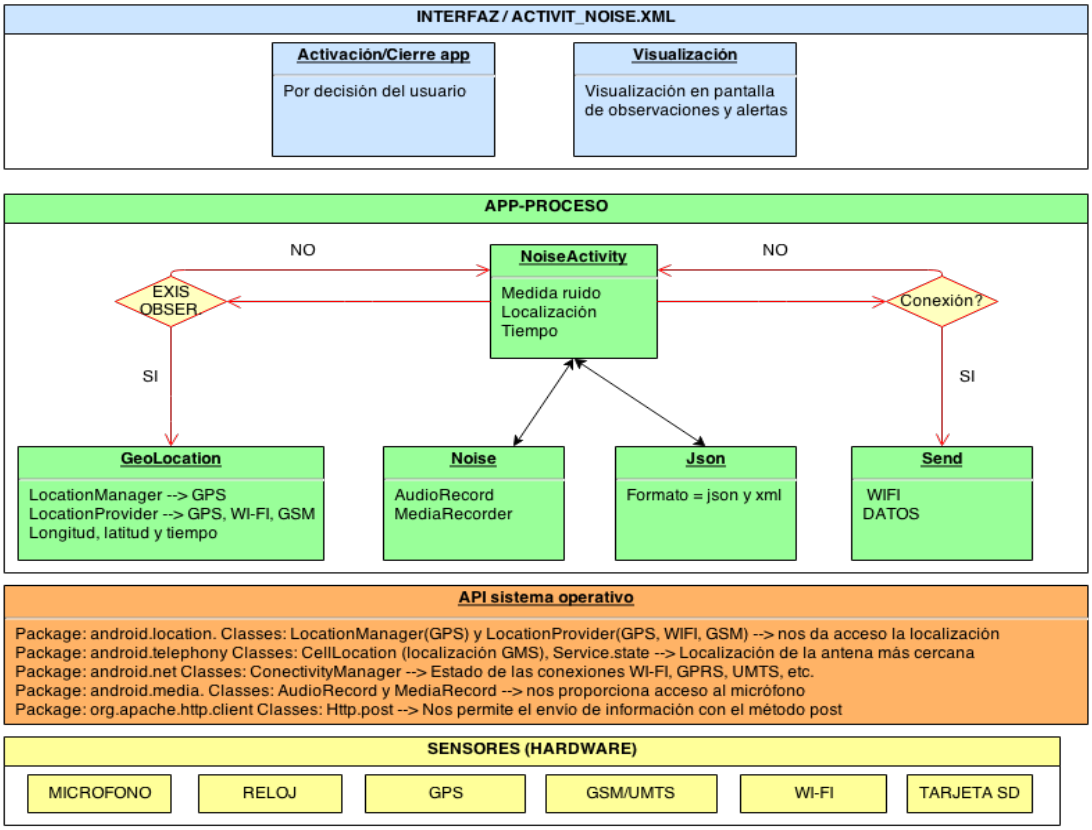


Ilustración 9: Arquitectura de la APP

A continuación vamos a describir cada uno de los módulos de la app:

- **ActivitiNoise**: solicitará la localización y las medidas de ruido. Después enviará los datos correspondientes al módulo de empaquetado y si hay alguna conexión disponible enviará los datos al módulo Send. La marca de tiempo de la observación la pondrá la plataforma en la recepción.
- **GeoLocation:** este módulo extraerá la localización geográfica del mejor proveedor disponible con la siguiente prioridad: GPS, WI-FI o GSM. En el caso de la app Speed también extraerá la velocidad cuando se posible.
- **Noise**: este módulo podría enviar cualquier otro parámetro como temperatura, luminosidad, humedad, etc. En este caso simula el envío de las medidas de ruido tomadas por el móvil.
- **Json**: este módulo cogerá los datos procedentes del proceso de datos y los empaquetará con el formato Json.
- **Send**: este módulo se encargará de decidir en cada momento la mejor forma de enviar los datos dependiendo de si el móvil está conectado a una red WI-FI (envío prioritario) o si se deben enviar por la conexión de datos.

Dependiendo de los sensores que están activos y de la líneas de datos disponibles la aplicación debe actuar de diferentes formas:

- La localización se recogerá con este orden de prioridad: GPS, WI-FI y antenas de telefonía. Cada cierto tiempo, la aplicación actualizará el estado de estos servicios y utilizará el posicionamiento del servicio que está activo con el orden de prioridad descrito.
- Cuando la localización no sea posible por ninguno de los métodos no enviará nada y a intervalos de tiempo interrogará al sistema hasta conseguir una localización, momento en el cual comenzará el envío otra vez.
- Cuando el dispositivo no tenga posibilidad de envío de la información advertirá al usuario e interrogará al sistema hasta tener una conexión.
- Los datos serán enviados por WI-FI como primera opción y por la línea de datos como segunda opción.
- Cuando el GPS esté activo pero no tenga todavía observación (durante el arranque del sensor) se tomará la observación del proveedor NETWORK. Esto es debido a que el GPS puede tardar hasta 15 minutos en obtener la primera localización aún estando activo.
- Cuando la medida de la observación del GPS tenga una antigüedad de más de 30 segundos será desechada y tomará la que procede del NETWORK. Este comportamiento dependerá de la variable "***desfase***" que mostrará la diferencia entre el momento actual y el momento en el que se tomo la medida.
- Con el GPS desactivado la antena Wi-Fi, aunque no esté asociada a ningún punto de acceso, obtendrá los datos de localización geográfica con prioridad sobre las antenas de telefonía. En este caso la conexión de datos se utilizará para el envío de observaciones y para las consultas de la ubicación a través de la MAC de los puntos de acceso detectados por la Wi-Fi.

5.3 Speed: Arquitectura y comportamientos

En este caso la app está pensada para usar en todo tipo de entornos (urbanos, rurales, interurbanos, etc), además la precisión es muy importante para el cálculo de la velocidad de desplazamiento. Por este motivo, solo se usará el GPS para la localización geográfica. En cuanto al envío de observaciones se adaptará al proveedor que esté activo (Wi-Fi o datos) aunque por las peculiaridades de la aplicación en la gran mayoría de ocasiones se utilizará la conexión de datos dada por nuestro proveedor de telefonía.

La estructura general de esta aplicación es idéntica a la anterior, solo hay algunas diferencias en cuanto al uso de sensores y número de clases de la aplicación:

- En la capa física no utiliza el sistema de localización a través de puntos de acceso Wi-Fi ni por antenas de telefonía. Estos dos métodos pertenecen al proveedor NETWORK.

- De la API de Android básicamente utiliza las mismas clases, salvo las relacionadas con el proveedor NETWORK. Para el cálculo de la velocidad se utiliza un método del objeto **Location** perteneciente a la API de Android.
- La interfaz también es muy parecida, solo difiere en la información mostrada.

En cuanto a los comportamientos de la aplicación los que difieren de la anterior son los siguientes:

1. La localización geográfica solo se recogerá del proveedor GPS actualizándose las medidas cada 3 segundos.
2. Cuando la localización no sea posible no enviará nada hasta que el GPS dé alguna observación. Si la velocidad no está disponible y la localización sí, enviará un valor especial para marcar como “sin datos de velocidad”.
3. Enviará la información por la conexión disponible en cada momento adaptándose a los cambios de conexión: Wi-Fi o datos. En este caso esto se podría simplificar ya que en muy pocas ocasiones se podrá estar conectado a la Wi-Fi y desplazándose al mismo tiempo.
4. Cuando el desfase sea superior a 30 segundos no se enviará la media y se mostrará un mensaje en la interfaz.
5. En ambos casos la interfaz mostrará avisos sobre los motivos del mal funcionamiento o, en su caso, la información enviada.

6 PREPARACIÓN DEL ENTORNO DE DESARROLLO

Como entorno de desarrollo vamos a utilizar Eclipse Indigo 3.7.2 con los plugins necesarios para aplicaciones para dispositivos móviles (app's) y su empaquetado, servicios web de tipo REST y su empaquetado y proyectos Java en general.

También se prepararán los repositorios locales y remotos para publicar el código que se realice y se configurará Eclipse para trabajar con ellos. Como entorno de pruebas se utilizará JUnit también sobre Eclipse.

6.1 SDK de Android y plugin de Eclipse ADT

Para el desarrollo de la aplicación sobre Android necesitamos herramientas específicas para poder compilar el código, empaquetar la aplicación y enviarla a un “teléfono software” (emulador) o a un teléfono físico. Todas estas herramientas están incluidas o se pueden obtener con el SDK (Software Development Kit) de Android[27]. Después de descargarlo y desempaquetar las SDK tendremos un gestor de descarga y un emulador de terminal “vacío” al que le falta el propio Android que debe ejecutar.

27 http://developer.android.com/sdk/index.html

La estructura que queda en el directorio de descarga tiene los siguientes subdirectorios:

- add-ons: ampliaciones a versiones concretas de Android de terceros.
- platforms: contendrá el software de las diferentes versiones que instalemos, ahora está vacío.
- Tools: contiene las herramientas básicas para la gestión de las SDK

Después de esto necesitaremos descargar las "Android Platforms" que contienen la información específica de cada una de las versiones del SO. Para ello desde /tools ejecutamos *./android* que es un gestor de descarga que nos permite instalar tres tipos de software:

- Tools: incluye las herramientas que hemos instalado antes y las platforms-tools con herramientas necesarias para el desarrollo.
- Android x.x.x (API X): habrá una por cada versión Android disponible. Esto es lo que se denomina Android Platforms. Por defecto está marcada la última versión pero podemos descargar e instalar las que queramos.
- Extras: dentro de este grupo hay una gran cantidad de software entre el que se encuentra el driver USB para comunicar sistemas Windows con el móvil.

Dentro de Android x.x.x tenemos que elegir las Platforms que deseamos tener instaladas en el equipo de desarrollo y después tenemos que crear los emuladores de terminal AVD[28] para poder ejecutar nuestra aplicación. Cada AVD está compuesto por:

- Perfil de hardware: que describe las características físicas del terminal.
- Imagen del sistema: el software que ejecuta (Versión de Android).
- Opciones adicionales: aspecto para imitar el terminal físico.
- Almacenamiento dedicado para almacenar las aplicaciones instaladas, la tarjeta de memoria, etc.

Para montar los emuladores de cada una de las versiones que deseemos solo tenemos que ejecutar desde *./tools/./android avd* y elegir el nombre del emulador, el dispositivo físico sobre el que queremos trabajar, la plataforma que ejecutará dicho dispositivo, la arquitectura de la CPU, si queremos que tenga SD, si queremos o no aceptar Snapshot, etc. Después de esto ya tendremos los emuladores preparados para probar la aplicación en distintos dispositivos y versiones de Android.

Con las SDK de Android podemos hacer todo lo necesario para el desarrollo de aplicaciones, pero es más fácil trabajar sobre un entorno de desarrollo como Eclipse instalando un plugin denominado ADT[29] ofrecido por Google.

28 Android Virtual Device.
29 Android Development Tools.

Una vez instalado el plugin en Eclipse podemos acceder a dos herramientas usadas anteriormente: el gestor de las SDK y el Android Virtual Device Manager para gestionar los AVD. Tenemos también dos nuevas: Android **LINT** para detectar errores comunes y asistentes que nos ayudan con los proyectos de Android en Eclipse.

Una última cuestión a tener en cuenta es que cuando comenzamos con un proyecto de desarrollo de una APP sobre Android tenemos que decidir sobre el nivel de API para la que la vamos a programarlo (Build SDK) y el mínimo nivel de API que contiene las características utilizadas (Minimum Required SDK). Nunca tenemos que utilizar características que estén en Buid pero que no estén en el Minimum para garantizar la correcta ejecución sobre plataformas correspondientes al nivel mínimo de API considerado. La herramienta LINT, citada anteriormente, nos ayuda con el control de estas posibles situaciones.

Con esto ya estamos en disposición de comenzar con el desarrollo de la aplicación sobre cualquier versión de Android. La última estadística realizada por Android sobre la cuota de mercado de cada una sus versiones terminó el 12 de Diciembre de 2013 arrojando los siguientes resultados:

Version	Codename	API	Distribution
2.2	Froyo	8	1.7%
2.3.3 - 2.3.7	Gingerbread	10	26.3%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	19.8%
4.1.x	Jelly Bean	16	37.3%
4.2.x		17	12.5%
4.3		18	2.3%

Ilustración 10: Cuota de mercado por versión de Android

A la luz de la tabla anterior vamos a generar tres dispositivos que corran bajo versiones Android con un nivel de API diferente: 2.3.3 (API 10), 4.0.X (API 15) y la 4.1 (API 16). Con esto tendremos nuestra aplicación comprobada en más del 85% de los dispositivos del mercado. Como API mínima pondremos la 8 en todos los casos.

Cuando terminamos un proyecto en Java podemos empaquetarlo para ser enviado, almacenado y posteriormente instalado. Tradicionalmente se ha utilizado la extensión ***.jar*** (Java Archive), archivos comprimidos en formato ZIP que contienen la aplicación preparada para ser instalada.

Para la distribución de aplicaciones, Android utiliza un fichero comprimido con con ZIP y extensión ***.apk*** (Application PacKage File). Este no es más que una variante del formato ***.jar*** descrita antes que suele tener una estructura interna determinada con directorios y ficheros. Algunos elementos que suele contener: manifest file, el/los certificados de la aplicación, código compilado para una o varias arquitecturas del procesador, fuentes sin compilar, etc.

Eclipse nos ofrece la posibilidad de realizar el empaquetado utilizando este formato y genera el certificado en modo "debug" de forma automática[30]. La siguiente imagen nos muestra el proceso de forma simplificada:

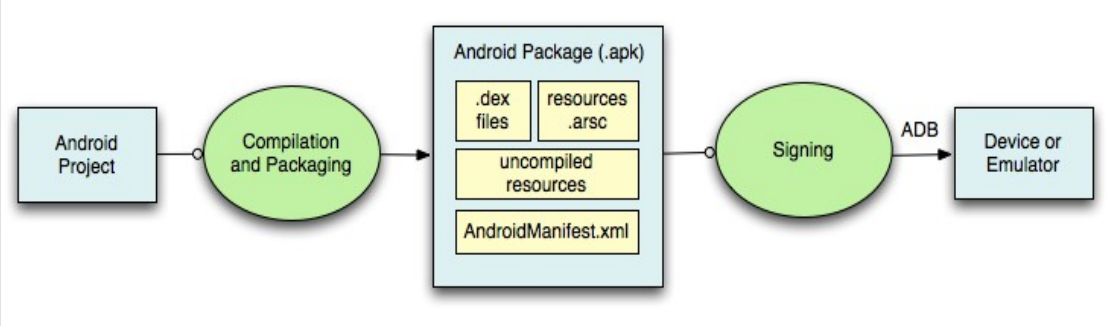


Ilustración 11: Proceso de empaquetado de App's

6.2 Desarrollo del servicio Web.

Como el servicio elegido es de tipo REST preparar el entorno de trabajo sobre Eclipse es muy sencillo. Simplemente tenemos que abrir un nuevo proyecto de tipo "Dynamic Web Project" con lo que se generará un esqueleto completo con la estructura adecuada. Después solo hay que añadir las clases necesarias para el proyecto.

Para poder probar el servicio web en local tenemos dos posibilidades:

- Montar toda la plataforma en local para poder implementar el servicio web y ejecutarlo contra el gestor Redis local.
- Implementar el servicio web para que lance las peticiones al servicio remoto. El inconveniente de este método es la lentitud del sistema cuando el resultado traiga gran cantidad de información. Otro inconveniente es que hay que configurar el gestor Redis para que escuche peticiones por la ip pública en lugar de escuchar solo por la 127.0.0.1. La ventaja es que se trabaja en real y no hay que montar toda la plataforma en el equipo de desarrollo.

De cualquiera de las maneras necesitamos tener una instancia de Tomcat7 en el equipo de desarrollo para desplegar y probar el servicio web. Para ello añadimos desde "Servers" un nuevo servidor eligiendo la misma versión que tenemos instalada en el servidor. Eclipse también nos facilita el empaquetado en formato war. Simplemente tenemos que exportar y elegir que parte del proyecto queremos empaquetar.

30 http://developer.android.com/tools/building/index.html

Para la automatización de las pruebas vamos a utilizar **JUnit** que es un FrameWork Open Source para realizar pruebas unitarias y de integración. Solo utilizaremos la anotación @test antes de cada uno de los métodos y el assertTrue para comprobar si las condiciones puestas son ciertas. También podríamos utilizar el assertArrayEquals y el asserEquals.

Paralelamente a este generaremos otro proyecto genérico para implementar las pruebas sobre **JUnit** en el que se tiene que incluir un "**JUnit Test Case**" e implementar un cliente del servicio que envíe todas las pruebas necesarias sobre el servidor.

6.3 Generación del repositorio de control de versiones

Para mantener las sucesivas versiones del software desarrollado, dar un acceso rápido y actualizado a los tutores y tener una copia de seguridad de todo el código del proyecto generaremos un repositorio con control de versiones local y remoto en el servidor virtual utilizando GIT.

Esta herramienta está licenciada bajo GPLv2 y tiene una estructura distribuida de forma que cada usuario que colabore en el proyecto tendrá su propio repositorio. Por tanto, la forma de trabajar será manteniendo un repositorio local en el ordenador de desarrollo y otro repositorio en el servidor virtual VPN[31].

Otra diferencia fundamental de GIT con respecto a otros sistemas de control de versiones es que cada cambio en el código genera una instantánea de todo el proyecto y no una versión del archivo modificado. Además de esto, en cada commit se hace una suma de verificación (hash:SHA) que proporciona a GIT un control de la integridad de cada una de las versiones almacenadas.

Al repositorio local añadiremos el repositorio remoto utilizando el protocolo ssh con el que tenemos los permisos de lectura y escritura:

```
git remote add servidor ssh://root@www.smartcitything.es/var/www/git/app-speed.git
```

Después de esto podemos sincronizar los dos repositorios ejecutando el siguiente comando en local:

```
git push servidor master
```

Si varias personas trabajan con el mismo repositorio remoto antes de modificar nuestra copia local la tendríamos que actualizar con un ***git pull –rebase.*** En este caso sobre el servidor nunca habrá ningún cambio ya que solo hay un desarrollador.

31 Virtual Private Server

Para acceder al repositorio se pueden emplear varios métodos:

- **SSH:** será necesario tener una cuenta en el sistema y el acceso será de lectura y escritura. Esta forma de acceder será para los usuarios que colaboran con el proyecto. Para clonar el repositorio localmente en un equipo con GIT instalado solo tenemos que ejecutar

```
git clone usuario@37.187.44.213:/var/www/git/app-Noise.git
```

- **GitWeb:** también se podrá acceder al repositorio a través de la página web del proyecto, para esto utilizaremos GitWeb.

Por otro lado, si queremos visualizar la documentación realizada con **JavaDoc** actualizada de forma automática tenemos que activar un anclaje en el repositorio remoto sobre la base de datos remota. Para ello solo hay que ir al directorio /.git/hooks y renombrar el fichero **post-update.**

Para poder utilizar el repositorio directamente sobre Eclipse tenemos que instalar el plugin **Egit** que permitirá utilizar el control de versiones desde el entorno de Ecplise. Después en la configuración solo habrá que compartir el proyecto y añadir el repositorio remoto con las credenciales de acceso de una cuenta con permisos de lectura y escritura. De esta forma tendremos los repositorios configurados como muestra la siguiente imagen:

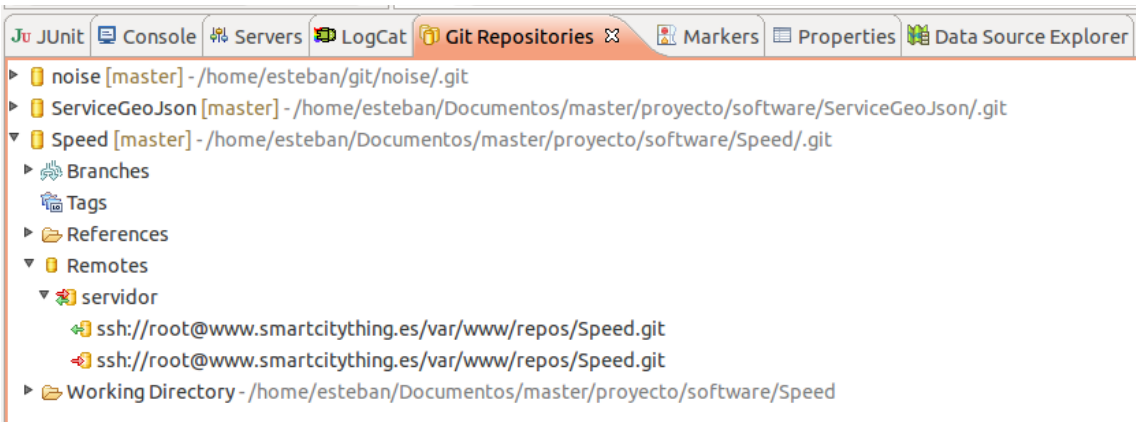


Ilustración 12: Repositorios de control de versiones: GIT

7 IMPLEMENTACIÓN Y USO DE LAS APP's

7.1 Desarrollo

Como se ha comentado anteriormente, hemos elegido como plataforma de implementación Android, cuyo lenguaje de programación es Java. La aplicación será programada para su compilación sobre la versión 4.4.2 de Android que utiliza la API Level 19 y el mínimo requerido será el Level 10.

Vamos a realizar dos aplicaciones: Noise y Speed. Esto es debido a que cada uno de los parámetros tienen una serie de peculiaridades que nos permitirá mostrar la diferente forma de tratar a cada uno de ellos.

La estructura de la aplicación Noise es:

- **Directorio src** que contiene el paquete cat.bcn.Noise con las clases principales del proyecto:
  - **NoiseActivity**: es la clase principal del proyecto que incluye la comunicación con la interfaz de usuario, comprobaciones de conexión de datos, llamadas para obtener la localización geográfica y las medidas de ruido. En caso positivo envía las observaciones con formato Json.
  - **Noise**: es una de las posibles funciones que obtienen los parámetros que se quieren enviar. En este caso simula la obtención del ruido ambiental.
  - **GeoLocation:** es la clase que extrae la localización geográfica utilizando el proveedor que esté disponible dando prioridad al proveedor GPS respecto al NETWORK. Gracias al Listener incluido en dicha clase la app se adapta en tiempo real a la activación y desactivación del GPS y Wi-Fi. Si ninguno de los dos está activo utilizará las antenas de telefonía para obtener la localización geográfica.
  - **Json**: recibe las observaciones de ruido, tiempo y localización y devuelve un paquete en formato Json que será el que se envíe.
  - **Send**: envía las observaciones en formato Json utilizando el protocolo http a la plataforma Sentilo.
- **Directorio res/layout** que contiene el archivo ***activity_Noise.xml*** con la descripción de la interfaz de comunicación con el usuario del móvil. En este se describen los botones, cuadros de texto para mostrar información y etiquetas identificativas.
- **Archivo AndroidManifest.xml** que entre otras cosas tiene el nombre del paquete, la versión, la Activity principal y los permisos necesarios sobre el dispositivo móvil para el funcionamiento de la aplicación. Estos permisos se identifican con android.permission más un parámetro que identifica el permiso en concreto. En este caso los permisos son :
  - INTERNET: permite a las aplicaciones abrir network socket.
  - ACCESS_NETWORK_STATE: permite que una aplicación acceda a la información sobre las posibles conexiones activas.
  - ACCESS_COARCE_LOCATION: con este permiso las app pueden acceder a la información sobre localización geográfica del dispositivo procedente de las antenas de telefonía o de redes Wi-Fi.

- ACCESS_FINE_LOCATION: con este permiso las aplicaciones pueden acceder a la localización procedente del GPS, de las antenas de telefonía y de las redes Wi-Fi.

La documentación del código está realizada con JavaDoc y se publica en el mismo servidor que la plataforma y el repositorio. El diagrama de clases del directorio src es el siguiente:

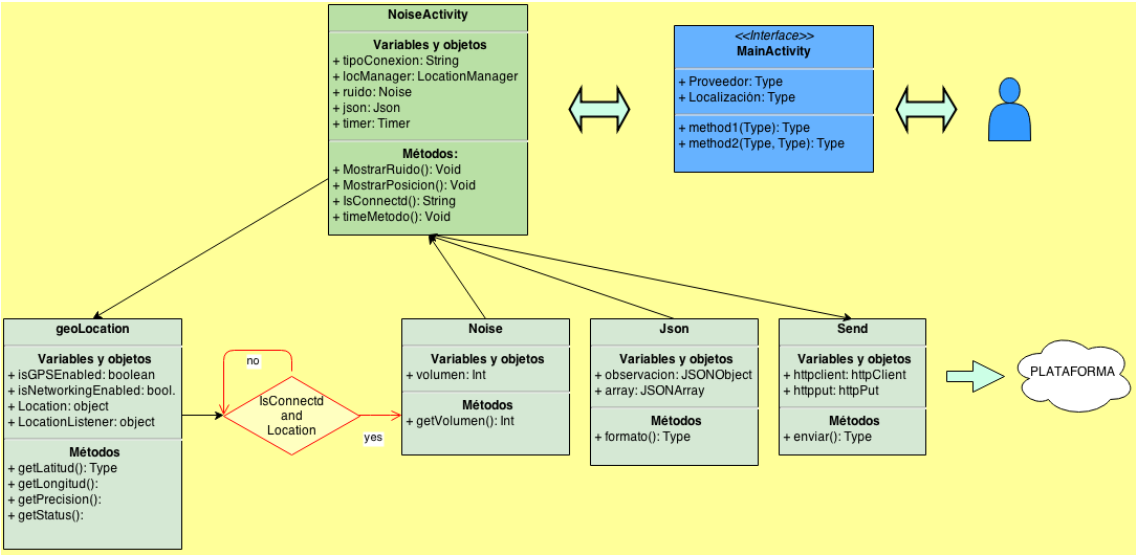


Ilustración 13: Diagrama de clases

La estructura de la aplicación **Speed** es ligeramente diferente ya que no necesita la clase que extrae las medidas de ruido ambiental. En este caso la velocidad de desplazamiento la obtendremos del objeto **location** de la API de Android. Hay que tener en cuenta los siguientes aspectos:

- Solo obtendremos la velocidad del proveedor GPS.
- Cuando utilizamos el método ***getSpeed*** del objeto Location nos puede dar medidas de velocidad 0.0 aunque no haya medida, por lo que utilizaremos el método ***hasSpeed*** para conocer primero si hay medida y en caso negativo enviaremos un -1 que es un valor imposible para el módulo de la velocidad.

7.2 Pruebas y uso de las app's

Las pruebas de las app's serán realizadas sobre distintos dispositivos físicos y sobre los emuladores citados anteriormente. Utilizar emuladores tiene la gran ventaja de disponer de una gran cantidad de posibles versiones de API y modelos pero la ejecución de dichas pruebas es muy lenta y no se pueden realizar pruebas de campo.

Con los dispositivos físicos la ejecución de las pruebas es mucho más rápida y se pueden realizar pruebas en el exterior. Los dispositivos móviles utilizados son:

- LG L3II con una pantalla de 3.2”
- Samsung Galaxy S i9000 con un pantalla de 4”
- Samsung
- Samsung Galaxy Tab

Un proceso sistemático y completo de pruebas excede los objetivos de este proyecto debido a la gran cantidad de marcas y modelos de dispositivos, niveles de API y circunstancias que se pueden dar en el uso habitual. Los aspectos que han sido considerados en las pruebas y han puesto de manifiesto algunos problemas resueltos posteriormente son los siguientes:

- **Ciclo de vida de la aplicación.**
  - Pulsaciones inadecuadas de los botones: “Activar” y “Cerrar”. Hay que controlar los posibles problemas que surgen cuando el usuario no los pulsa adecuadamente. Para resolver este problema solo es visible un botón en cada momento para evitar arrancar varias instancias al pulsar varias veces el “Activar” o “Cerrar” sin que esté abierta con anterioridad lo que provocaba una excepción.
  - Giro del móvil mientras está activa: en este caso la aplicación se reinicia. Esta incidencia no ha sido resuelta.
  - Inicio del estado de reposo del móvil: la aplicación no se ve afectada.
- **Estado del hardware del dispositivo y permisos sobre los servicios.**
  - Si el GPS o la Wi-Fi no están activos o la aplicación no tiene suficientes permisos provocaba excepciones por no contener valores nulos algunas variables. La solución pasa por hacer comprobaciones de existencia y tratar las excepciones.
  - GPS activo pero sin observaciones. Se resuelve con un test de existencia de la observación y el uso del proveedor NETWORK en caso negativo.
  - GPS con observaciones desfasadas. Se implementa una variable que mide el desfase para poder desechar estas medidas y utilizar las observaciones del NETWORK.
  - Prioridad de los proveedores de la localización geográfica.
  - Cambio del estado de los proveedores de localización geográfica durante la ejecución de la aplicación para comprobar cómo se adapta.
  - Cambio del estado de los proveedores de conexión de datos para comprobar la adaptación.
- **Diferencias entre distintos dispositivos móviles.**
  - Distintos tamaños de pantalla de cada uno de los dispositivos. Hemos ajustado la precisión con la que se representaba la longitud y latitud para poder incluirlo en una línea ya que en la pantalla del LG l3II no entraba toda la información.

El uso dependerá del entorno en el que se encuentre el dispositivo. Para zonas abiertas la localización geográfica será posible con el GPS que nos da las medidas más precisas. En ciudades pobladas y edificios no muy altos tendremos a nuestra disposición los dos proveedores (NETWORK y GPS), en interiores y zonas con una gran cantidad de edificios altos tendremos solo disponible el proveedor NETWORK.

En cuanto a la posibilidad de envío de datos tenemos que solo cuando estemos asociados a un punto Wi-Fi podremos prescindir del servicio de datos de nuestro proveedor.

Dependiendo de los sensores que estén activos el **consumo** puede llegar a ser considerable, este es un aspecto que hay que tener muy en cuenta en los dispositivos móviles ya que su autonomía depende de esto. Para calcular el gasto de batería vamos a utilizar una app llamada BatteryDrain que nos da el porcentaje de consumo por hora. El consumo de un dispositivo a otro es muy variable por lo que mostraremos las pruebas realizadas para el Samsung Galaxy 9000i sobre los siguientes escenarios de consumo:

- GPS, Wi-Fi y línea de datos activa: en esta situación pueden funcionar las dos aplicaciones y el consumo está entorno al 24% de la batería por cada hora de funcionamiento. Esta medida se ha realizado en un intervalo de media hora.
- Wi-Fi activa y línea de datos activa: en este caso solo puede funcionar noise y su consumo está alrededor del 30%.
- GPS conectado y ninguna línea de datos activa: el consumo está alrededor del 4%. En estas condiciones evidentemente ninguna aplicación puede funcionar.
- El gasto mínimo del móvil está en torno a un 1% de batería por hora.

Estos datos solo pueden ser orientativos ya que el estudio realizado no es riguroso y no han sido aislados los consumos de cada sensor. Además de comprobar que las medidas de consume dependen del nivel de carga de la batería. Como conclusión podemos deducir que para este modelo el tiempo máximo de uso de las aplicaciones no superará las **5 horas** en ningún caso.

La **precisión** asociada a cada uno de los proveedores es muy variable y en algunos casos se producen anomalías como las mostradas en las ilustraciones 20 y 21. En cualquier caso tenemos que el GPS suele mostrar precisiones comprendidas entre 15 y 100 metros, la Wi-Fi entre 30 y 150 metros y las antenas de telefonía al rededor de 500 a 1000 metros.

Por último destacar que el **desfase** de las medidas suele ser más alto para el GPS que en muchos casos supera los 10 segundos, mientras que el proveedor NETWORK (que incluye las antenas de telefonía y los puntos Wi-Fi) suele tener desfases de uno o dos segundos.

8 INSTALACIÓN DE LA PLATAFORMA DE RECEPCIÓN Json

8.1 Compilación del código de Sentilo

Anterior a la compilación del código de Sentilo tenemos que tener instalado el **KIT JDK 1.6.0** y **Maven 2.2.1+** y exportar las dos variables de entorno relacionadas con cada uno de ellos. Para ello editamos el /etc/environment y añadimos sendas directivas que contienen el path de dónde están instaladas las dos herramientas anteriores:

```
export Java_HOME="/usr/lib/jvm/Java-6-openjdk-i386"
export M2_HOME="/usr/share/maven2"
```

Realizado esto ya podemos obtener el código fuente de Sentilo y compilarlo utilizando Maven. Descargamos el código[32], lo descomprimimos en /opt y ejecutamos el siguiente script.

```
/opt/sentilo-master/scripts# ./buildSentilo.sh
```

Este script lanza un proceso que compila el código, genera el .war de la aplicación web que después tendremos que desplegar sobre Tomcat y genera los script para arrancar los distintos procesos que componen la plataforma. Durante el proceso se van realizando una serie de test que nos informan sobre el resultado del proceso.

En el caso de querer modificar alguna de las opciones que trae por defecto Sentilo como los números de puerto, usuarios, contraseñas, etc, tendremos que hacer los cambios antes de la compilación en los directorios concretos descritos en la documentación. Si queremos modificar el código o no compilar alguno de los artefactos podemos editar el código con Eclipse para realizar estos cambios.

Una vez que tenemos el código compilado podemos extraer el war de la aplicación web que gestiona la plataforma Sentilo.

```
root@vps27430:/opt/sentilo-master/sentilo-catalog-web/target# ls -l
total 12796
…..................
drwxr-xr-x 5 root root     4096 Feb  8 18:18 sentilo-catalog-web
-rw-r--r-- 1 root root 13067884 Feb  8 18:18 sentilo-catalog-web.war
…..................
```

8.2 Instalación y configuración de los servidores

Ahora hay que configurar todos los servidores que utiliza Sentilo antes de desplegar la aplicación web y tratar de arrancar la plataforma. Como ya se mostró en la descripción de la PSAB, la plataforma necesita: Tomcat7, MySQL, Mongodb y Redis.

32 https://github.com/sentilo/sentilo/archive/master.zip

Realizamos la instalación completa de **Tomcat** (docs, examples y admin) y exportamos la variable de entorno CATALINA_HOME en el /etc/environment indicando dónde lo hemos instalado en /usr/share/tomcat7.

Una vez terminada la instalación tenemos que agregar ciertos usuarios para que puedan administrar el servidor. En este caso los usuarios están en /etc/tomcat7/tomcat-user.xml. Dentro de este directorio vamos a añadir un usuario administrador añadiendo las siguientes líneas:

```
<role rolename="manager-gui"></role>
<user username="esteban" password="contraseña" roles="manguer-gui"></user>
```

Para conectarnos simplemente vamos a la url ***http://ip_servidor:8080*** y seleccionamos el web-manager para gestionar nuestros servicios web (cargar, desplegar, etc). Desde aquí será desde donde despleguemos el .war producido por Sentilo y posteriormente el servicio web implementado para extraer las observaciones.

Después de la instalación de **MySQL** tenemos que crear una base de datos llamada **sentilo** y un usuario administrador de dicha base de datos llamado **sentilo-user** con la contraseña sentilo-pwd (estos son los parámetros por defecto que se deben cambiar modificando los parámetros de configuración como indica en la documentación). Generada la base de datos ejecutamos un script para la creación de las tablas necesarias para el uso interno de Sentilo que está almacenado en ***sentilo-agent-relational/src/main/resources/bd*** y se llama agent_mysql.sql. Para ello simplemente ejecutamos:

```
mysql -u sentilo-user -p < agent_mysql.sql
```

Para que Java pueda acceder a MySQL fácilmente vamos a instalar el JDBC (Java Data Base Connectivity) que es un conjunto de clases que permitirán el acceso fácil desde las clases usadas por Sentilo para acceder a MySQL. Después podemos exportar una variable de entorno para señalar el directorio de instalación o almacenar este driver en el propio .war en el directorio /lib del proyecto. Sentilo utiliza MySQL para el almacenamiento permanente, inicialmente no se utiliza a no ser que ejecutemos un servicio llamado relational-agent.

Después instalamos **Redis** y activamos la autenticación. Para ello editamos el archivo /etc/Redis/Redis.conf para añadir la palabra de paso para la autenticación:

```
requirepass sentilo
```

Una vez reiniciado el servicio podemos comprobar que funciona la autenticación con:

```
root@vps27430:/etc/Redis# nano Redis.conf
root@vps27430:/etc/Redis# Redis-cli
Redis 127.0.0.1:6379> set h helloworld
(error) ERR operation not permitted
Redis 127.0.0.1:6379> auth sentilo
OK
Redis 127.0.0.1:6379> set h helloworld
OK
Redis 127.0.0.1:6379>
```

Después de la instalación de MongoDB tenemos que añadir un usuario administrador del sistema gestor de bases de datos. Para ello nos autenticamos como administradores, creamos la base de datos sentilo con el usuario sentilo y ejecutamos el script que nos genera todos los clave/valor necesarios para el uso de sentilo.

```
$ ./mongo
> use admin                              → base de datos de administración
> db.addUser("root", "contraseña")       → añadimos usuario administrador
> db.auth("root", "contraseña")          → nos logeamos
> use sentilo                            → añadimos la base de datos sentilo
> db.addUser("sentilo", "sentilo")       → añadimos un usuario admin a la base
```

Después solo tenemos que ir al directorio **./scripts/nongodb/** de sentilo y ejecutar dos script, el primero para incluir la estructura necesaria y el segundo para incluir datos para realizar los test. Para ello solo hay que ejecutar:

```
mongo -u sentilo -p sentilo sentilo init_data.js
mongo -u sentilo -p sentilo sentilo init_test_data.js
```

8.3 Despliegue y puesta en marcha de la plataforma

Después de esto ya tenemos todos los servidores preparados y solo tenemos que desplegar los artefactos y ponerlos en marcha. Dichos artefactos quedan generados en el primer paso, pero si queremos ir generando uno a uno tenemos que ejecutar los siguientes goal de maven:

```
~/sentilo-master/mvn clean package
```

Este genera (~/sentilo-master/sentilo-catalog-web/target) el war de la Web Application Catalog, que es la interfaz web de la herramienta. Para acceder a esta plataforma solo tenemos que poner la url http://ip_server:8080/sentilo-catalog-web/

Ilustración 14: Página de inicio de Sentilo sobre el servidor virtual

Para generar el **Subscription/publication server**, que será el que nos active la API de la plataforma con la que podremos insertar observaciones, tenemos que ejecutar el goal:

```
~/sentilo-master/sentilo-platform/sentilo-platform-server# mvn package
appassembler:assemble -P dev
```

Con esto se ha generado una estructura de directorios en:

```
~/sentilo-master/sentilo-platform/sentilo-platform-server/target/appassembler# ls -l
drwxr-xr-x 2 root root 4096 Feb 13 21:41 bin
drwxr-xr-x 9 root root 4096 Feb 13 21:41 repo
```

Tenemos que copiar estos dos directorios en el path en el que queremos tener instalados los servicios que comprende la plataforma. Desde /bin tenemos el script sentilo-server que es el que arranca el servidor descrito cuando lo ejecutamos. Para que la ejecución se mantenga independiente de la sesión en la que se ha ejecutado el script, hay que ejecutar el siguiente comando:

```
/opt/sentilo-server/bin# nohup ./sentilo-server > foo.out 2> foo.err < /dev/null &
[1] 14374    → este es el PID del proceso que queda arrancado
```

Finalizado esto podemos hacer un test para comprobar el funcionamiento de todos los servicios instalados anteriormente y el **Subscription/publication server**

```
root@vps27430:~/sentilo-master/scripts# ./testServerStatus.sh
1111
```

Hecho todo esto la parte fundamental de la plataforma ya está en funcionamiento y el API de Sentilo está escuchando en el puerto 8081 que es donde tenemos que enviar las observaciones de los móviles. Sobre la plataforma tenemos que registrar una serie de elementos que almacenarán la información del proyecto:

- **Providers**: es el elemento principal sobre el que después añadiremos componentes y sensores. Cada "providers" tendrá una serie de sensores asociados y una **clave de identificación** que tendrán que contener las peticiones de inserción de observaciones sobre dicho proveedor. Para este proyecto hemos añadido uno llamado **proyecto** que solo contendrá un sensor llamado ruido.
- **Componente**: los componentes aglutinan varios sensores y una de sus características consiste en si son estáticos o móviles. Los estáticos tienen representación sobre el mapa del que viene provista la plataforma (siempre que se especifiquen las coordenadas) y la representación de los datos enviados por dispositivos móviles está en desarrollo en la actualidad.
- **Sensor**: son los que caracterizan el tipo de las observaciones que recoge la plataforma. Dentro de sus características podemos destacar: nombre, tipo de acceso (público o privado), tipo de sensor (humedad, temperatura, ruido, etc), tipo de datos que almacena, unidad de medida, etc. También hay que especificar a que Providers y Components pertenece. Cada sensor tiene un identificador dentro de Sentilo que identifica las observaciones que le pertenecen del resto de sensores.

Estos elementos definen la URL sobre la que tenemos que enviar los datos ya que tenemos por un lado el servidor, por otro el servicio (data) y por otro el proveedor y los sensores:

http://www.smartcitything.es:8081/data/proyecto

A la plataforma podemos enviar sensores fijos (latitud y longitud dada y fija) y sensores móviles en los que la información de localización geográfica va en la observación. Sentilo solo representa en un mapa los valores de los sensores fijos por lo que la representación de los sensores móviles la desarrollaremos durante el presente proyecto.

Todos los datos enviados a la plataforma, procedentes de sensores móviles, se irán almacenando en el sensor correspondiente cuyo nombre está incluido en el Json enviado. De esta forma un mismo proveedor puede tener varios sensores. La plataforma muestra gráficamente los últimos datos enviados para cada uno de los sensores móviles como se puede ver en la siguiente imagen:

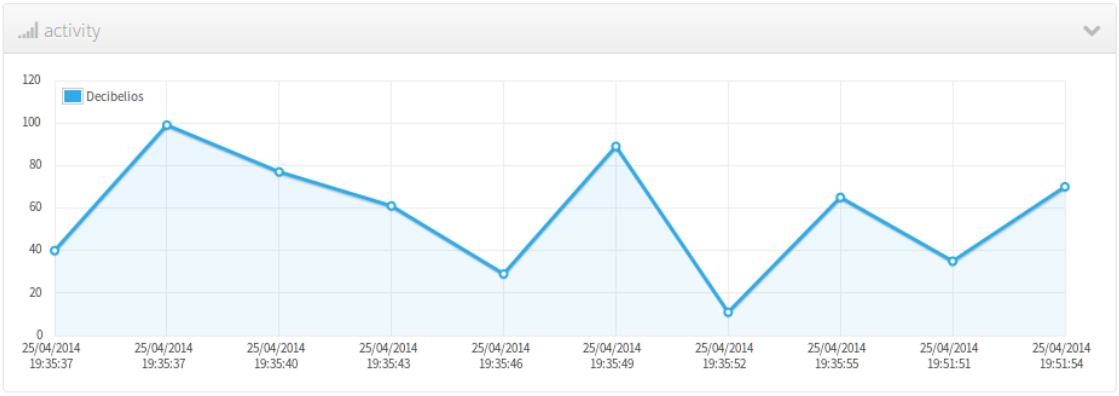


Ilustración 15: Representación gráfica de las observaciones

Todos los datos que se envían a la plataforma se almacenan en memoria principal utilizando el gestor Redis que tiene varios tipos de estructuras para almacenar información. En la implementación de Sentilo se han utilizado dos tipos:

- **Sorted Sets:** es un tipo de estructura compuesta por un conjunto de String ordenados. Los comandos para tratar con esta estructura comienzan por Z, por ejemplo con ZADD podemos añadir un elemento a la pila. Cada sensor está representado por un Sorted Sets que contiene la colección de identificadores de cada observación almacenados junto a la marca temporal de recepción. Estos son los apuntadores a las observaciones reales que están almacenadas en un hash.
- **Hash**: este formato permite almacenar objetos identificados mediante un string que se relacionan con un conjunto de cualidades identificadas por su nombre y su valor. Con los comandos HMSET podemos agregar dichos campos y con HGET (ALL) podemos ver directamente las observaciones que tiene el servidor. En este formato se almacena toda la información de cada una de las observaciones compuesta por:
  - **Location**: contiene la longitud y latitud de la medida separadas por un espacio.
  - **Sid**: es el identificador que diferencia unos sensores de otros, para dos observaciones del mismo sensor este valor es el mismo.
  - **Data**: es el valor de la medida o parámetro que viene en la observación. En este proyecto se envían medidas de ruido y medidas de velocidad.
  - **Ts**: es la marca de tiempo expresada en microsegundos desde 1970. Si la observación no trae este campo la plataforma la añade poniendo el momento en que ha recibido dicha observación.

Si activamos el agente ***Relational Agent*** tendremos que además de almacenar la información en Redis también se almacenará en MySQL en una de las tablas creadas para tal efecto. Mientras que no tengamos dicho agente activo, las

observaciones solo son almacenadas en Redis. Para activarlo tenemos que realizar una operación similar al artefacto anterior. Desde el directorio ***sentilo-agent-relational*** ejecutamos el goal ***mvn package appassembler:assemble -P dev*** y trasladamos, como en el caso anterior los directorios /bin y /repo a donde queramos montar dicho artefacto. En este caso también está en /opt/sentilo-agent-relational-server. Para arrancar la ejecución del agent-relational solo tenemos que ejecutar el script /bin/sentilo-agent-relational-server y ya tendremos que las observaciones son enviadas a MySQL también.

Hay que tener en cuenta que en la actualidad solo se almacena el proveedor, el sensor y las observaciones de dicho sensor pero **no** se almacena la localización geográfica. Esta funcionalidad está en la actualidad en desarrollo.

9 MAPAS DE OBSERVACIONES

Sentilo representa mediante mapas las observaciones obtenidas desde dispositivos fijos. Para este proyecto es necesario publicar las observaciones de dispositivos móviles en dichos mapas. Por tanto hay que implementar tanto los procesos de extracción de los datos desde Redis como su publicación gráfica.

9.1 Extracción de las observaciones desde Redis.

Para la representación de las observaciones en un mapa utilizaremos un método productor/consumidor. Las observaciones están en la plataforma almacenadas utilizando Redis por lo que necesitamos un proceso que extraiga las que nos interesen y las ponga a disposición de otro proceso que las represente.

Sentilo almacena la información de los sensores utilizando dos estructuras de Redis: Sorted Set y Hash. Cada sensor tiene asociado un Sorted Set identificado por un “sid” de forma única, de tal forma que si tenemos un proveedor llamado “proyecto” al que pertenece un sensor llamado “ruido” podemos obtener su identificador con:

```
Redis 127.0.0.1:6379> get sensor:proyecto:ruido:sid
"1"
```

Dentro de este Sorted Set tenemos almacenados todos los identificadores de las observaciones relacionadas con el sensor. Si queremos borrar dicha colección (del sid:1:observations) tenemos que tener en cuenta que las observaciones seguirán estando ya que solo borraremos las referencias a ellas en el Sorted Set. Con el siguiente comando podemos obtener el número total de observaciones asociadas a un sensor:

```
Redis 127.0.0.1:6379> zcard sid:1:observations
(integer) 25524
```

Para ver todas las entradas de un Sorted Set incluyendo la marca de tiempo (score) de dichas medidas, podemos ejecutar:

```
zrange sid:1:observations 0 -1 WITHSCORES
……..
54599) "27549"
54600) "1398597869410"
54601) "27550"
54602) "1398597870330"
…..
```

Dicho comando nos muestra una lista de los identificadores de todas las observaciones relacionadas con este Sorted Set y el score de cada una de ellas. En este caso, es el timestamp que coincide con el ts de la observación que contiene la información de geolocalización y el parámetro enviado. Si queremos borrar[33] las observaciones realizadas entre dos tiempos podemos utilizar:

```
ZREMRANGEBYSCORE sid:1:observations 1398597874210 1398597877210
```

Si lo que queremos es seleccionar las observaciones relacionadas con un sensor en concreto incluidas en un intervalo de tiempo podemos utilizar:

```
ZRANGEBYSCORE sid:1:observations 1398722357364 1399109837364
```

Una vez que tenemos seleccionados todos los identificadores de las observaciones que nos interesan podemos extraer la información almacenada en cada una de ellas con

```
Redis 127.0.0.1:6379> hgetall sdid:21006
1) "sid"
2) "1"
3) "data"
4) "42"
5) "ts"
6) "1395686300733"
7) "location"
8) "40.3881405 -3.7686558"
```

O si queremos tener solo un campo concreto de la observación para después incluir su valor en una variable podemos utilizar:

```
Redis 127.0.0.1:6379> hget sdid:27000 location
"42.5899954 -5.57659152"
```

33 Realmente la observación no se borra, solo se desvincula del Sorted Set.

Usando las estructuras anteriores podemos implementar un proceso que ponga a disposición de la plataforma de mapas la información contenida en Redis. El proceso de extracción de todas las observaciones que nos interesan se puede hacer de dos formas:

- **Ficheros**: podemos extraer y almacenar la información en formato GeoJson sobre un fichero que se genere mediante un programa en Java ejecutado cada cierto tiempo. Dicho fichero se puede publicar con una url específica para ser cargado en un mapa web que represente dichas medidas. La ventaja de este método es que se optimiza el rendimiento del servidor ya que cada cliente que acceda a dichos datos solo descarga un fichero estático. El inconveniente es que el mapa no es en tiempo real. Este método será adecuado para la generación de mapas históricos y por tanto estáticos. Estos ficheros serán públicos.
- **Servicio Web**: podemos publicar la información en un servicio REST en formato GeoJson. Para ello tenemos que implementar un servlet que se ejecute sobre el contenedor de aplicaciones Tomcat7 que tenemos para la plataforma. Como ventaja tenemos que los datos representados en los mapas podrían ser en tiempo real y además se puede adaptar la petición a las necesidades del usuario sin más que enviar los parámetros adecuados en el método get. Utilizando este método la información está accesible para cualquier aplicación web cliente externa al proyecto que es uno de los requisitos iniciales. Como inconveniente tenemos que cada petición de un usuario genera un proceso que puede sobrecargar al servidor.
- También se puede utilizar una aplicación web que extraiga los datos directamente de Redis sin publicarlos en forma de servicio web. Esto impediría que se pudieran programar clientes web para procesar la información de sensores. Esta solución será la única que no se implementará.

En este proyecto contemplamos el uso de las dos alternativas generando ficheros estáticos que además sirvan de copia de seguridad y un servicio web para representar las medidas en tiempo real y adaptarse a las peticiones de los usuarios. Vamos a utilizar como lenguaje de programación Java ya que es el utilizado en la app.

Como los datos están almacenados en Redis necesitamos una librería que nos ayude a manejar dicho gestor de base de datos. En la web oficial de Redis aconsejan una serie de clientes entre los que se encuentra Jedis[34].

Jedis es un proyecto de código abierto publicado en GitHub[35]. Para utilizarlo hay que generar el .jar para incluirlo en nuestro servicio. Descargamos el código[36], lo compilamos y lo empaquetamos en forma de jar. Para este proceso solo tenemos que utilizar **Gradle**[37] como se muestra a continuación

```
chmod +x gradlew
./gradlew build
./gradlew jar
```

34 http://Redis.io/clients
35 https://github.com/xetorthio/jedis
36 https://github.com/xetorthio/jedis
37 http://www.gradle.org/

También tenemos que incluir el Commons Pool de Apache[38] y la librería Json.simple[39] para empaquetar datos en formato Json como hicimos en la app. Una vez realizado todo esto para la conexión al gestor Redis tenemos que incluir la dirección del servidor y la autenticación en nuestro servlet:

```
Jedis jedis = new Jedis("www.smartcitything.es");
jedis.auth("sentilo");
jedis.dbSize();
```

Para seleccionar las observaciones de cada sensor podemos crear una lista con los identificadores del sensor concreto con la siguiente sentencia Java:

```
Set<String> sdid = jedis.zrange("sid:2:observations", 0, -1);
```

También podemos generar esta lista con todos aquellos sensores comprendidos entre dos marcas de tiempo. Por ejemplo, una que nos da el momento actual (ahora) y otra que nos da la marca temporal 24 horas antes (en ambos casos el tiempo se expresa en microsengundos desde 1970)

```
Set<String> sdid = jedis.zrangeByScore("sid:2:observations", un_dia_menos, ahora);
```

Estas estructuras servirán para implementar las aplicaciones que generarán los ficheros estáticos y el servicio web.

9.2 Generación de ficheros históricos.

Para generar los ficheros vamos a desarrollar dos clases de Java usando la librería clásica de Java Java.io. Cada una de estas clases extraerá todas las observaciones de cada uno de los sensores: velocidad y ruido.

Cuando tengamos los ficheros generados podemos optar por borrar las observaciones y la referencia de cada una de ellas en el Sorted Set o dejarlas en la plataforma. Para borrar podríamos utilizar las siguientes sentencias Java:

```
jedis.zrem("sid:2:observations", identificador-observación); → Borra las referencias
jedis.del(sdid:identificador-observación); → Borra las observaciones
```

Para optimizar recursos del servidor podemos borrar todas las observaciones una vez al mes e ir almacenando los ficheros como históricos mensuales.

38 http://commons.apache.org/proper/commons-pool/download_pool.cgi
39 http://code.google.com/p/Json-simple/

Una vez terminados los programas tenemos que importarlos al servidor y programar la ejecución del los mismos cada 24 horas. Esto lo podemos realizar con el cron del propio sistema o utilizando el proyecto quartz-scheduler[40] para programar la tarea.

Para importar los ejecutables al servidor empaquetaremos cada una de las clases compiladas en formato jar (ejecutable) con las librerías necesarias. Para ello tenemos que crear un formato similar al mostrado a continuación:

```
drwxrwxr-x 3 esteban esteban   4096 may 11 18:58 META-INF
drwxrwxr-x 3 esteban esteban   4096 may 11 14:18 org
drwxrwxr-x 3 esteban esteban   4096 may 11 14:18 Redis
-rw-rw-r-- 1 esteban esteban   4029 may 11 10:59 Speed_all.class
-rw-rw-r-- 1 esteban esteban 324281 may 11 14:20 Speed_all.jar
```

En el directorio META-INF se almacena la información de cuál es la clase que contiene el método main y dónde están las clases invocadas por ella. En el directorio Redis está todo lo relacionado con el cliente Jedis que nos da acceso al gestor Redis y en el directorio org están todas las clases relacionadas con los objetos Json. Todo esto lo podemos empaquetar con extensión .jar y queda el Speed_all.jar que llevaremos al directorio /opt/mapas junto al resto de script de Sentilo.

9.3 Servicio Web

Uno de los requisitos del proyecto es el de ofrecer toda la información de forma pública y fácil. La forma de acceder a las observaciones será mediante mapas y también se podrá acceder mediante el servicio web en formato GeoJson.

Para la generación del servicio web recurriremos a un servlet de Java que desplegaremos en formato .war sobre Tomcat. El servicio web dará una respuesta en tiempo real y podrá adaptarse a las peticiones del usuario ya que puede recoger parámetros procedentes de la petición del cliente http (método get).

La clase a utilizar será una extensión de un servlet (public class Location extends HttpServlet) que tendrá un método con dos objetos:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

El objeto ***request*** capturará los parámetros que proceden de la petición http utilizando el request.getParameter ("nombre-parámetro") y gracias a dicho parámetro podemos realizar una búsqueda de observaciones adaptada a las peticiones del usuario.

Con el objeto ***response*** vamos a enviar la respuesta formateada usando GeoJson (objstring) para cargar los datos sobre el mapa. Gracias a este objeto también podemos cargar las cabeceras necesarias para usar la API de Google como se explica en el punto de generación de mapas.

40 http://quartz-scheduler.org/

```
response.setHeader("Access-Control-Allow-Origin", "*");
response.setHeader("Access-Control-Request-Method", "GET,POST");
PrintWriter out = response.getWriter();
out.println(objstring);
```

En concreto, el servicio web acepta dos parámetros: ***horas*** y ***sensor***. El parámetro horas es el que determina cómo de grande será el intervalo temporal para generar la respuesta y sensor nos indicará el identificador del sensor que queremos ver: 2 o 1. La url que tenemos que enviar para que nos muestre las observaciones de las últimas cuatro horas de las medidas de velocidad sería el siguiente:

```
www.smartcitything.es:8080/ServiceGeoJson/Sensor?horas=4&sensor=2
```

A parte de estos parámetros sería muy sencillo implementar otras funciones como extraer las observaciones en un intervalo de tiempo dado. Esto sería interesante de cara a la observación, por ejemplo, de la fluidez del tráfico a lo largo de la misma hora en todos los días de la semana.

La estructura del servicio implementado es la siguiente:

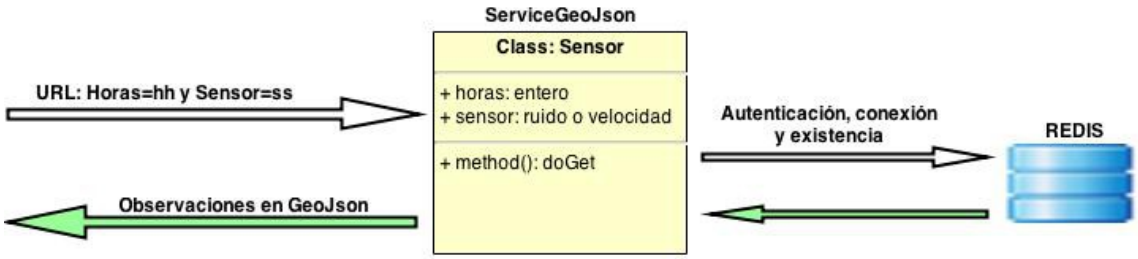


Ilustración 16: Diagrama ServiceGeoJson

Los test unitarios que se pueden realizar sobre **ServiceGeoJson** se pueden dividir entre pruebas sobre los parámetros de entrada y pruebas sobre el acceso al gestor Redis. Las pruebas sobre los parámetros se realizan fácilmente ya que podemos ir modificando la url enviada al servicio web y comprobar los resultados. Las pruebas realizadas están relacionadas con estos posibles errores en la URL:

- Sobre la URL enviada por el cliente puede haber varios problemas:
  - El servicio no existe o no está especificado.
  - La variable horas o la variable sensor no están incluidas.
  - Alguna de estas variables no es un entero positivo (letras, negativos, etc).
  - Todo está bien codificado pero no existe dicho sensor. En este caso no se enviarán observaciones.

- Sobre el servidor podemos tener varios problemas pero en todos los casos el cliente solo podría contactar con el administrador para notificar dicho error.
  - El servicio web tiene un problema para autenticarse sobre Redis.
  - El servicio Redis está caído por lo que no se puede realizar la conexión.
- Relacionado con el formato de la url puede suceder que esté mal formada.

Estos errores serán manejados por las siguientes excepciones:

```
ServletException, IOException, NumberFormatException, JedisConnectionException,
JedisDataException, MalformedURLException
```

Como el servicio puede ser usado por cualquier aplicación web tenemos que implementar un conjunto de respuestas a petición con errores sobre la URL enviada o posibles defectos de funcionamiento de la plataforma. Este conjunto de respuestas puede hacerse de forma muy variada y con distintos modelos de implementación[41]. Entre estas posibilidades podemos escoger enviar el mensaje de error en formato Json (response.setContentType("application/Json")), enviar una web en html identificando el error (response.setContentType("text/html")) o utilizando los códigos de error del protocolo html (response.sendError(response.SC_BAD_REQUEST, "mensaje")).

En formato Json cada tipo de error mencionado tendría una respuesta diferente:

- En la URL enviada falta alguno (o los dos) de los parámetros necesarios: horas y sensor:

```
http://localhost:8080/ServiceGeoJson/Sensor
{ERROR:"Java.lang.NumberFormatException: null" ,COMENTARIO:"El sensor o las
horas son nulos"}
```

- Alguno de los parámetros enviados tienen un formato defectuoso o es negativo:

```
http://localhost:8080/ServiceGeoJson/Sensor?horas=10&sensor=a
{ERROR:"Java.lang.NumberFormatException: For input string: "a""
,COMENTARIO:"El tipo de dato del sensor o las horas no son un entero"}
```

- Contraseña de autenticación del servicio web sobre Redis errónea:

```
{ERROR:"Redis.clients.jedis.exceptions.JedisDataException: ERR invalid password"
,COMENTARIO:"Problema con la autenticación: Contactar con el adminsitrador"}
```

- Servicio Redis parado:

41 http://martinfowler.com/articles/richardsonMaturityModel.html

{ERROR:"Redis.clients.jedis.exceptions.JedisConnectionException:Java.net.ConnectException: Connection refused" ,COMENTARIO:"El gestor Redis no responde: Contactar con el adminsitrador"}

Si utilizamos los códigos de error del protocolo http obtendríamos mensajes similares pero en formato html. En la siguiente imagen podemos ver el mensaje obtenido para una URL en la que el parámetro "sensor" enviado es una "a":

Ilustración 17: Mensaje de error con códigos http

Esta segunda opción es la utilizada para el proyecto. Los posibles códigos de error que se pueden obtener dependiendo de cada una de las situaciones son los siguientes:

- **200**: todo ha ido correctamente.
- **202**: petición aceptada pero no se han encontrado observaciones.
- **400**: falta alguna de las variables (horas o sensor) o alguna tiene un formato defectuoso.
- **404**: el servicio web especificado no está disponible en el servidor.
- **401**: hay un fallo de autorización en el servicio web sobre el sistema gestor Redis.
- **503**: el gestor Redis está caído o parado.

Para la realización de pruebas de forma sistemática y automatizada emplearemos Junit, que permite programar un conjunto de pruebas que se ejecutan de forma fácil en cada una de las compilaciones.

Para efectuar las pruebas generamos otro proyecto Test con una clase llamada ***sensorTest*** y seis métodos para probar todos los posibles errores en la formación de la URL enviada por el cliente. Estos métodos no solo envían la petición errónea si no que además comprueban si el código de error enviado es el correcto. Los métodos implementados se pueden ver en la siguiente imagen:

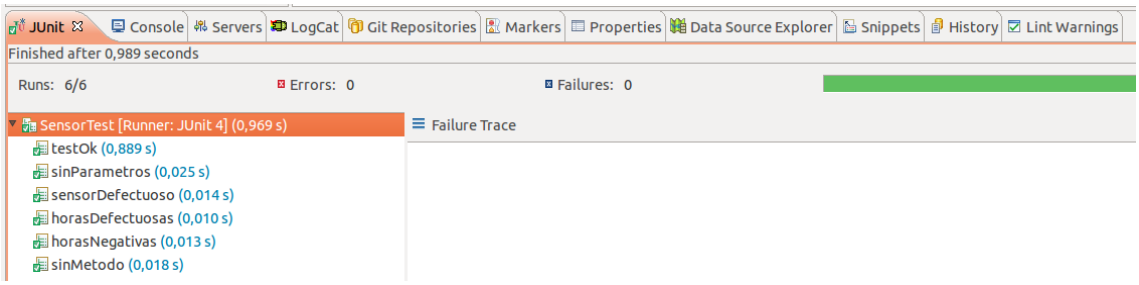


Ilustración 18: Métodos de JUnit

Con estas pruebas no se comprobarán los problemas derivados de un mal funcionamiento del servicio web o por tener los servicios parados, ya que hay que modificar o parar el servicio web y no es operativo si queremos hacer las pruebas en cada compilación.

9.4 Mapas web

Para la representación de las observaciones de ruido y velocidad sobre un mapa web nos serviremos de la API de JavaScript que nos ofrece Google[42]. Con ella tenemos la opción de importar sobre un mapa genérico al que se le pueden ir superponiendo capas con datos en formato GeoJson[43]. Dicho formato es un estándar abierto para la codificación de estructuras geográficas simples (puntos, líneas, polígonos, etc) y valores de parámetros relacionados con ellas. Una estructura GeoJson es siempre un objeto que cumple las especificaciones del formato Json.

Una condición que tenemos que cumplir para poder utilizar la API de Google es que el servidor que contiene la información en formato GeoJson envíe un permiso explícito en la cabecera de los paquetes http. Este requisito se satisface de forma diferente si los datos proceden de los ficheros históricos servidos por Apache que si procede del servicio web ofrecido por Tomcat.

En el caso de Apache, tenemos que añadir una directiva en el fichero de configuración del VirtualHost que ofrece dichos ficheros históricos como se puede ver a continuación:

```
<Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
        Header set Access-Control-Allow-Origin "*"
    </Directory>
```

En el caso del servicio web podemos incluirlo de forma genérica en la configuración de Tomcat o específicamente en cada uno de los servicios desplegados en él. Para incluirlo directamente sobre ServiceGeoJson añadimos una cabecera al objeto ***response*** como se muestra a continuación:

42 https://developers.google.com/maps/documentation/Javascript/tutorial
43 http://GeoJson.org/

```
response.setHeader("Access-Control-Allow-Origin", "*");
response.setHeader("Access-Control-Request-Method", "GET,POST");
```

Una vez hecho esto, solo hay que diseñar la página web, e incluir el código JavaScript que carga el mapa y sobre éste cargar la capa con las observaciones procedentes de un fichero o de la dirección del servicio web. Para cargar un mapa genérico especificando el zoom y el centro del mapa y después añadir una capa con las observaciones procedentes de uno de los ficheros podemos usar las siguientes directivas:

```
map = new google.maps.Map(document.getElementById('map-canvas'), {
   zoom: 15,
   center:{lat: 40.3820, lng: -3.7636}
 });
map.data.loadGeoJson('http://www.smartcitything.es/speed.Json'); → Observaciones
```

En el caso de las observaciones procedentes del servicio web en el map.data.load habría que incluir la url siguiente:

```
http://www.smartcitything.es:8080/ServiceGeoJson/Sensor?horas=24&sensor=1
```

Cada observación estará representada por un icono cuyo color dependerá del valor del parámetro recibido, en este caso la velocidad de desplazamiento del móvil. El resultado se puede ver en la siguiente imagen en la que se representa la velocidad del móvil expresada en Km/h, el valor numérico se puede ver en uno de los puntos. También se puede observar la aceleración en el cambio de color del verde, más lento, al rojo, con mayor velocidad.

Ilustración 19: Mapa de observaciones de ruido

Las observaciones se pueden tomar utilizando tres métodos como se ha indicado anteriormente, y cada uno de estos métodos suele estar en un rango de precisión dependiendo de la ubicación en que nos encontremos:

- Antenas de telefonía (proveedor network): nos suele dar una precisión de unos cientos de metros a algún kilómetro. Por tanto este proveedor solo se podría utilizar para el envío de parámetros que no requieran precisión.
- Antenas Wi-Fi (proveedor network): cuando nos encontramos en ubicaciones muy pobladas la precisión suele ser de varias decenas de metros pero dependemos del número de puntos Wi-Fi que haya alrededor. Aunque la precisión suele parecer similar al GPS, las coordenadas dadas con este método suelen representar mucho peor la realidad como se puede ver en la imagen anterior.
- GPS: en este caso el valor numérico de la precisión suele coincidir con el valor del proveedor anterior pero finalmente representa de una forma más fidedigna la situación real del dispositivo. Este sensor necesita estar en espacios abiertos en los que tenga visibilidad directa con los satélites. En la siguiente imagen se puede comprobar cómo las observaciones están correctamente centradas sobre la línea de desplazamiento del dispositivo.

En la imagen podemos ver las observaciones tomadas utilizando como proveedor el GPS.

Ilustración 20: Mapa de observaciones procedentes de GPS

En la siguiente, tenemos las observaciones tomadas utilizando como proveedor el NETWORK con la Wi-Fi activada. Podemos ver como el GPS representa con mayor precisión el recorrido ya que los puntos se ajustan más a la calle. En el segundo caso se ve claramente cómo en algunas circunstancias los puntos pueden tener un desplazamiento irreal.

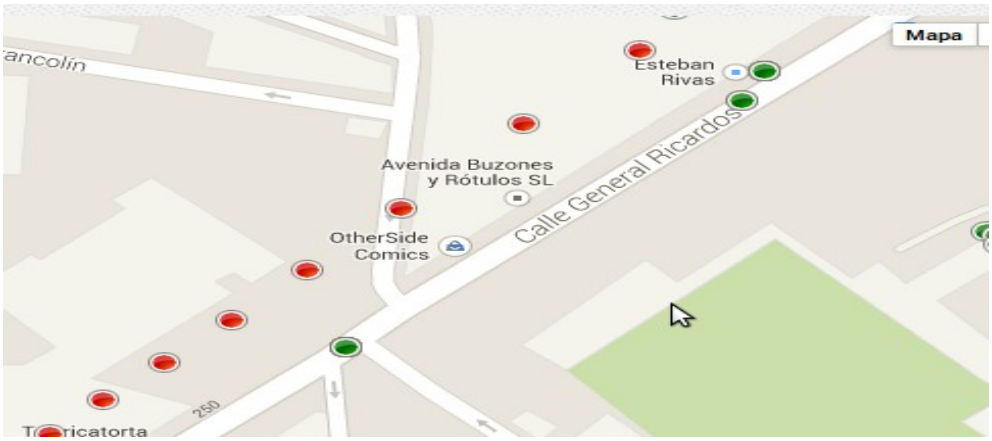


Ilustración 21: Mapa de observaciones procedentes del proveedor NETWORK

10 BURNDOWN CHART Y RETROSPECTIVA

10.1 Sprint 1

En la "Reunión de planificación del Sprint1" el equipo completo seleccionó los requisitos del Product Backlog que iban a ser atendidos. Una de las dificultades encontradas fue el acceso a la información sobre las aplicaciones del Ayuntamiento y uso de mapas por lo que se llegó a los siguientes acuerdos:

- Posponer para el tercer Sprint el análisis de la situación actual de las técnicas utilizadas por el Ayuntamiento para publicar los datos almacenados en la PSAB[44].
- Posponer también el análisis, diseño y generación de los mapas de ruido.
- Incluir el análisis y montaje del entorno de desarrollo de software perteneciente al proyecto.

Durante el Sprint1 se han completado las tareas 0, 1, 2 y 3 del proyecto con un computo total de 160 horas en 86 días. Por tanto quedan pendientes las tareas de la 4 a la 7 para completar en los dos siguientes sprint. Finalmente se comprobó que este sprint cumplía con los requisitos establecidos para esta etapa.

En la siguiente gráfica se puede el ritmo de evolución del cumplimiento de las tareas. Se puede ver como el ritmo inicial no era el deseado por las dificultades encontradas.

44 Plataforma de Sensores y Actuadores de Barcelona.

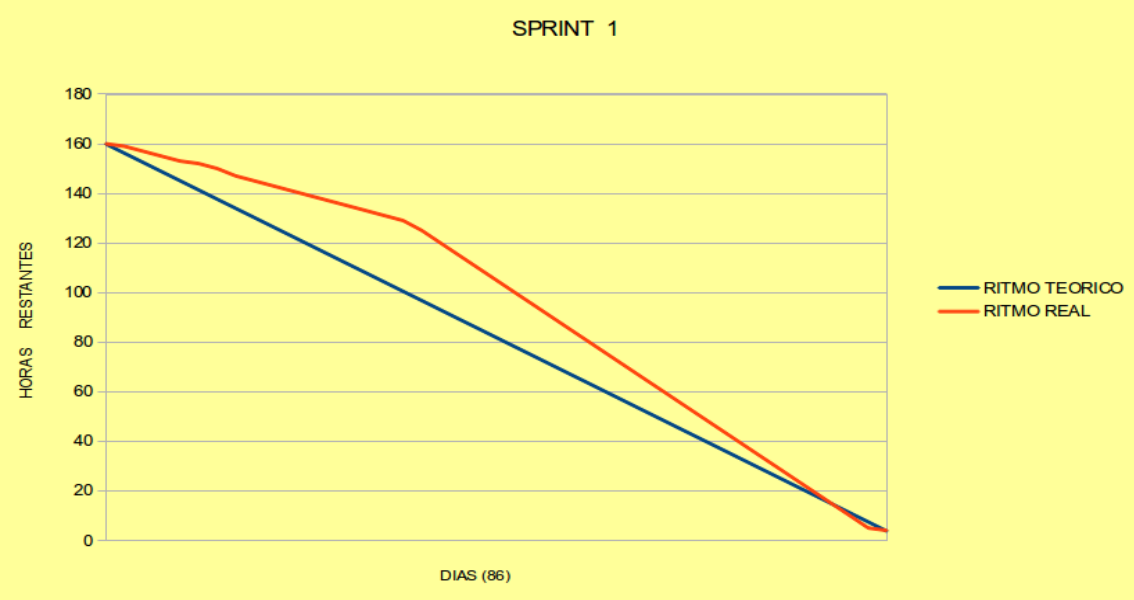


Ilustración 22: Horas consumidas hasta el momento

10.2 Sprint 2

Durante el Sprint2 se han tenido que realizar algunas modificaciones que afectan tanto a las tareas originales como a los objetivos del proyecto. El equipo Scrum ha decidido que para realizar las pruebas de envío de observaciones desde el móvil se montara una plataforma independiente de la del Ayuntamiento. Inicialmente se iba a utilizar la plataforma PSAB ya montada y en funcionamiento.

El software elegido para montarla es el mismo que se utiliza en la PSAB (Sentilo). Para hacer el ajuste de horas las funcionalidades de Noise se han reducido, de tal forma que las medidas de ruido enviadas son ficticias.

Además del incremento de horas que supone el montaje de la plataforma hay que sumar los problemas derivados de la instalación de Sentilo debido a algunos errores en la documentación de la herramienta que fue lanzada pocos días antes de la instalación en este proyecto. Estos errores fueron subsanados por el equipo de soporte de Sentilo una vez advertidos.

Al finalizar este Sprint se han realizado las tareas 4 y 5 teniendo en cuenta las modificaciones descritas anteriormente. El total dedicado en este sprint, sin tener en cuenta el tiempo perdido por dichos errores, es de 150 horas durante un periodo de 83 días. En la siguiente gráfica se puede observar el ritmo de evolución y cómo hubo un ligero retraso durante el periodo de Navidades.

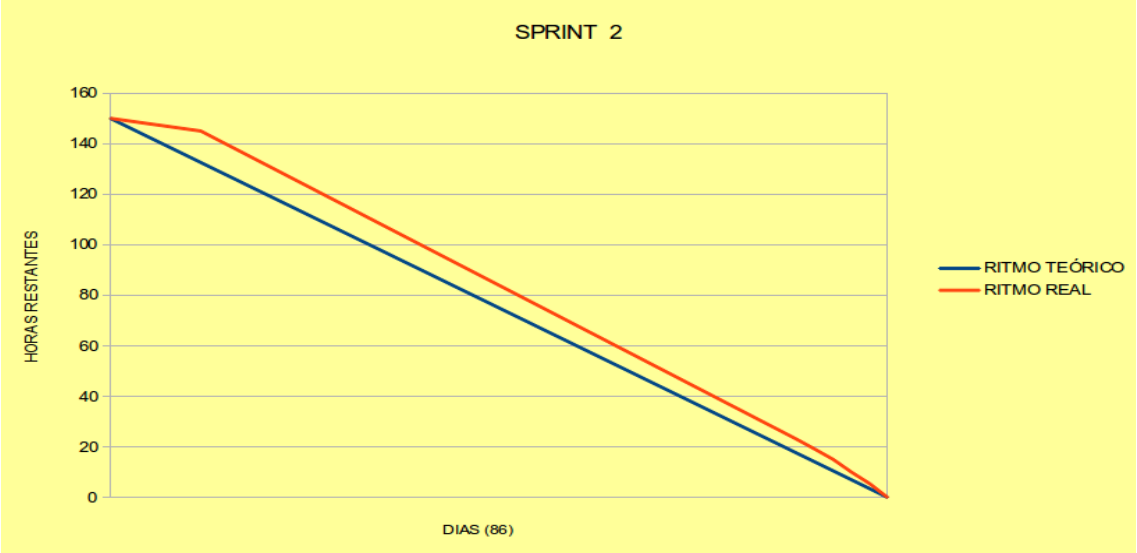


Ilustración 23: Gráfica Sprint 2

10.3 Sprint 3

Durante el tercer Sprint se han completado las tareas 6 y 7 que incluían la generación de mapas web para publicar la información y todas las adaptaciones y mejoras posteriores a las pruebas de integración sobre las app's, el servicio web, la generación de ficheros y el desarrollo de la web del proyecto. La dedicación ha sido de 150 horas en un total de 83 días. La línea horizontal corresponde con Semana Santa, periodo durante el cual no se avanzó en la elaboración del proyecto.

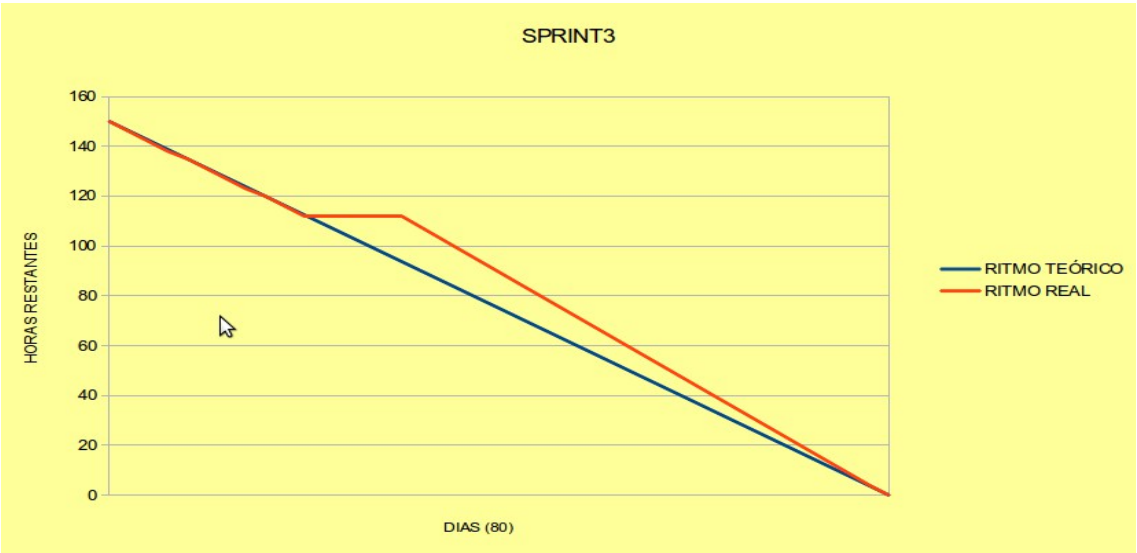


Ilustración 24: Gráfica Sprint 3

11 INDICE DE IMÁGENES

12 REFERENCIAS Y BIBLIOGRAFÍA

- [1] Mitchell, W. J. (2007). Ciudades inteligentes. *UOC Papers: revista sobre la sociedad del conocimiento*, (5), 1.
- [2] Haubensak, O. (2011). Smart cities and internet of things. In *Business Aspects of the Internet of Things, Seminar of Advanced Topics, ETH Zurich* (pp. 33-39).

- [3] Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *Software, IEEE*, *17*(4), 26-32.
- [4] Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, *2*(2), 115-150
- [5] Gómez Bertoli, D. (2012). Estudio, implementación y análisis de métodos de trilateración para la localización de usuarios desde sus terminales móviles.
- [6] Yilin Zhao. Standarization of mobile phone positioning for 3G systems. IEEE Communications Magazine, 40:108–116, Jul 2002.
- [7] Parlamento Europeo, Directiva 2003/10/CE, de 6 de febrero de 2003, sobre las disposiciones mínimas de seguridad y de salud relativas a la exposición de los trabajadores a los riesgos derivados de los agentes físicos (ruido). DOUE n.° L 042 de 15-02-2003 p. 38-44 [21-1-2008]

  Jefatura del Estado Ley 37/2003, de 17 de noviembre, del Ruido, BOE n.º 276 de 18-11-2003, España [20-1-2008]

  Ministerio de la Presidencia, Real Decreto 286/2006, de 10 de marzo, sobre la protección de la salud y la seguridad de los trabajadores contra los riesgos relacionados con la exposición al ruido,
- [8] Ley 16/2002 de la Generalitat de Catalunya, de 28 de junio, de protección contra la contaminación acústica

  Ordenança municipal tipus, reguladora del soroll i les vibracions Diari Oficial de la Generalitat de Catalunya. Núm. 2126 - 10.11.1995