



UNIVERSIDAD DE ZARAGOZA
Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

Proyecto de Fin de Carrera
Ingeniería de Telecomunicaciones
Agosto 2011

# VISUALIZACIÓN Y SEGUIMIENTO VÍA WEB DE SENSORES EN MEDIOS ACUÁTICOS

## MEMORIA 1/2

Autor: Bárbara Pérez Felices
Director: Éric Renault
Télécom SudParis
Ponente: Javier Nogueras Iso
Escuela de Ingeniería y Arquitectura


IAAA
Grupo de Sistemas de Información Avanzados

Universidad Zaragoza

# Agradecimientos

En primer lugar quiero dar las gracias a Eric por brindarme la posibilidad de participar en el proyecto Mobesens y por hacerme muy grata mi segunda parte de la estancia en Télécom SudParis.

En segundo lugar, a Javier por su paciencia y buenos consejos.

Por último quiero agradecer a mis padres y a mi hermana el aguantarme y apoyarme incondicionalmente; a todos mis amigos por estar ahí cuando los necesito y a Mau por su positivismo y ánimos constantes.

# RESUMEN

Este proyecto se ha desarrollado en la universidad francesa Telecom SudParis, formando parte del proyecto de la Unión Europea MOBESENS (Mobility for Long Term Water Quality Monitoring) cuyo objetivo es la creación de una infraestructura completa que permita vigilar y monitorizar una zona acuática dada, mediante sensores, con fines medioambientales.

El objetivo del presente proyecto ha sido diseñar, desarrollar e implementar una aplicación de monitorización ergonómica y eficaz a fin de vigilar la actividad de los sensores y las medidas efectuadas por ellos vía web. Ofreciendo la posibilidad al operador de reaccionar frente a los problemas eventuales que puedan sobrevenir en este tipo de medio acuático.

Las líneas principales de trabajo en este proyecto han sido las siguientes:

- Definición de un modelo de datos que permita representar toda la información recogida por los sensores acuáticos. Como formato de intercambio de la información de los sensores se propone la utilización de XML.
- Gestión y almacenamiento de los documentos XML (que guardan la información de los sensores) a través de una base de datos nativa XML que soporta xQuery como lenguaje de consulta.
- Creación de un simulador que genere la información de los sensores y los almacene en la base de datos nativa. En las primeras fases del proyecto MOBESENS no se dispone todavía de todos los dispositivos físicos de los sensores, y se hace necesario simular un entorno real que permita avanzar en el desarrollo de la infraestructura.
- Desarrollo de una aplicación de monitorización, mediante tecnologías web, que llevará a cabo las siguientes funciones: determinación geográfica de la posición de los sensores, así como de su movimiento en tiempo real; especificación de los valores de las medidas realizadas por los sensores; y análisis de dichas medidas mediante un sistema de creación de gráficos históricos.

Para la realización del proyecto, se han utilizado las siguientes herramientas: ICEfaces y SDK Eclipse como entornos de programación para Java y aplicaciones web dinámicas; el API de Google Maps para la integración del dispositivo de localización geográfica de los sensores; el API de Google Charts para la creación del sistema de gráficos; y eXistDB y Tahoe LAFs para la gestión de la base de datos nativa.

# Índice de contenidos

# Índice de figuras

# Capítulo 1. Introducción

## *1.1. Alcance del proyecto y motivación*

Los ambientes acuáticos como los ríos, los lagos o los océanos son lugares frágiles en los que una ínfima variación del medio puede tener consecuencias dramáticas sobre la fauna, la flora o los seres humanos que viven en las proximidades. Así pues, a fin de poder reaccionar eficazmente a los problemas eventuales que puedan sobrevenir en este tipo de medio acuático, la comunidad europea ha favorecido estos últimos años la financiación de diversos proyectos con el objetivo de desarrollar infraestructuras ligadas a la vigilancia y prevención de riesgos ecológicos. Es en este contexto en el que se inició el proyecto FP7 STREP MOBESENS [1].

El proyecto MOBESENS tiene como objetivo el desarrollo de una infraestructura completa que permita vigilar y monitorizar una zona acuática dada. Para ello, el proyecto debe, en particular:

- Desarrollar sensores capaces de almacenar los parámetros físico-químicos del agua (como el pH, la conductividad, la temperatura) ya sea en la superficie o en el fondo.
- Determinar y gestionar la posición de dichos sensores, por una parte a fin de localizar la región en la que se aplican las medidas hechas por el sensor, y por otra parte a fin de desplazar los sensores de forma remota y poder situarlos según las necesidades en las regiones más apropiadas.
- Poner en funcionamiento una red de comunicaciones sin infraestructura previa a fin de enviar las informaciones medidas por los sensores hasta el corazón del sistema.
- Salvaguardar de manera permanente los datos generados por los sensores y proporcionar las herramientas que permitan acceder a dichos datos y tratarlos rápidamente.
- Desarrollar una interfaz de visualización ergonómica y eficaz a fin de vigilar la actividad de los sensores y las medidas efectuadas por ellos, ofreciendo la posibilidad al operador de reaccionar retroactivamente sobre los sensores mediante la consola de visualización.

En la realización del proyecto Mobesens toman parte varias empresas y universidades que trabajan en colaboración. En este marco Telecom SudParis aporta su experiencia en lo que concierne a la salvaguarda perenne y el tratamiento de la información, así como de la interfaz de visualización de los datos.

El presente proyecto ha sido realizado en la universidad Telecom SudParis, dentro del departamento RS2M (Réseaux et Services Multimédia Mobiles) y ha consistido en dar los primeros pasos en la creación de la

aplicación de monitorización que será parte esencial del proyecto Mobesens, ya que permite la interacción de los usuarios finales con el sistema completo.

## 1.2. Antecedentes

Previamente a mi incorporación, el proyecto estaba en general en fase de desarrollo, algunos sensores acabados, otros en fase de diseño o creación. Tanto el hardware (el espacio de almacenamiento es una red de ordenadores), como el software de la base de datos, estaban siendo creados en esos momentos por otros miembros del departamento RS2M. En concreto, la interfaz para poder comunicarse con dicha base de datos (bautizada NetInf por sus creadores) estaba en fase de desarrollo, aunque ya había algunas funciones principales creadas y utilizables.

En cuanto a la interfaz de visualización que consistirá en una aplicación web de monitorización, estaba en estado inicial. Consistía en un mapa creado a través del API de Google Maps en el que se veían algunos sensores y sus medidas en modo estático, ya que no se disponía de datos reales de los sensores en la base de datos y por tanto no se podía acceder a ellos en tiempo cuasi real como es el propósito final de la aplicación. Esta carencia de datos reales ha sido de vital importancia a lo largo del proyecto; de ahí la necesidad de crear un simulador.

## 1.3. Objetivos

El objetivo final de este proyecto es la creación de una aplicación de monitorización y seguimiento vía web de sensores en medios acuáticos. La amplitud del proyecto Mobesens y el hecho de que la aplicación de visualización interacciona con otras partes del sistema desarrolladas por distintos colaboradores, hacen necesaria una fase previa de estudio de la infraestructura global del proyecto que ha sido definida hasta el momento.

También será importante familiarizarse con el entorno de la base de datos, es decir, conocer su funcionamiento y características básicas y como manejar la interfaz (NetInf) que nos permite comunicarnos con ella y aprender los fundamentos del lenguaje de consulta (xQuery) que nos permite solicitar información a la base de datos.

Dado que ya había algunas decisiones tomadas en cuanto a marcos de trabajo se refiere, se deseaba también un acercamiento y familiarización a estos últimos, entre los que se encuentran ICEfaces y el API de Google Maps.

Pasada esta fase de adaptación y amoldamiento, las principales líneas de trabajo que se seguirán a lo largo del proyecto son las siguientes:

- La definición de un modelo de datos que permita representar la información recogida por los sensores acuáticos, mediante el estudio previo de los parámetros y características del agua medidos por ellos.

- La gestión y almacenamiento de la información de los sensores a través de la interfaz NetInf que permite la comunicación con la base de datos nativa XML (la información se almacena como documentos XML) mediante el lenguaje de consulta xQuery.

- La creación de un simulador, realizado en lenguaje Java, que genere y trate la información de los sensores de tal manera que se consiga imitar lo más fielmente posible al sistema final en el que la información se transfiere de los sensores a la base de datos y de ahí a la aplicación. Esto se debe a que en las primeras fases del proyecto Mobesens no se dispone todavía de los dispositivos físicos de los sensores, y se hace necesario simular un entorno real que permita avanzar en el desarrollo de la infraestructura.

- El estudio de las diferentes posibilidades de implementación de cada una de las partes que forman la aplicación de monitorización, con las distintas aplicaciones o lenguajes de programación que conllevan.

- El desarrollo de una aplicación de monitorización, mediante tecnologías web, que llevará a cabo las siguientes funciones: determinación geográfica de la posición de los sensores, así como de su movimiento en tiempo real; especificación de los valores de las medidas realizadas por los sensores; y análisis de dichas medidas mediante un sistema de creación de gráficos en modo histórico y en tiempo real. Se prestará especial atención sobre la interacción, es decir, la interconexión con otras herramientas y/o proyectos de la misma naturaleza.

## *1.4. Materiales y herramientas utilizadas*

A lo largo del proyecto se ha hecho uso de una serie de materiales y herramientas que han sido fundamentales para la realización del mismo. A continuación damos una breve descripción de ellos:

- Eclipse Galileo (v.3.5): Entorno de desarrollo integrado de código abierto multiplataforma para desarrollar "Aplicaciones de Cliente Enriquecido", basadas en el lenguaje de programación Java.
- ICEfaces 1.8: Marco de trabajo de código abierto para construir aplicaciones web con AJAX tipo RIA (Rich Internet Application), que hace que AJAX sea transparente al programador. Usamos el plug-in de ICEfaces para Eclipse.
- Apache Tomcat 6.0: Servidor web, que funciona como contenedor de servlets e implementa las especificaciones de los servlets y de las Java Server Pages (JSP) de Sun Microsystems.
- EXistDB 1.4: Herramienta de administración de bases de datos nativas de código abierto, que hace uso de la tecnología XML y utiliza como lenguaje de consulta xQuery.
- Tahoe-LAFS 1.8.2: Sistema de almacenamiento de datos en nube, gratuito y abierto, que nos permite distribuir los datos en diferentes servidores.
- ActiveMQ 5.3.2: Proveedor del servicio de mensajes de Java (JMS).
- Google Maps API v.2.0: Biblioteca JavaScript de código abierto creada por Google para la integración de mapas en nuestra aplicación.
- Google Charts API V.3.0: Biblioteca JavaScript de código abierto creada por Google para la integración de gráficos en nuestra aplicación.

## *1.5. Estructura de la memoria*

El presente documento está dividido en dos partes bien diferenciadas: la primera es la Memoria del PFC en la que se describe a grandes rasgos el proyecto y la segunda son los Anexos, que contienen información más detallada sobre algunos aspectos del mismo.

La Memoria contiene un breve resumen del proyecto y a su vez está dividida en varios capítulos estructurados de la siguiente forma:

- Capítulo 1, en el que se hace una introducción al tema que se va a tratar en esta memoria.
- Capítulo 2, en el que se describe la arquitectura del sistema completo, así como su funcionamiento.
- Capítulos 3, 4 y 5, en los que se describen detalladamente las partes más importantes del sistema como son la base de datos, el simulador y la aplicación web de monitorización.
- Capítulo 6, en el que se plantean las conclusiones y líneas de trabajo futuro.

Los Anexos son cuatro:

- El Anexo I, correspondiente al análisis y diseño de la aplicación.
- El Anexo II, correspondiente a las tecnologías y herramientas utilizadas, en el que éstas se describen con detalle y se dan razones de su elección a la vista de las ventajas e inconvenientes respecto a otras opciones.
- El Anexo III, correspondiente al Manual de instalación de la aplicación, en el que se enumeran los componentes necesarios para el correcto funcionamiento de la aplicación y se dan las instrucciones necesarias para su instalación.
- El Anexo IV, correspondiente al Manual de usuario, en el que se dan unas breves pautas para la utilización de la aplicación.

# Capítulo 2. Arquitectura del sistema

## 2.1. Visión general

Para una mejor comprensión del presente PFC , debemos tener en cuenta que el proyecto Mobesens se planteó con una duración de tres años y que la realización de éste PFC comenzó cuando sólo había transcurrido la mitad de ese tiempo. Es importante tener esta visión temporal, ya que tiene ciertas implicaciones en cuanto a la arquitectura de la aplicación de monitorización que es el objetivo principal del proyecto. Así pues, hay que diferenciar dos fases en el proyecto Mobesens:

- Fase Inicial: En el período de tiempo en el que se realizó este proyecto, el proyecto Mobesens en general estaba todavía en fase de desarrollo, algunos sensores no estaban terminados, la interfaz de la base de datos no estaba plenamente operativa y no se había realizado todavía ninguna prueba "in situ" con los sensores en el medio acuático. Por tanto, no se disponía de datos reales con los que trabajar en la aplicación de monitorización. Esto nos llevó a la necesidad de crear un simulador que imitara de la forma más real posible al sistema final.

- Fase Final: Entendemos como fase final, aquella en la que los sensores, cuyas informaciones son las que se quieren visualizar vía la aplicación web de monitorización, ya estén posicionados en el entorno acuático a monitorizar y plenamente operativos. En esta fase también estarán terminados el resto de sistemas que afectan a la implementación de la aplicación, como son la red de comunicaciones que nos permite transmitir la información de los sensores a la base de datos y la propia base de datos en sí.

A continuación se va a detallar la arquitectura del sistema en estas dos fases del proyecto Mobesens, haciendo hincapié en la fase inicial ya que es la que se ha realizado a lo largo de este PFC. A su vez se comentarán las diferencias que pueda haber entre las dos arquitecturas.

## 2.2. Situación inicial

En el esquema de la Figura 1, podemos observar la arquitectura del sistema que se ha desarrollado, como fase inicial del proyecto Mobesens en lo que se refiere a la creación de la aplicación de monitorización. Como hemos comentado previamente, en esta situación inicial, no se disponía de datos reales de los sensores y por lo tanto hubo que crear un simulador programado en el lenguaje Java, para suplir esta carencia de datos. Sin embargo, hay que mencionar que se dispuso de una serie de datos históricos de sensores reales proporcionados por uno de los colaboradores del proyecto Mobesens. Estos se introdujeron en la base de datos y es a partir de ellos que se crean los gráficos.

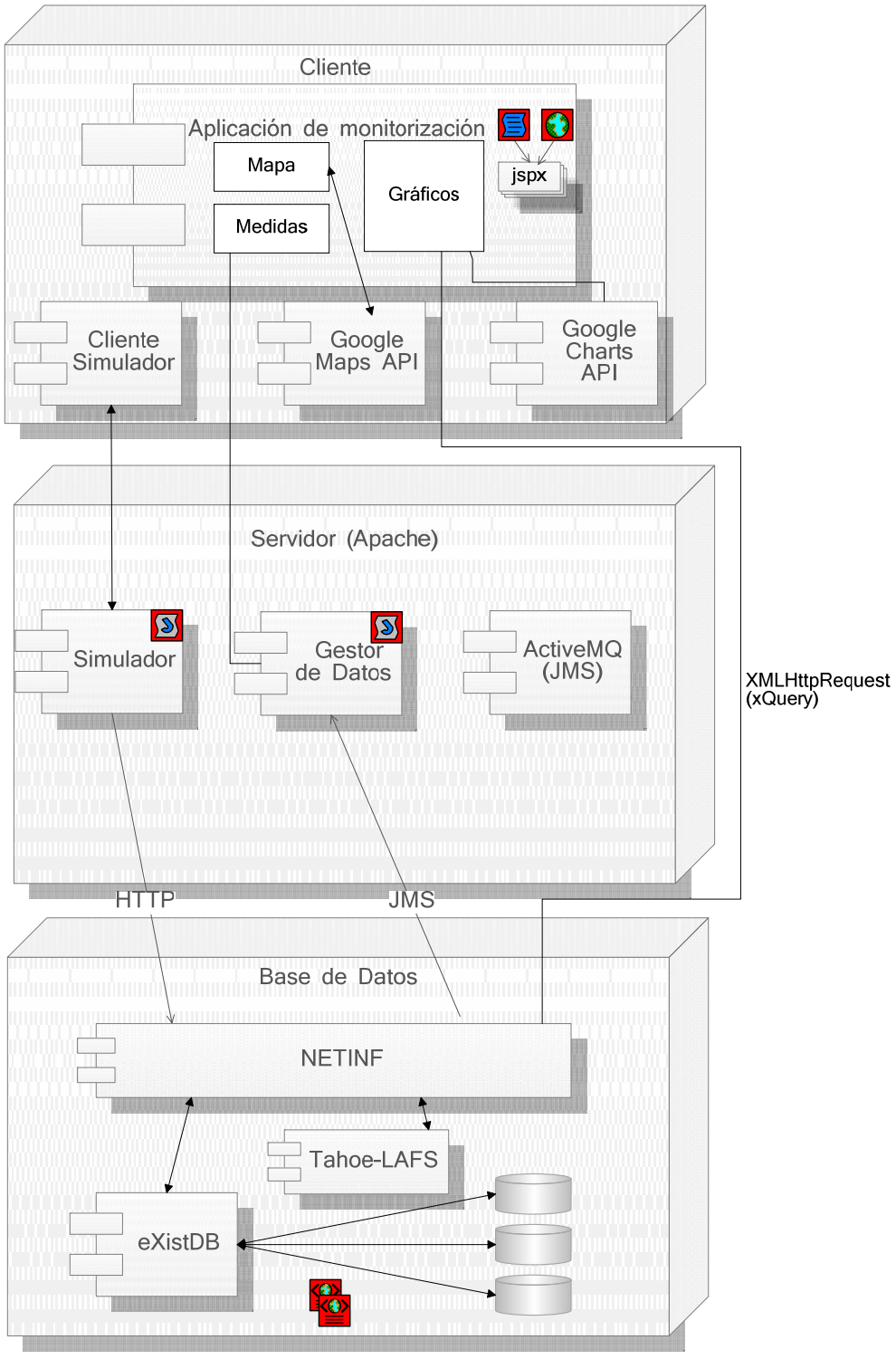


**Figura 1: Diagrama de la arquitectura del sistema inicial**

A la vista de este diagrama, distinguimos tres partes bien diferenciadas en la arquitectura del sistema: el cliente, el servidor y la base de datos. A pesar de que en el PFC todas las partes están implementadas en el mismo equipo, en la práctica cada una se encontrará en una localización. Es decir, con cliente nos

referimos al navegador, que puede estar situado en cualquier lugar del mundo, desde el cual un usuario accede a la aplicación vía Internet. Por otra parte, tanto el servidor o aplicación del lado del servidor como la base de datos (que incluye el espacio de almacenamiento físico y la interfaz de gestión y administración) se encontrarán situados en un conjunto de equipos destinado a esa finalidad en la universidad Telecom SudParis.

A continuación describimos brevemente cada una de estas tres entidades por separado y las comunicaciones que existen entre ellas.

### 2.2.1. Cliente

En la parte del cliente tenemos la aplicación de monitorización en sí misma, de la cual hablaremos con más detalle en el Capítulo 5 de la memoria. Por ahora baste decir que está construida como una aplicación web dinámica, usando un conjunto de tecnologías que incluyen: JSP (Java Server Pages) como tecnología Java que nos permite generar contenido dinámico para web y Ajax (Asynchronous JavaScript And XML) que es una técnica de desarrollo web que nos permite realizar cambios sobre las páginas sin necesidad de recargarlas, lo cual es muy importante en este proyecto, ya que necesitamos proveer al usuario de información en tiempo real.

El usuario dispone de tres herramientas principales en la aplicación de monitorización:

- Un **mapa** del medio acuático, en el que puede localizar los sensores en cualquier instante de tiempo. Dicho mapa es creado e integrado en la aplicación por medio del API de **Google Maps**.

- Un **cuadro de medidas**, en el que el usuario puede visualizar en tiempo real las medidas realizadas, así como otras informaciones proporcionadas por los sensores. Este cuadro de medidas ha sido creado mediante el marco de trabajo ICEfaces y obtiene los datos del simulador, situado en el servidor, mediante una conjunción de las tecnologías JSP y Ajax.

- Un conjunto de **gráficos** que el usuario puede utilizar para tener una visión histórica de los parámetros del agua. Estos gráficos se crean e integran en la aplicación mediante el API de **Google Charts**. Los datos que nutren los gráficos son recuperados directamente de la base de datos, enviando consultas xQuery a la misma mediante el objeto XMLHttpRequest.

Además está el Simulador en la parte de cliente que es desde donde se pone en marcha la simulación.

### 2.2.2. Servidor

En la parte de la aplicación correspondiente al servidor, programada en lenguaje Java, se identifican dos módulos principales:

- **Simulador:** se trata de las clases Java que imprimen movimiento a los sensores simulando el movimiento real que tendrán éstos en el medio acuático en el que se sitúen. A su vez tiene la tarea de crear documentos XML con los datos que genera y enviarlos a la base de datos vía HTTP.

- **Gestor de datos:** consiste en la estructura de clases Java que se va a utilizar para almacenar la información de los sensores y para proporcionársela a la aplicación web de monitorización en el cliente. Es en esta estructura donde se inicia la comunicación con la base de datos vía el servicio de mensajes de Java (JMS), configurando este extremo de la comunicación como el consumidor de los mensajes.

Para poder invocar al gestor de datos y al simulador desde la aplicación de monitorización se usa una funcionalidad de la tecnología JSF, los **Managed Beans**. Éstos nos dan la posibilidad de trabajar con datos dinámicos en la aplicación web, ya que podemos acceder a los atributos y métodos de las clases Java así definidas desde nuestras páginas JSPX.

Además de los módulos pertenecientes a la aplicación propiamente dicha, en la Figura 1 podemos observar otra entidad que ha de estar corriendo para que la aplicación funcione. Se trata del **ActiveMQ** que es el proveedor del servicio de mensajes Java (JMS), que utilizamos para que la base de datos envíe los documentos XML que contienen la información de los sensores en cuanto los recibe del simulador. Una instancia de ActiveMQ ha de estar corriendo tanto en el servidor como en la base de datos (si se encuentran en equipos diferentes) para que la aplicación funcione correctamente.

El servidor en el que corre la aplicación es **Apache Tomcat**, se trata de un servidor web que tiene soporte para servlets y JSPs. Esto entre otras características y cualidades lo hacen propicio para ser usado en el proyecto.

### 2.2.3. Base de datos

Como sistema de almacenamiento para los datos del proyecto Mobesens se utiliza una base de datos nativa XML, la interfaz de comunicación con esta base de datos ha sido creada en el departamento RS2M de la universidad Telecom SudParis, ha sido programada en el lenguaje Java y se le ha dado el nombre de **NetInf**.

NetInf hace uso de la herramienta **eXistDB**, que facilita la gestión y administración de bases de datos nativas XML; asimismo se utiliza un sistema de almacenamiento en nube (cloud), que proporciona privacidad y seguridad en la distribución de la información hacia múltiples servidores, llamado **Tahoe-LAFS**.

Además NetInf toma parte en la comunicación JMS de la que hemos hablado entre servidor y base de datos, en la que se configura como el extremo Publicador de la comunicación. Información más detallada sobre el funcionamiento de la base de datos se encuentra en el Capítulo 3 de la memoria.

## *2.3. Situación final*

En la fase final del proyecto, habrá varios cambios en lo que se refiere a la arquitectura del sistema. Principalmente en las partes del servidor y de la base de datos. En la Figura 2, tenemos un diagrama de la arquitectura de la aplicación en lo que sería la fase final del proyecto Mobesens. En ella podemos observar las diferencias con respecto al esquema anterior.

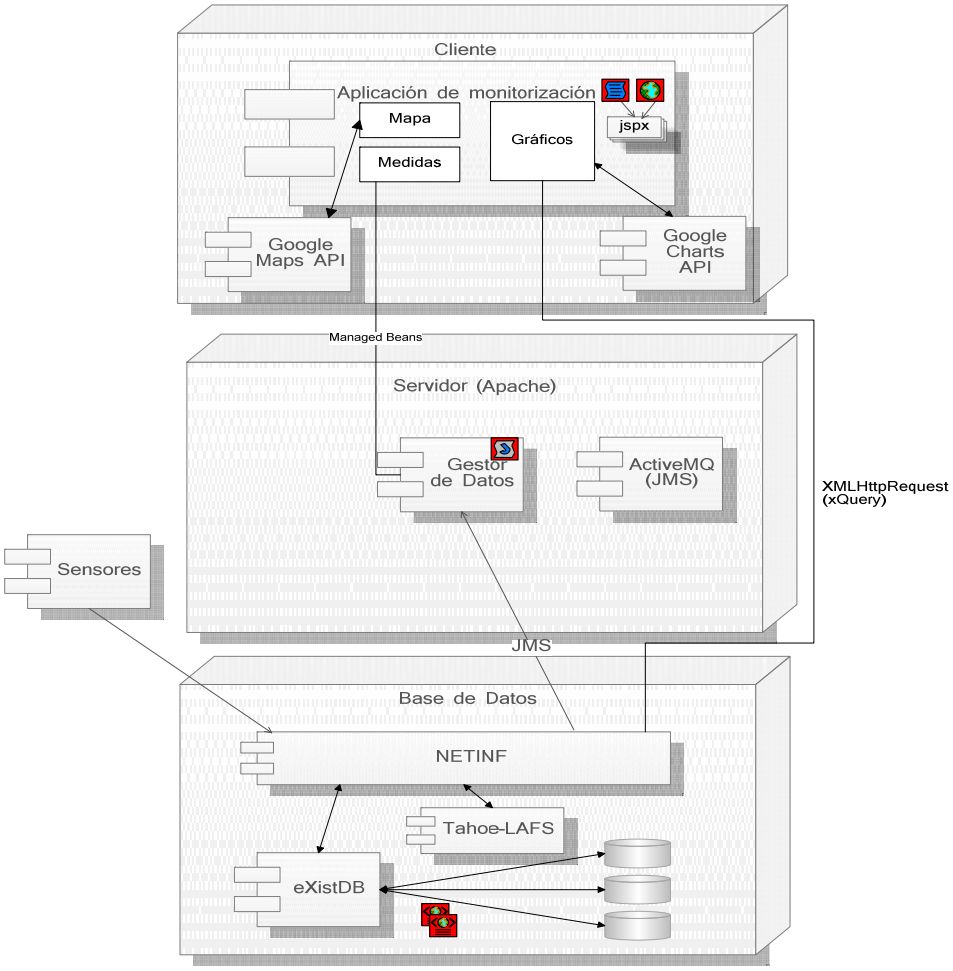


**Figura 2: Diagrama de la arquitectura del sistema final**

Dado que en ese momento del proyecto ya estarán situados los sensores en el medio acuático a monitorizar, aparece un nuevo componente al que llamamos Sensores, que se refiere a la información que estos envían a través de la red de comunicaciones hasta la base de datos. Con este nuevo componente no serán necesarios, por tanto, ni el módulo del simulador presente en el servidor ni su comunicación vía HTTP con la base de datos, ni el módulo del simulador en la parte del cliente. El resto de bloques de la estructura del sistema se mantendrán iguales a no ser que en el desarrollo de las últimas fases del proyecto sea necesario cambiar algo o hacer algún añadido.

# Capítulo 3. Base de datos nativa XML

## 3.1. Introducción

Uno de los objetivos clave en el proyecto Mobesens es la integración del mismo con futuros trabajos o proyectos de la Unión Europea en el ámbito de vigilancia y prevención de riesgos ecológicos. En especial se desea que el sistema de almacenamiento de la información sea lo más genérico y escalable posible para que futuros proyectos puedan hacer uso del mismo. En vista de esto, queda claro que una base de datos relacional no es una buena elección, ya que su estructura tabular dificulta que una expansión sea llevada a cabo de manera sencilla. Se pensó por tanto en otro tipo de base de datos, llamada base de datos nativa XML, que brinda la escalabilidad y potencia necesarias para el proyecto Mobesens.

La característica principal de una base de datos nativa, es que no almacena datos como tales, sino documentos enteros en formato XML o cualquier tipo de documento en otro formato que pueda ser convertido a XML. Además nos permite el procesamiento de los datos contenidos en los documentos XML (aunque de una forma quizá más complicada de lo que nos resultaría en una base de datos relacional, dado que la información en los documentos XML tiene una estructura jerárquica); así como el almacenamiento y posterior recuperación de la información y obviamente la posibilidad de realizar búsquedas en la base de datos, mediante un lenguaje de consulta.

## 3.2. Estructura

Como podemos ver en el esquema de la base de datos de la Figura 3, NetInf es toda una aplicación que se ocupa de gestionar la base de datos, aislando a los usuarios del funcionamiento interno de la misma. Consta de una interfaz de comunicación en desarrollo, mediante la cual se podrá realizar la carga y recuperación de documentos y otras múltiples acciones sobre los datos almacenados.

Además NetInf crea lo que llamamos espacio de indexación. Es una parte fundamental de la base de datos, ya que nos permite la elaboración de un índice que contenga la información de forma ordenada, de tal manera que las búsquedas en la base de datos se realicen más rápida y eficientemente y obtengamos los resultados en menos tiempo. Lo que en nuestro caso es muy importante, dado que la aplicación web cuenta con funcionalidades en tiempo real que se nutren de la información contenida en la base de datos y por tanto un sencillo y rápido acceso a ellos es crucial.

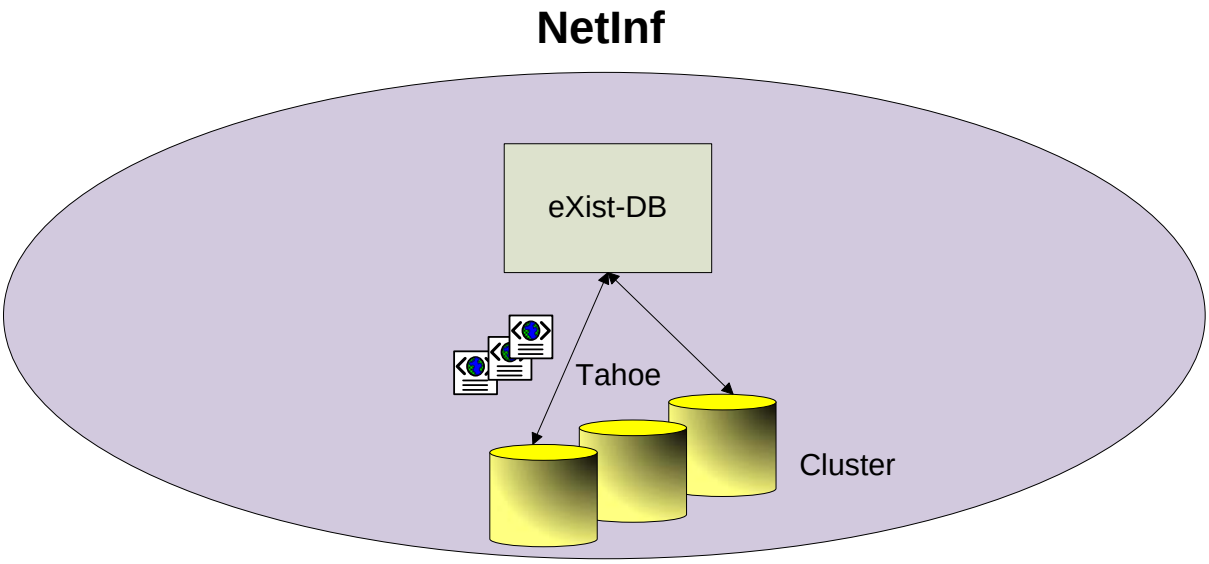


**Figura 3: Esquema de la base de datos**

Como espacio físico de almacenamiento de los datos, se usa un "cluster" o conjunto de ordenadores conectados en red que ha sido construido en la universidad Telecom SudParis. Para la gestión y administración de dicho cluster se hace uso de un sistema de almacenamiento en nube, abierto y gratuito, llamado Tahoe-LAFS [2]. Este sistema distribuye los datos a través de múltiples servidores, de tal forma que en caso de que los servidores fallen o sean víctima de un ataque, el sistema de ficheros al completo continúe funcionando correctamente. Además preserva la seguridad y privacidad de los datos.

Para la gestión de la base de datos, se cuenta con eXist-DB [3]. Se trata de una herramienta de administración de bases de datos open source, que hace uso de la tecnología XML y utiliza como lenguaje de consulta xQuery. Además soporta muchas tecnologías web, que le convierten en el entorno idóneo para el desarrollo de aplicaciones web, como la que se ha desarrollado en este PFC.

## *3.3. Formato de la información: XML*

La información almacenada en nuestra base de datos, estará disponible en formato XML. XML es en realidad un metalenguaje extensible de etiquetas desarrollado por el W3C, que nos permite el intercambio de información estructurada entre diferentes plataformas. Esto nos viene muy bien, dada nuestra futura necesidad de integrar diferentes proyectos. Para ello XML intenta estructurar la información de la manera más abstracta y reutilizable posible. Esto no quita para que todo documento XML deba estar "bien formado", es decir, que cumpla con todas las definiciones básicas y reglas de formato y así

pueda ser correctamente analizado por cualquier analizador sintáctico (parser) que cumpla la norma.

A la hora de estructurar y dar forma a los documentos XML que almacenan la información de los sensores, se han tenido en cuenta los estándares del OGC (Open GeoSpatial Consortium) [4]. Se trata de un consorcio de empresas, agencias gubernamentales y universidades; que desarrolla en consenso todo tipo de estándares abiertos e interoperables en el marco de los sistemas de información geográfica y de la worl wide web. En concreto el esquema de los documentos XML que contienen la información de los sensores está basado y sigue las especificaciones de los estándares OM (Observations and Measurements) [5] y SWE (Sensor Web Enablement) Common Data Model [6], que nos proporcionan modelos de documentos para el intercambio de información sobre las observaciones y sus resultados; así como para el intercambio de datos relacionados con sensores. De esta forma diferentes aplicaciones y/o servidores pueden estructurar, codificar y transmitir conjuntos de datos de los sensores de una manera autodescriptiva y permitida semánticamente.

En la Figura 4 podemos observar una muestra de la estructura y forma del documento XML definido a lo largo del proyecto.

```
<?xml version="1.0" encoding="UTF-8"?>
<om:Observation xmlns:om="http://www.opengis.net/1.0../om.xsd"
xmlns:gml="http://www.opengis.net/gml" xmlns:swe="http://www.opengis.net/swe/1.0.1">
  <om:result>
    <swe:DataRecord definition="urn:ogc:def:property:OGC:ISFETSpecialOutput">
      <swe:field name="SensorType">
        <swe:text definition="urn:sensor:classifier:sensorType">
          <swe:value>ISFET-Ta2O5</swe:value>
        </swe:text>
      </swe:field>
      <swe:field name="UID">
        <swe:text definition="urn:ogc:def:identifier:OGC:uuid">
          <swe:value>EF00145222ABC678</swe:value>
        </swe:text>
      </swe:field>
      <swe:field name="Time">
        <swe:Time definition="urn:ogc:def:identifier:OGC:uuid">
          <swe:uom Xlink="urn:ogc:def:unit:ISO:8601" />
          <swe:value>Sun Aug 14 12:32:28 CEST 2011</swe:value>
        </swe:Time>
      </swe:field>
      <swe:field name="temperature">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_temperature">
          <swe:value>1.7827576</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="acidity">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_pH">
          <swe:value>6.721234</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="NH4plus">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_NH4plus">
          <swe:uom code="mMol" />
          <swe:value>6.697467</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="Ca">
        <swe:Quantity definition="urn:ogc:def:property:OGC:measure_Ca">
          <swe:uom code="mMol" />
          <swe:value>8.09595</swe:value>
        </swe:Quantity>
      </swe:field>
```

```
        <swe:field name="k">
          <swe:Quantity definition="urn:ogc:def:property:OGC:measure_K">
            <swe:uom code="deg" />
            <swe:value>5.2739477</swe:value>
          </swe:Quantity>
        </swe:field>
         <swe:field name="GPS_Location_Vector">
           <swe:Vector definition="urn:ogc:def:property:GPSlocationVector">
             <swe:coordinate name="latitude">
               <swe:Quantity definition="urn:ogc:def:phenomenon:latitude">
                 <swe:uom code="deg" />
                 <swe:value>46.394</swe:value>
               </swe:Quantity>
             </swe:coordinate>
             <swe:coordinate name="longitude">
               <swe:Quantity definition="urn:ogc:def:phenomenon:longitude">
                 <swe:uom code="deg" />
                 <swe:value>6.342461666666666</swe:value>
               </swe:Quantity>
             </swe:coordinate>
             <swe:coordinate name="altitude">
               <swe:Quantity definition="urn:ogc:def:phenomenon:altitude">
                 <swe:uom code="deg" />
                 <swe:value>0</swe:value>
               </swe:Quantity>
             </swe:coordinate>
           </swe:Vector>
         </swe:field>
       </swe:DataRecord>
     </om:result>
   </om:Observation>
```

**Figura 4: Esquema del documento XML**

Hay que decir que, en su momento, se planteó la opción de utilizar también el formato JSON (JavaScript Object Notation). JSON es un formato ligero para el intercambio de datos cuyo uso se ha generalizado debido a que resulta una alternativa muy atractiva a XML en el uso de la tecnología Ajax. Para nosotros resultaba una opción atrayente dado que hacemos uso de dicha tecnología y sobretodo porque es el formato que usa Google en su Google Charts API y nosotros nos servimos de dicha aplicación a la hora de crear los gráficos. En cualquier caso esta opción no queda descartada y se podría incorporar en cualquier momento, ya que en la base de datos se pueden almacenar documentos que no sean de tipo XML.

## *3.4. Lenguaje de consulta*

Dado que eXist-DB utiliza como lenguaje de consulta para interrogar a la base de datos xQuery[3], es este lenguaje el que hemos aprendido a manejar y usado a lo largo del proyecto para recuperar los datos necesarios para la aplicación web. xQuery nos proporciona los medios para extraer y manipular información de documentos XML.

Se trata de un lenguaje de consulta muy completo, ya que incluye algunas capacidades de programación como la creación de funciones propias del usuario (también tiene funciones predefinidas) que pueden ser muy útiles; y utiliza expresiones XPath que nos permiten buscar y seleccionar datos teniendo en cuenta la estructura jerárquica de los documentos XML, así como unas expresiones parecidas a las usadas en SQL, conocidas como expresiones

FLWOR. Estas expresiones toman su nombre de los 5 tipos de sentencias de las que pueden estar compuestas: FOR, LET, WHERE, ORDER BY y RETURN. Además, xQuery nos permite construir o reconstruir nuevos documentos XML a partir de los resultados de la consulta.

# Capítulo 4. Aplicación de la parte del servidor

En este capítulo de la memoria vamos a describir más detalladamente la arquitectura de la aplicación, en lo que se refiere a la parte del servidor. Como ya hemos comentado en el Capítulo 2, tenemos dos módulos principales: el Simulador y el Gestor de Datos ambos programados en lenguaje Java [7]. A continuación vamos a explicar las distintas funciones y tareas que llevan a cabo cada uno de ellos.

## *4.1. Simulador*

A pesar de que a la finalización del proyecto Mobesens, no formará parte de la aplicación web, el simulador es en estos momentos una de las partes fundamentales del proyecto. Como su nombre indica, realiza una simulación, en la que crea un cierto número de sensores, les asigna unas medidas y hace que esos sensores se muevan por la superficie del lago (en lo que llamamos “paseo”) y modifica los valores de las medidas. De esta manera, a efectos de la visualización, tenemos un sistema real, en el que podemos observar los sensores, ver como se mueven por el mapa y acceder a las medidas que realizan.

En la Figura 5, se observa el diagrama de flujo de datos del simulador, que explicamos a continuación:

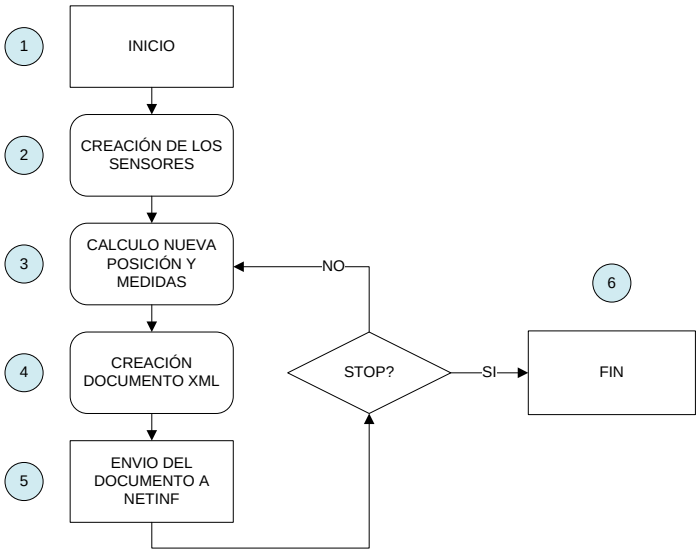


**Figura 5: Diagrama de flujo de datos del simulador**

1º. El simulador se pone en marcha mediante un botón en la aplicación de monitorización que dispara un evento manejado desde las clases java.

2º. Se crean los sensores como procesos (threads) para que cada uno se comporte como una entidad independiente y tengan así diferentes conductas en cuanto a su movimiento y medidas. Se definen varios sensores con sus posiciones de inicio y fin del paseo.

3º. Se imprime movimiento a los sensores mediante la clase RandomWalk, cuyo funcionamiento queda explicado en la Figura 6. A partir de la duración de la simulación se calcula el número de pasos que tienen que realizar los sensores a lo largo de su paseo. Una vez hallado el número de pasos y utilizando como referencia las coordenadas del punto de inicio del paseo y del punto final, se calculan los pasos de cada movimiento en latitud y longitud y de ahí cada vez que se requiere, la nueva posición del sensor.

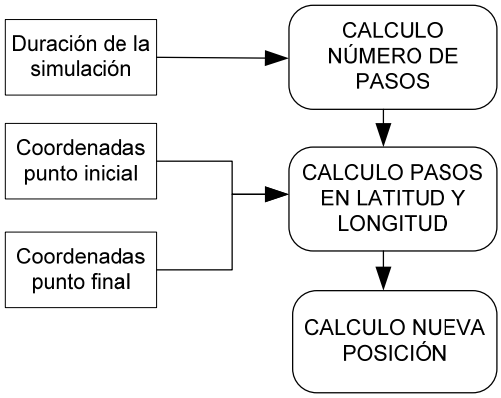


**Figura 6: Esquema explicativo del RandomWalk**

A su vez se calculan las medidas, para cada nueva posición de los sensores, de forma pseudo-aleatoria mediante una función Random modificada.

4º. Se crea un documento XML con los datos de cada nueva posición y/o medidas de los sensores según el formato comentado en la sección 3.3, basado en los estándares del OGC.

5º. Se envía dicho documento XML a la base de datos vía HTTP.

6º. Los procesos asociados a cada sensor se paran mediante un botón en la aplicación de monitorización.

## *4.2. Gestor de datos*

La principal tarea del gestor de datos es crear una estructura de datos que permita describir toda la información referente a los sensores, de tal manera que se pueda acceder a ella desde la aplicación de monitorización. Asimismo el componente del gestor de datos debe mantener una comunicación

con la base de datos, que le permita a su vez obtener la información de los sensores en el momento que ésta llega a la base de datos, para conseguir un sistema que trabaje en cuasi tiempo real.

En el proyecto Mobesens se han creado los siguientes tipos de sensores:

- GIME (Gel-Integrated MicroElectrode Array): para la detección de tóxicas de varios metales pesados como el Cobre, el Plomo, el Cadmio o el Zinc.
- ISFET: mide parámetros del agua como el pH, NO3,NH4 y Ca2.
- BAW (Bulk Acoustic Resonator): mide el pH, el Potasio y PAH (Polycyclic Aromatic Hydrocarbons).

Todos estos sensores se pueden representar mediante la estructura de datos presentada en el diagrama de clases de la Figura 7.

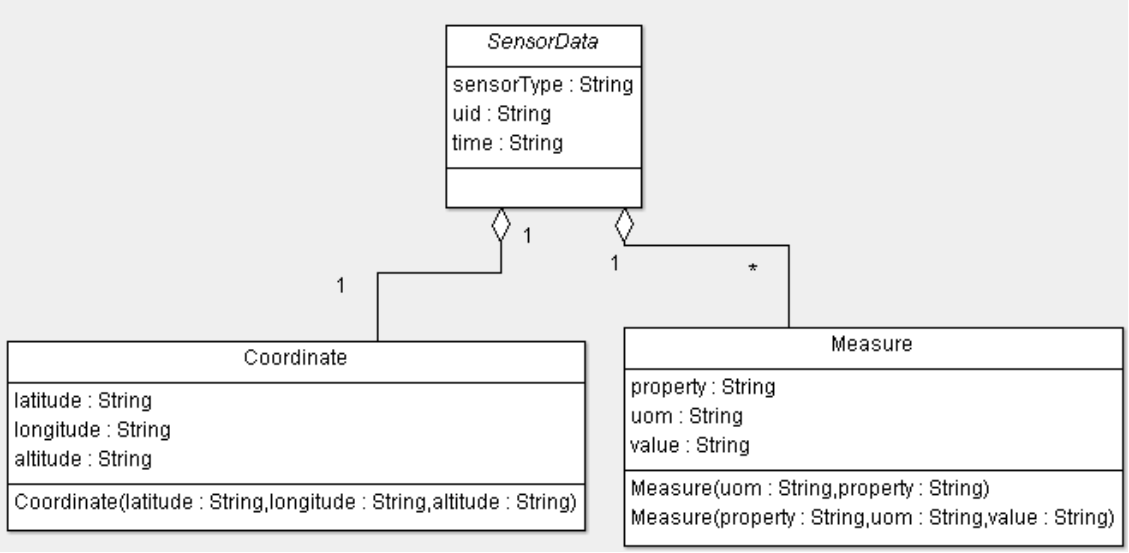


**Figura 7: Estructura de datos de los sensores**

Como podemos observar en la Figura 7 la información básica de un sensor consiste en:

- Tipo: tipo de sensor, ya sea GIME, ISFET o BAW
- Identificador: se trata del número o combinación alfanumérica que identifica al sensor.
- Fecha: fecha y hora de la realización de una medida o conjunto de medidas.
- Coordenadas: posición geográfica del sensor descrita en Latitud, Longitud y Altitud (la altitud es importante porque algunos sensores se desplazan verticalmente en el medio acuático).
- Medidas: vector de los parámetros medidos por el sensor. Cada medida queda descrita por la propiedad, la unidad en que se mide y el propio valor de la medida.

El gestor de datos, además de contener las clases java que representan a los sensores, cumple las siguientes funciones, asociadas a las diferentes fases del diagrama de flujo de datos de la Figura 8:

1º. Poner en marcha el servicio de mensajes java (JMS) para abrir el canal en el extremo de la comunicación correspondiente al servidor.

2º. Una vez puesta en marcha la comunicación nos quedamos a la espera de cualquier mensaje JMS que llegue de la base de datos conteniendo los documentos XML.

3º. Extraer la información de los sensores de los documentos XML.

4º. Actualizar la lista de sensores con la nueva información.

5º. Las medidas de los sensores y su posición visualizadas en la aplicación de monitorización se recogen de la lista de sensores contenida en el gestor de datos.

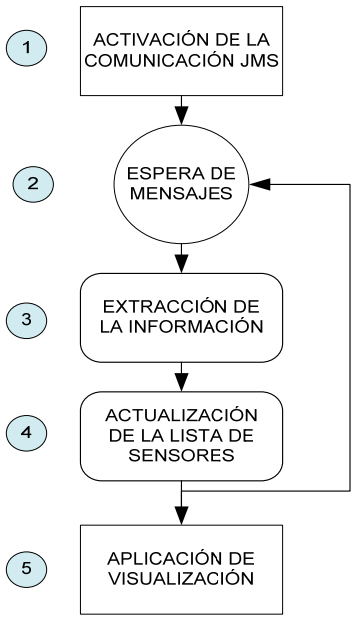


**Figura 8: Diagrama de flujo de datos del gestor**

# Capítulo 5. Descripción de la aplicación de monitorización

Como ya se ha comentado en el Capítulo 2 de la memoria, la aplicación de monitorización consta básicamente de tres partes:

- **Mapa.** El mapa realizado e integrado en nuestra aplicación web mediante el API de Google Maps [8], es una de las partes fundamentales de la aplicación, ya que permite a los usuarios ver donde están situados los sensores en todo momento.

- **Cuadro de medidas.** Es una tabla donde los usuarios podrán visualizar en todo momento las medidas llevadas a cabo por los sensores en tiempo real. La integración del cuadro de medidas en la aplicación se lleva a cabo mediante los componentes que ofrece le marco de trabajo ICEfaces [9].

- **Gráficos.** La sección de creación de gráficos a partir de la información de los sensores, es también uno de los aspectos clave de la aplicación, ya que es la forma que tienen los usuarios de acceder a los datos históricos de una manera más visual y coherente, como es su representación gráfica en función del tiempo o de la profundidad. Su realización e integración en nuestra aplicación se realiza mediante el API de Google Charts[10].

En las siguientes secciones de este capítulo se van a describir más detalladamente cada una de estas partes.

## *5.1. Mapa y medidas*

La idea de integrar un mapa en nuestra aplicación web, es que los usuarios puedan situar los sensores en dicho mapa y tener así una visión general del emplazamiento de éstos. Dado que los sensores no están fijos en una posición determinada, sino que se pueden ir moviendo, ya sea porque tienen un control remoto (como el kayak) o porque las propias corrientes del medio acuático los mueven, el mapa deberá reflejar asimismo el movimiento de los sensores, es decir, nos permitirá ver en todo momento la localización específica de cada uno de ellos. Para ello, es necesario por tanto que el mapa se nutra con información en tiempo real, obtenida de la base de datos.

Se ha elegido centrar el mapa en el lago Lemán, situado entre Suiza y Francia a cuyas orillas se encuentra la ciudad de Ginebra, dado que las pruebas con los sensores se realizarán en él. Sin embargo, el proyecto, abarca cualquier superficie acuática, ya sean lagos, ríos o incluso mares; con lo cual una vez el

proyecto Mobesens esté en pleno funcionamiento, el mapa tendrá que ser capaz de mostrar los sensores deseados independientemente de su localización.

Por otra parte, todas las medidas realizadas por los sensores se visualizan en el cuadro de medidas de la aplicación web. Se trata de una tabla, en la que podemos ver información sobre cada uno de los sensores. Esta información incluye el identificador del sensor (que será único para cada sensor, para así poder identificarlos individualmente), el tipo de sensor (para poder organizarlos por tipos en caso de ser necesario), las coordenadas exactas del sensor en un momento dado expresadas en latitud, longitud y altitud y los valores de sus mediciones de parámetros del agua. Dado que en este momento todavía no se tienen datos reales de los sensores, se han incluido como medidas algunos parámetros del agua que sabemos que los sensores van a medir, como la temperatura, el pH, el nivel de Calcio, el nivel de NH4+ y el nivel de Potasio; aunque una vez que se tengan las medidas reales de los sensores se pueden incluir o quitar parámetros con facilidad.

En la Figura 9, podemos ver la página principal de la aplicación de monitorización en la que se presentan el mapa y el cuadro de medidas.


| -- Type -- | -- UID -- | -- Temperature -- | -- PH -- | -- Ca -- | -- NH4plus -- | -- k -- | -- Latitude -- | -- longitude -- | -- altitude -- |
|---|---|---|---|---|---|---|---|---|---|
| ISFET-Ta2O5 | EF00145222ABC133 | 14 | 7 | 8 | 4 | 9 | 46.462205 | 6.536865 | 0 |
| ISFET-Ta2O5 | EF00145222ABC55 | 6.1807127 | 2.0449133 | 5.960926 | 5.7206445 | 1.1780542 | 46.45 | 6.766913333333334 | 0 |
| ISFET-Ta2O5 | EF00145222ABC4156 | 9.974679 | 3.6536956 | 0.3286773 | 5.3314166 | 6.042456 | 46.412543 | 6.88527 | 0 |
| ISFET-Ta2O5 | EF00145222ABC123 | 9.787485 | 3.4299808 | 3.6589246 | 4.6562934 | 0.84938467 | 46.45208376 | 6.438848155 | 0 |
| ISFET-Ta2O5 | EF00145222ABC678 | 1.2864006 | 4.4886074 | 9.470098 | 9.168435 | 8.654832 | 46.39367 | 6.343385 | 0 |

**Figura 9: Mapa y cuadro de medidas**

Tanto el mapa como el cuadro de medidas han de estar actualizándose continuamente con las informaciones provenientes del Gestor de Datos. En la Figura 10, se muestra un diagrama del comportamiento de la aplicación en cuanto a estas actualizaciones:

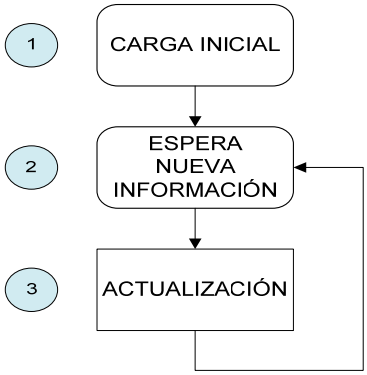


**Figura 10: Comportamiento del mapa y cuadro de medidas**

1º. Se realiza la carga inicial de la página con el mapa y el cuadro de medidas.
2º. La aplicación se queda a la espera de nueva información de los sensores.
3º. La actualización del mapa y del cuadro de medidas se realiza de forma automática en cuanto se tiene nueva información en el Gestor de Datos. Esto se consigue para el cuadro de medidas mediante los componentes de ICEfaces que utilizan una combinación de las tecnologías JSF (Java Server Faces) y Ajax y para el mapa mediante una función JavaScript llamada setInterval

## *5.2. Gráficos*

Toda la parte que tiene que ver con las representaciones gráficas de los datos de los sensores es de vital importancia en la aplicación web, ya que permite a los usuarios visualizar los datos de una manera más sencilla, coherente y agradable a la vista. El diseño, creación e integración de los gráficos en la aplicación se realiza mediante el API de Google Charts.

Después de una reunión con los compañeros que se encargaban del diseño y fabricación de los sensores y con otras personas que ya tenían cierta experiencia con sensores en medios acuáticos y por tanto podían aportar una opinión más cercana al punto de vista del usuario; se decidió que existía la necesidad de integrar en la aplicación de monitorización dos tipos de gráficos fundamentales: históricos y de perfil. En ambos casos se proveerá al usuario de un menú para seleccionar los parámetros del agua que se quieran representar y en el caso de los gráficos históricos también existirá un campo para introducir el intervalo de tiempo del que se quieren visualizar los datos.

Los gráficos históricos consisten en una representación de las medidas realizadas por los sensores en función del tiempo, de tal manera que los usuarios puedan revisar el comportamiento de uno o varios parámetros del agua. Para la creación de este tipo de gráfico histórico se ha utilizado la visualización interactiva del API de Google Charts llamada

`AnnotatedTimeLine` que nos permite realizar un gráfico interactivo, en el que aparecerán una serie de líneas (cada una de estas líneas representará una medida diferente del sensor) dibujadas en función del tiempo.

El término interactivo se usa en referencia a una serie de propiedades que tiene este tipo de gráfico que permiten al usuario realizar ciertas acciones que aumentan la usabilidad del mismo, estas acciones incluyen la posibilidad de hacer zoom in y zoom out en el gráfico (puede ser de mucha utilidad cuando los gráficos históricos abarcan largos períodos de tiempo) y la visualización de los datos concretos del gráfico dependiendo de donde se encuentre situado el puntero del ratón del usuario, lo que nos permite obtener información exacta de las medidas en cualquier instante concreto del intervalo de tiempo representado.

En la Figura 11 podemos ver la parte de la aplicación de visualización correspondiente a los gráficos históricos.

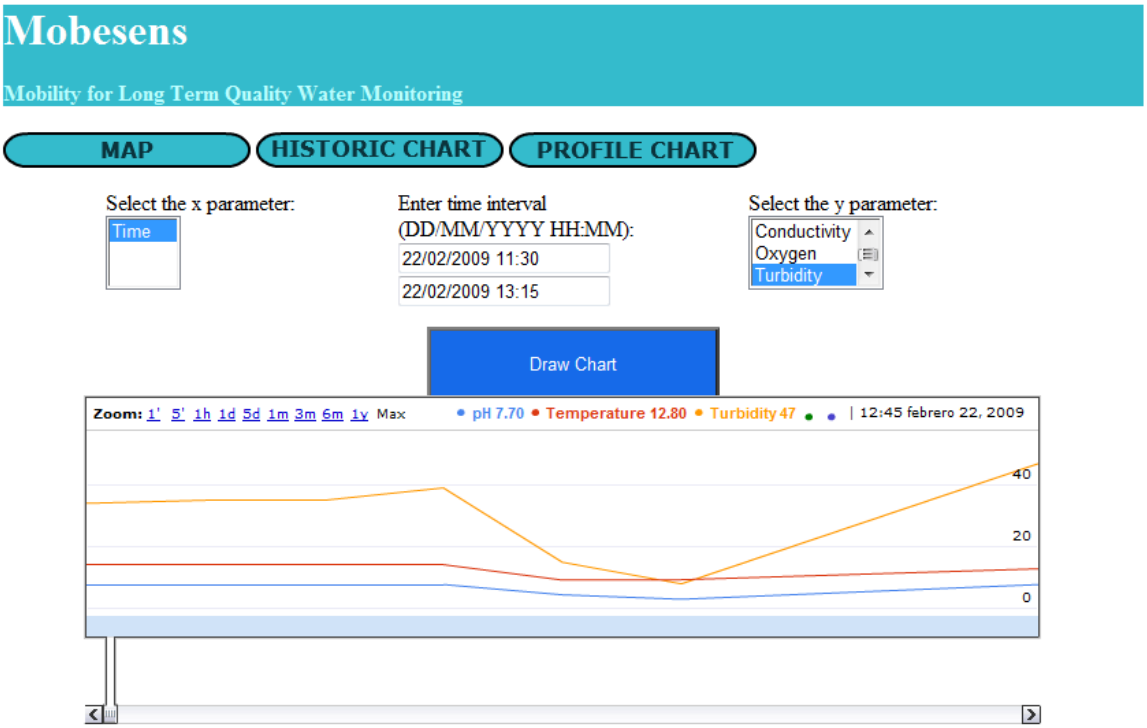


**Figura 11: Gráfico histórico**

Por otra parte, el gráfico de perfil es muy parecido al gráfico histórico, se trata de representar las medidas que realizan los sensores en función de la profundidad a la que han sido hechas, en vez de en función del tiempo como en el gráfico histórico. Para su creación e integración en la aplicación se ha utilizado la visualización `scatterChart` del API de Google Charts. En la Figura 12, se presenta la parte de la aplicación correspondiente a los gráficos de perfil.

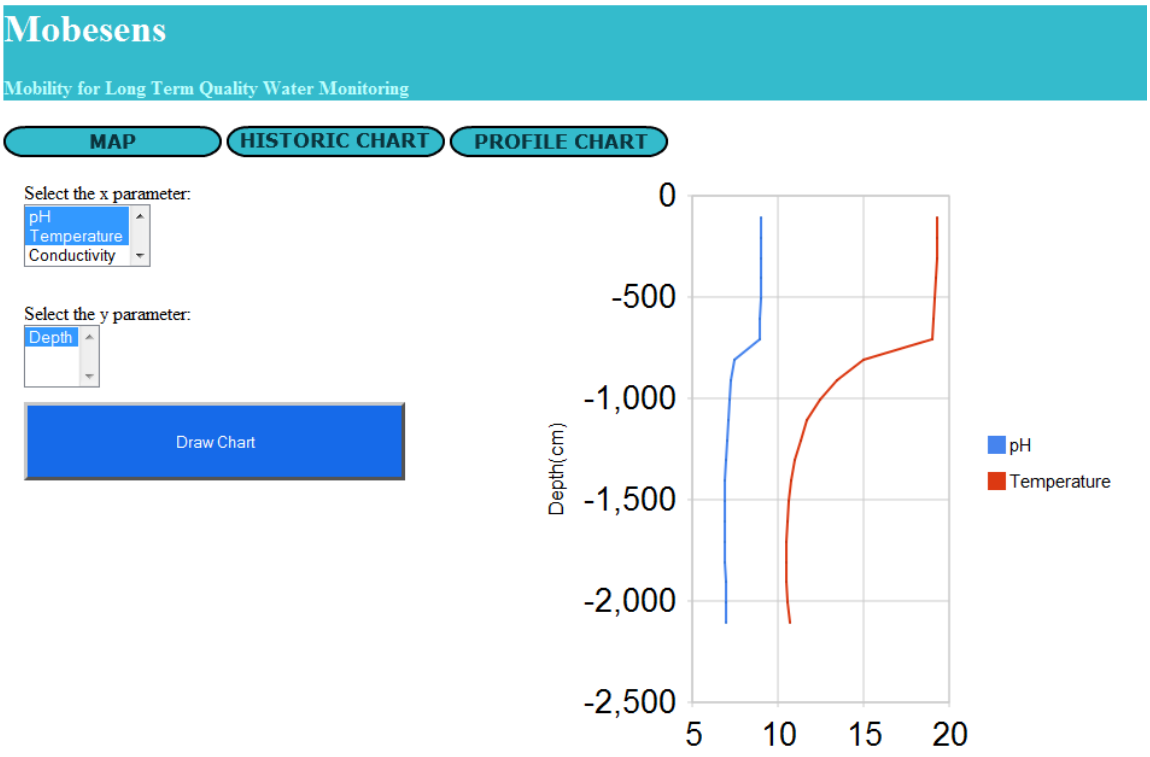


**Figura 12: Gráfico de perfil**

El proceso de creación de los gráficos es el mismo para los dos tipos, lo vemos ilustrado en la Figura 13 y lo detallamos a continuación:

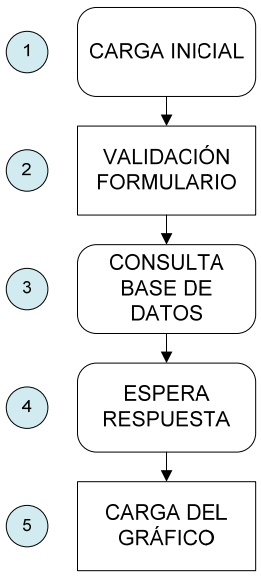


**Figura 13: Diagrama de flujo de datos de los gráficos**

1º. En la carga inicial de la página sólo podemos ver los menús mediante los que el usuario decide que parámetros quiere ver en el gráfico y un botón para dibujarlos una vez dichos parámetros hayan sido seleccionados.

2º y 3º. Se validan los parámetros seleccionados y se envía la consulta xQuery a la base de datos en función de ellos mediante el objeto XMLHttpRequest [11](forma básica de la tecnología Ajax).

4º y 5º. Se espera a que llegue la respuesta de la base de datos para construir el gráfico y cargarlo en la página. Esto se hace así porque si se intentara cargar el gráfico antes de tener los datos habría un error y no se cargaría el gráfico.

# Capítulo 6. Conclusiones y líneas futuras

## *6.1. Conclusiones*

A lo largo de este proyecto se ha desarrollado la primera fase de una aplicación web para la visualización y seguimiento de sensores en medios acuáticos. En esta primera fase, se han creado las partes principales que constituirán la aplicación web final y que incluyen:

- Un gestor de datos que forma parte de la aplicación del lado del servidor y que contiene una estructura de datos creada mediante clases Java para describir la información de los sensores y que hace de puente entre la base de datos y la aplicación de la parte del cliente.

- Un simulador programado en Java que suple la carencia, en esta primera fase del proyecto Mobesens, de datos reales para mostrar en la aplicación de monitorización.

- Un mapa, realizado mediante el API de Google Maps, de la zona donde están situados los sensores, sobre el que se visualizan dichos sensores y en el que se ve reflejado el movimiento de éstos en tiempo real. Incluyendo algunas funcionalidades como aumentar o disminuir la resolución del mapa, o la visualización de las medidas realizadas por los sensores mediante ventanas de información que se activan al hacer click sobre cualquiera de los sensores.

- Un cuadro de medidas, realizado mediante el marco de trabajo ICEfaces, en el que se visualizan en tiempo real todos los parámetros del agua medidos por sensores; así como información del propio sensor, como son su identificador y su tipo.

- Una serie de gráficos, realizados mediante el API de Google Charts, que nos permitirán representar las medidas de los sensores en función del tiempo (gráficos históricos) y en función de la profundidad a la que han sido realizadas dichas medidas (gráficos de perfil).

Esta aplicación web va a resultar muy útil para la monitorización y visualización de sensores en medios acuáticos en general y en particular dentro del marco del proyecto Mobesens. Permitirá a los usuarios disponer de una interfaz de fácil manejo y alta usabilidad para el estudio de medios acuáticos, ya sea para fines medioambientales o para otros fines.

Como experiencia profesional, este proyecto me ha enseñado las distintas fases de desarrollo que tiene un proyecto tan grande como es Mobesens, así como las dificultades que entraña coordinar a distintos grupos de gente que trabajan en paralelo. También me llevo conmigo muchos conocimientos técnicos que he ido aprendiendo en su realización: distintos lenguajes de programación que antes no conocía, conocimientos sobre bases de datos y manejo de algunas herramientas y entornos de trabajo.

Como experiencia personal, me ha servido sobretodo para aprender a resolver problemas sin la ayuda de nadie y a ser un poco autodidacta, pero también a trabajar en equipo.

## *6.2. Trabajos futuros*

Dado que este proyecto sólo abarca la primera fase de la creación de la aplicación web, hay varias líneas de trabajo que se pueden seguir para mejorarla:

- Construcción de sistemas de filtrado de información para el mapa y el cuadro de medidas, de tal manera que el usuario pueda decidir que datos visualizar.

- Ampliación de los gráficos con un gráfico tipo isobárico, que nos permita ver en un mapa zonas con distintos colores representando distintos valores de una misma medida.

- Implementación de gráficos en tiempo real tanto para la modalidad de gráfico histórico como para la de gráficos de perfil.

- Realizar un diseño personalizado de la aplicación para cada usuario de manera que aumente la usabilidad de la aplicación web.

# Bibliografía


[1] "Portal Web del proyecto MOBESENS (Mobility for Long Term Water Quality Monitoring)". http://www.mobesens.eu/ (Última actualización: Agosto 2011).

[2] "Portal Web del proyecto Tahoe-LAFS". http://tahoe-lafs.org (Última actualización: Agosto 2011).

[3] Wolfang M.Meier, 2009. "Exist, Open Source Native XML Database Documentation". http://exist.sourceforge.net/documentation.html (Último acceso: Agosto 2011).

[4] "Portal Web del Open Geospatial Consortium". http://www.opengeospatial.org/standards (Última actualización: Marzo 2011).

[5] Simon Cox. "Observations and Measurements-XML Implementation". http://www.opengeospatial.org/standards/om Marzo 2011.

[6] Alexandre Robin. "SWE Common Data Model Encoding Standard". http://www.opengeospatial.org/standards/swecommon . Enero 2011.

[7] Javier Nogueras Iso. Apuntes de la asignatura "Ampliación de Informática". Centro Politécnico Superior (2009).

[8] Google,2010. "API de Google Maps". http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/v2/introduction.html (Último acceso: Agosto 2011).

[9] ICEsoft Technologies, 2011. "ICEfaces 2 Documentation". http://wiki.icefaces.org/display/ICE/ICEfaces+2+Documentation (Último acceso: Agosto 2011).

[10] Google, 2011. "Google Chart Tools". http://code.google.com/intl/es-ES/apis/chart/interactive/docs/reference.html (Último acceso: Agosto 2011).

[11] "Tutorial sobre XMLHttpRequest". http://www.toutjavascript.com/savoir/xmlhttprequest.php3 (Último acceso: Agosto 2011).

[12] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Hedí y William Lorensen, 1996. "Modelado y diseño orientados a objetos, Metodología OMT". Prentice Hall.

[13] James Rumbaugh, Ivar Jacobson, Grady Booch. "El lenguaje unificado de modelado. Manual de referencia". Adisson Wesley (1999).

[14] Google, 2011. "Gmaps4jsf: The complete integration of Google Maps with the Java Server Faces". http://code.google.com/p/gmaps4jsf/ (Último acceso: Agosto 2011).

[15] "Manuales de desarrollo web y programación". http://www.desarrolloweb.com/manuales/ (Último acceso: Agosto 2011).

[16] "Introducción a JMS (Java Message Service)".

http://www.programacion.com/articulo/introduccion_a_jms_java_message_service_142 (Último acceso: Agosto 2011).

# Siglas y acrónimos

**AJAX**: Asynchronous Javascript And XML

**API**: Application Programming Interface

**DOM**: Document Object Model

**HTML**: HyperText Markup Language

**HTTP**: HyperText Transfer Protocol

**JMS**: Java Message Service

**JSF :** Java Sever Faces

**JSON**: JavaScript Object Notation

**JSP**: Java Server Pages

**OGC**: Open Geospatial Consortium

**OM**: Observations and Measurements

**OMT**: Object Modeling Technique

**PFC**: Proyecto Fin de Carrera

**RF**: Requerimiento Funcional

**RIA**: Rich Internet Application

**RNF**: Requerimiento No Funcional

**RS2M**: Réseaux et Services Multimédia Mobiles

**SQL**: Structured Query Language

**SWE**: Sensor Web Enablement

**UML**: Unified Modeling Language

**URL**: Uniform Resource Locator

**W3C**: World Wide Web Consortium

**XML**: Extensible Markup Language