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

ANEXOS 2/2

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

Índice de contenidos

Índice de figuras

Anexo A

Anexo B

Anexo C

ANEXO A. ANÁLISIS Y DISEÑO

1.1. 1. INTRODUCCIÓN

En este anexo de la memoria se pretende documentar y presentar más en detalle el proceso de análisis y diseño de la aplicación desarrollada. Dado que no es una tarea sencilla, se trató de encontrar procesos y metodologías a seguir, que fuesen sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del software final.

En concreto se ha utilizado a lo largo del proyecto la metodología OMT (Object Modeling Technique) [12], ya que es una de las más maduras y eficientes que existen en la actualidad. Esta metodología nos permitirá construir un modelo del problema a tratar, en nuestro caso la aplicación de visualización, de forma que consigamos una descripción resumida, concisa y abstracta del sistema, mostrando sus propiedades y conceptos más importantes.

En la siguiente sección se presentan los requisitos de la aplicación. Y a continuación se va presentando el análisis y diseño detallado de los componentes pertenecientes tanto a la aplicación de visualización como al simulador y el gestor de datos. La notación utilizada para los distintos modelos de análisis y diseño (diagramas de clases, diagramas de componentes, diagramas de secuencia) ha sido UML (Unified Modeling Language) [13].

1.2. 2. REQUERIMIENTOS

El primer paso en el desarrollo de un software, pasa por la definición de una serie de requisitos o requerimientos; se trata de un conjunto de técnicas y procedimientos que nos permiten conocer los elementos necesarios para definir un proyecto de software. Se dividen en funcionales y no funcionales.

Algunos de estos requerimientos fueron definidos al principio del proyecto y otros han sido añadidos a lo largo del mismo, después de algunas reuniones con el resto de colaboradores del proyecto Mobesens. Concretamente con los diseñadores de los sensores, que a falta de un cliente final, eran los que podían indicarnos con más precisión algunas funciones y comportamientos deseados en la aplicación de visualización.

2.1. FUNCIONALES

Los requerimientos funcionales especifican una condición o capacidad de un sistema requerida por el usuario para resolver un problema o alcanzar un objetivo.

RF-1: El usuario final ha de poder personalizar la configuración de la visualización del área monitorizada, con la capacidad de seleccionar todos o sólo algunos de los parámetros asociados con un punto de medida determinado.

RF-2: El interfaz de visualización ha de proveer medios para crear distintas capas, que pueden ser visualizadas en conjunto o por separado (por ejemplo: hacer visibles o enmascarar selectivamente los parámetros de las medidas).

RF-3: La aplicación ha de contar con una interfaz que permita al usuario final visualizar las medidas en forma de gráficos.

RF-4: Se hace necesario un gráfico histórico (parámetros del agua en función del tiempo).

RF-5: Se hace neceario un gráfico de perfil (parámetros del agua en función de la profundidad).

2.2. NO FUNCIONALES

Los requerimientos no funcionales se definen como una condición o capacidad que debe poseer un sistema par satisfacer un contrato, un estándar, una especificación u otro documento formalmente impuesto.

RNF-1: El sistema Mobesens debe ser compatible con los sistemas de información geográfica y estándares de medidas más usados y relevantes en cuanto a sistemas de monitorización de entornos acuáticos.

RNF-2: En cuanto a la interfaz gráfica de usuario, se requiere un acceso personalizado a los datos, a la visualización del área a monitorizar, a las medidas y valores.

RNF-3: La interfaz ha de permitir acceso online y bajo demanda de las medidas realizadas vía Internet.

RNF-4: El sistema Mobesens debe facilitar también la interacción con otras herramientas de medida para integrar los datos y procesamiento de sistemas externos e incorporar éstos en un marco de trabajo común para la visualización y el análisis. Facilitando por tanto la creación de un espacio europeo común para la monitorización medioambiental, administración, cooperación y colaboración.

RNF-5: Mobesens pretende usar el API de Google Maps para la visualización del área monitorizada.

RNF-6: Se pretende que la aplicación de visualización sea ergonómica y eficaz

RNF-7: La información a visualizar estará limitada a aquella más útil y relevante.

RNF-8: Como espacio de almacenamiento para los datos se usará una base de datos nativa.

RNF-9: El formato de información almacenada será documento XML, cumpliendo con los estándares de la OGC.

1.3. 3. ARQUITECTURA

La arquitectura del sistema consta de tres partes fundamentales: el Cliente, el Servidor y la Base de datos; como podemos observar en el diagrama de la Figura 1.

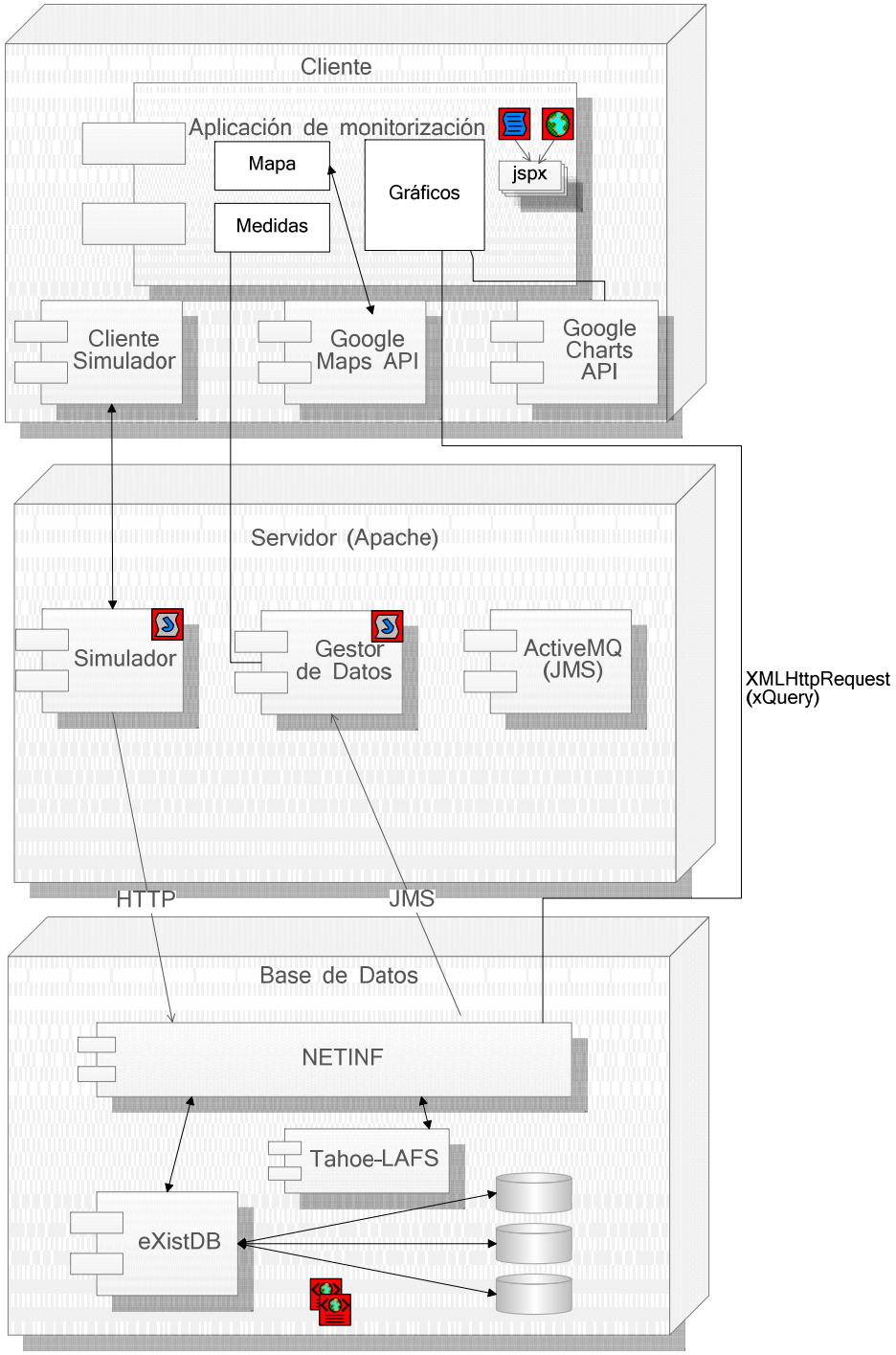


Figura 1: Diagrama de la arquitectura del sistema

A lo largo de las siguientes secciones de este Anexo A, se van a desgranar todos los detalles referentes al diseño y modelado de la aplicación, que incluye las partes Cliente y Servidor presentadas en el diagrama de la Figura 1. El diseño de la base de datos queda fuera del campo de trabajo de este PFC, ya que fue desarrollada por otros miembros del equipo Mobesens.

En la parte del servidor se encuentran el Simulador y el Gestor de Datos, ambos programados en Java. Asimismo una instancia del proveedor del servicio de mensajes Java, ActiveMQ, que nos proporciona la comunicación entre el Gestor de Datos y la propia base de datos.

En la parte del cliente, que es la que verán los usuarios finales, se presentan varias funcionalidades que permitirán al usuario monitorizar un medio acuático. Estas funcionalidades se incluyen en la aplicación de monitorización y son: un mapa de la zona, integrado mediante el API de Google Maps, sobre el que se visualizan los sensores y su movimiento; un cuadro de medidas en el que se presenta la información de los sensores que incluye los valores de las medidas realizadas en tiempo real, este cuadro se crea mediante el marco de trabajo ICEfaces; una serie de gráficos, integrados en la aplicación mediante el API de Google Charts, que permiten al usuario tener una visión más completa de las medidas, habiendo dos modalidades: históricos (parámetros del agua en función del tiempo) y de perfil (parámetros del agua en función de la profundidad).

Asimismo y aunque no pertenezca a la aplicación de monitorización como tal, tenemos el Simulador Cliente. Se trata de una pequeña interfaz gráfica, que podemos ver en la Figura 2 desde la que el administrador puede poner en marcha o parar el simulador.

Figura 2: Interfaz gráfica del simulador

Se trata de un fichero .jspx en el que se crean dos botones mediante el componente ICEfaces `commandButton` y a los que se les asocian mediante los Managed Beans sendos métodos de la clase `sensorDataBean` del simulador que arrancan o paran la simulación.

En las siguientes secciones se presenta el diseño de cada uno de los componentes que constituyen la aplicación mediante diagramas de clases para explicar el modelado de los componentes y diagramas de secuencia, en los que interaccionan todas las partes, para describir el comportamiento de la aplicación desde el punto de vista del administrador y desde el punto de vista del usuario.

1.4. 4. APLICACIÓN DE MONITORIZACIÓN

Como ya hemos dicho, la aplicación de monitorización tendrá tres funcionalidades importantes: un mapa de la zona integrado mediante Google Maps, un cuadro de medidas creado mediante ICEfaces y unos gráficos (históricos y de perfil) integrados mediante Google Charts. El diseño y modelado de estas tres partes de la aplicación será descrito más detalladamente en los siguientes puntos de la memoria. Dichas partes o funcionalidades de la aplicación fueron desarrolladas paralelamente en archivos separados con el fin de que su elaboración fuera más sencilla y también de simplificar las cosas a la hora de buscar problemas y soluciones en la programación sin perdernos en un sinfín de líneas de código. Así pues, tenemos tres archivos:

- **mobesensvis.jspx:** donde se integran el mapa y el cuadro de medidas.
- **historicChart.jspx:** en el que se crean los gráficos históricos, así como distintos menús para que el usuario pueda seleccionar los parámetros de creación del gráfico histórico.
- **profileChart.jspx:** en el que se crean los gráficos de perfil, así como distintos menús para que el usuario pueda seleccionar los parámetros de creación del gráfico de perfil.

En la Figura 3 , podemos ver un esquema del diseño de la aplicación.

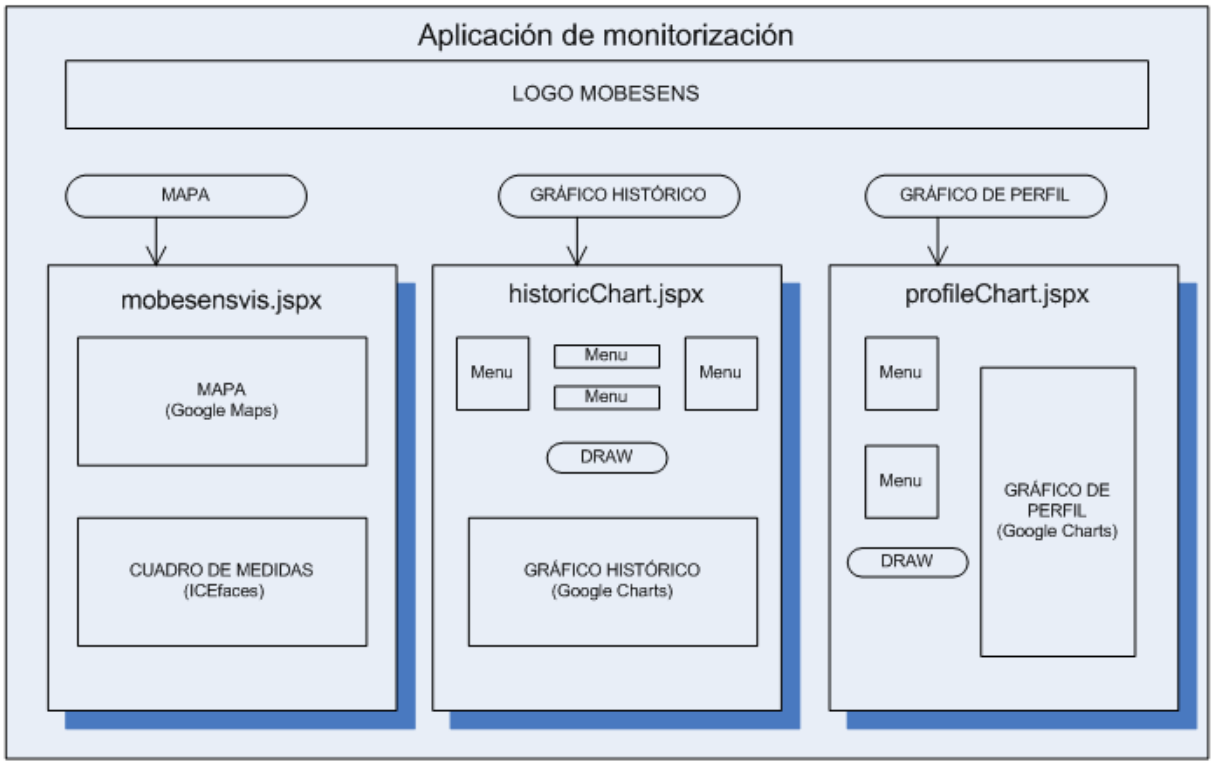


Figura 3: Esquema de diseño de la aplicación de monitorización

El archivo mobesensvis.jspx se propone como la página principal de la aplicación, pudiéndose acceder a la creación de los gráficos tanto históricos como de perfil mediante un sistema de navegación de botones como el que podemos observar en la Figura 4.

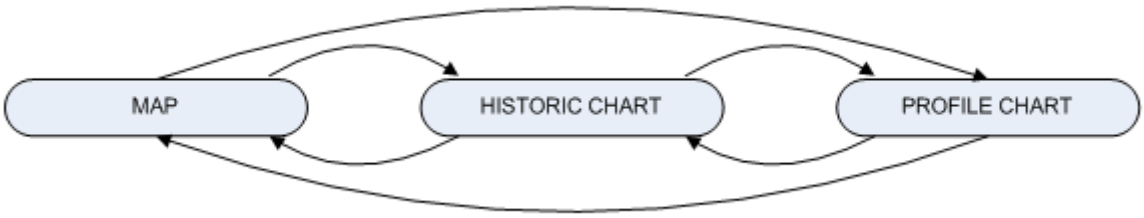


Figura 4: Diagrama de navegación de la aplicación de monitorización

Para el diseño puramente gráfico de la aplicación, como el Logo de Mobesens, los botones, los formularios o menús, la posición de cada elemento en la aplicación y el aspecto general de la misma se ha usado una combinación de HTML(HyperText Markup Language) y CSS (Cascade Style Sheets).

4.1. MODELADO DEL MAPA

Para la creación del mapa donde visualizamos la posición de los sensores así como su movimiento, se ha hecho uso del API de Google Maps. Este API nos ofrece multitud de posibilidades para la creación y personalización de mapas páginas o aplicaciones web, se trata de una amplia biblioteca de clases y funciones javascript.

Para la integración del mapa en la aplicación se ha usado GMAPS4JSF [14]. Se trata de una librería que permite integrar completamente Google Maps con Java Server Faces (JSF). La encontramos como solución para poder ir actualizando el mapa conforme nos llegaban nuevos datos de los sensores, porque sino no podíamos. De esta forma, varía un poco la creación del mapa respecto a lo que aconsejan en el tutorial del API de Google Maps, pero nos permite escribir código adicional javascript para actualizar el mapa con las nuevas posiciones de los sensores.

A continuación en la Figura 5, vamos a mostrar un diagrama de clases de aquellas que hemos usado durante el proyecto. Incluiremos a su vez una descripción más detallada de las clases con sus atributos y métodos, las relaciones entre ellas y su función y objetivo en el sistema global.

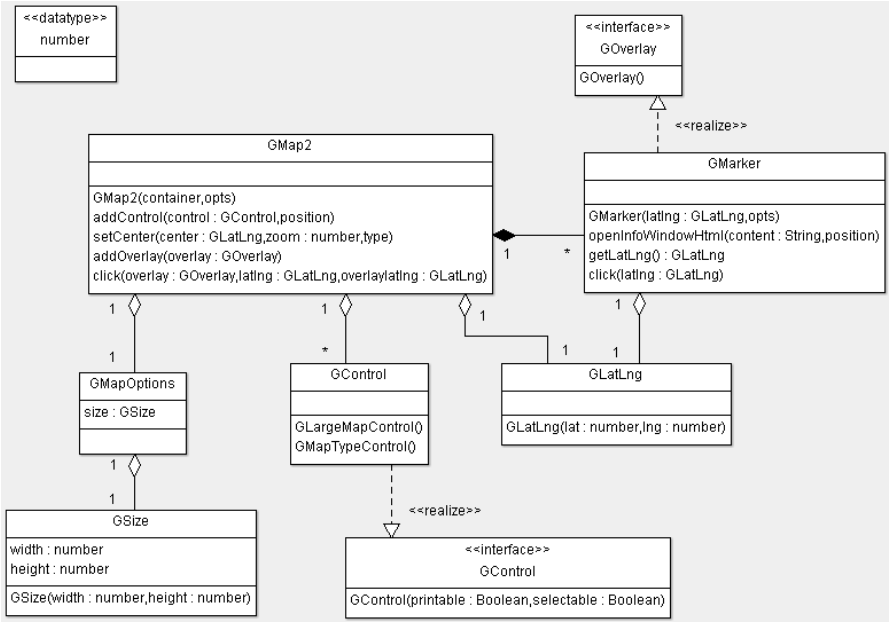


Figura 5: Diagrama de clases del mapa

Clase GMap2

Es la clase central del API, crea una instancia de la clase GMap2 para poder crear un mapa. No tiene atributos, sólo métodos y un constructor.

- **Constructor**
  → GMap2(container:Node, opts?: GMapOptions)
  Crea un nuevo mapa dentro del contenedor HTML en cuestión, que generalmente suele ser un elemento DIV. Podemos especificar diferentes características del mapa, como su tamaño pasando como argumento opcional un objeto de la clase GMapOptions. En nuestro caso, el uso de este constructor se hace a través de ciertas etiquetas de la librería gmaps4jsf.

- **Métodos**
  → setCenter(center: GLatLng, zoom?: number, type?: GMapType)
  Este método ha de utilizarse inmediatamente después de haber creado un mapa con el constructor, para definir su estado inicial. Define la vista del mapa de acuerdo al centro especificado y de manera opcional el nivel de acercamiento (zoom) y el tipo de mapa. No se deben ejecutar operaciones en un objeto GMap2 recién creado hasta que no se ejecute esta función.

  → addControl(control: GControl, position?: GControlPosition)
  Añade un control al mapa. La posición en el mapa viene determinada por el argumento opcional *position*. Si este argumento no está presente, se aplica la posición predeterminada del control. No se debe agregar más de una instancia del control al mapa.

→ addOverlay(overlay: GOverlay)
Añade una superposición al mapa, por ejemplo un marcador.

→ click(overlay: GOverlay, latlng: GLatLng, overlaylatlng: GLatLng)
Este evento se activa cuando el usuario hace clic en el mapa con el ratón. Un evento click transmite distintos argumentos en función del contexto en el que se ha hecho clic y de si se ha hecho clic en una superposición habilitada para clic. Si el usuario hace clic en una superposición clickable (como GMarker), el argumento overlay contendrá el objeto de la superposición, mientras que el argumento overlaylatlng contendrá las coordenadas de la superposición en la que se ha hecho clic. Además, también se activará un evento click en la misma superposición.

Clase GMarker

Marca una posición en el mapa. Implementa la interfaz de GOverlay y después se agrega al mapa mediante el método GMap2.addOverlay(). Los objetos de marcador tienen elementos latlng correspondientes a la posición geográfica en la que los marcadores se anclan en el mapa. Tras agregarse a un mapa, la ventana de información se podrá abrir a través del marcador. El objeto marcador activará eventos de ratón y eventos de ventana de información.

- **Constructor**
  → GMarker(latlng: GLatLng, opts?: GMarkerOptions)
  Crea un marcador en las coordenadas indicadas en el argumento latlng.

- **Métodos**
  → getLatLng(): GLatLng
  Devuelve las coordenadas geográficas en las que se ancla el marcador.

  → click(latlng: GLatLng)
  Este evento se activa cuando se ha hecho clic en el icono del marcador, transmitiendo las actuales coordenadas del marcador dentro de su argumento latlng. Este evento irá también dirigido al mapa, pasando el marcador como primer argumento a su gestor de eventos.

  → openInfoWindowHtml (content: String, opts?: GInfoWindowOptions)
  Abre la ventana de información sobre el icono del marcador. El contenido de la ventana de información se indica como una cadena que contiene texto HTML.

Clase GLatLng

Es un punto de acuerdo a una longitud y una latitud de coordenadas geográficas. La coordenada de latitud siempre se escribe primero (es decir, como primer argumento), seguida por la longitud.

- **Constructor**
  → GLatLng(lat: Lumber, lng: number)
  Construye un objeto de la clase GLatLng como acabamos de comentar, a partir de una latitud y una longitud determinadas.

Clase GMapOptions

Representa argumentos opcionales para el constructor de GMap2. Aunque no tiene constructor, se crea instancia como un objeto literal.

- **Atributos**
  → size: GSize
  Define el tamaño del mapa en píxeles. El tamaño del contenedor que se pasa al constructor del mapa se cambiará de acuerdo con el tamaño que se indique. De forma predeterminada, el mapa asumirá el tamaño de su contenedor.

Clase GSize

Corresponde al tamaño en píxeles de un área rectangular del mapa.

- **Atributos**
  → width: number
  Anchura, definida como una diferencia de puntos en la coordenada x.
  → height: number
  Altura, definida como una diferencia de puntos en la coordenada y.

- **Constructor**
  → GSize(width: Lumber, height: number)
  Construye un objeto de la clase GSize como acabamos de comentar, a partir de una anchura y una altura determinadas.

Clase GControl

Implementa la interfaz GControl. Pone a nuestra disposición varios constructores, de los cuales hemos utilizado los siguientes:

- **Constructor**

  → GLargeMapControl()

Crea un control con botones para hacer desplazamientos en cuatro direcciones y para acercar y alejar la imagen, así como una barra deslizante para usar el acercamiento.

→ GMapTypeControl()

Crea un control de tipo de mapa estándar para seleccionar distintos tipos de mapas admitidos (normal, satélite y híbrido) y pasar de uno a otro a través de botones.

Interfaz GControl

Esta interfaz la implementan todos los controles. En contraste con las superposiciones, que se colocan en relación al mapa, los controles se colocan en relación a la vista del mapa, es decir, que no se mueven cuando se mueve el mapa.

- **Constructor**

  → GControl(printable?: Boolean, selectable?: Boolean)

  Crea la instancia prototipo para una clase de control nueva. La marca *printable* indica que el control debe estar visible en el mapa impreso. La marca *selectable* indica que el control contendrá texto seleccionable.

Interfaz GOverlay

Esta interfaz la implementan varias clases del API de Google Maps, de las cuales la que nos interesa es GMarker. Se puede poner en el mapa una instancia de GOverlay con el método GMap2.addOverlay(). El mapa llamará entonces al método GOverlay.initialize() en la instancia de la superposición para mostrarse a sí mismo inicialmente en el mapa.

- **Constructor**

  → GOverlay()

  Este constructor crea implementaciones simuladas de los métodos. Aún así, al heredar de esta clase, el constructor de clase derivado deberá llamar a este constructor para poder terminar su trabajo.

4.2. MODELADO DEL CUADRO DE MEDIDAS

La integración del cuadro de medidas en la aplicación de monitorización se lleva a cabo mediante el marco de trabajo ICEfaces que nos proporciona una suite de componentes, de los cuales usamos los siguientes en la realización del cuadro de medidas:

- Form: crea un elemento HTML de tipo “form”.
- DataTable: crea un elemento HTML de tipo tabla. Permite mostrar objetos de una colección, siendo cada objeto una fila y cada columna un atributo o variable del objeto. Esto se consigue mediante los atributos de iteración de patrones JSF “value” y “var” e introspección. En nuestra aplicación cada fila será un sensor y cada columna corresponderá a diferentes informaciones del sensor incluyendo su identificador, tipo, valores de sus medidas y coordenadas geográficas.
- Column: es el componente de que define una columna dentro del dataTable.
- Facet: es el componente que nos permite definir un título para cada columna, es decir, describir el contenido de cada columna. Además nos permite colocarlo ya sea en la parte superior de la tabla o en la inferior.
- OutputText: presenta el valor especificado en el atributo “value”. En el cuadro de medidas se utiliza para mostrar los valores de las celdas del cuadro provistos por el Managed Bean del Gestor de Datos.
- InputText: presenta un elemento HTML “input” de tipo texto. Lo utilizamos en modo “invisible” (no se muestra en el cuadro de medidas) para las coordenadas de los sensores a cuyos valores accedemos en las funciones JavaScript que nos permiten cargar y actualizar los sensores en el mapa.

4.3. MODELADO DE LOS GRÁFICOS

Para crear los gráficos tanto de tipo histórico como de perfil y a la hora de integrar dichos gráficos en nuestra aplicación de visualización, se ha utilizado el API de Google Charts, que consiste en una biblioteca de clases y funciones JavaScript que nos permite crear y personalizar una amplia variedad de gráficos en páginas o aplicaciones web. El primer paso en la creación de un gráfico es rellenar el objeto `dataTable` a partir del cual se crea la visualización elegida (que ha tenido que ser cargada previamente desde Google Charts). También se crean una serie de formularios o menús para que el usuario pueda seleccionar los parámetros con los que se va a dibujar el gráfico.

4.3.1. Visualizaciones

Para la creación de los gráficos históricos, en un principio se pensó y se implementó la visualización `LineChart`, pero más adelante se cambio a la `AnnotatedTimeLine`, porque se vio que era más interactiva y nos daba más posibilidades acordes a nuestras necesidades. Por tanto, 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. Este gráfico se visualiza en el navegador del usuario mediante Flash.

Uno de los pasos importantes en la creación del gráfico, es rellenar el objeto **`dataTable`**. En el caso concreto de la visualización **`AnnotatedTimeLine`** podemos representar tantas líneas (en nuestro caso cada línea corresponderá a una medida del sensor) como queramos. Cada fila en el `dataTable` representa una posición "x" en el gráfico, es decir cada fila contendrá los valores de todos los parámetros diferentes que se quieran dibujar para un instante de tiempo concreto. El número de columnas a crear es variable y depende del número de líneas que se quieran visualizar en el gráfico. La primera de las columnas siempre es de tipo `date` o `datetime` y especifica el valor "x" del punto en el gráfico (es decir, es la que se encarga de representar el tiempo). Así mismo, cada línea estará descrita por una serie de columnas que pueden ir desde una hasta tres, aunque nosotros sólo utilizaremos una en la que se describe el valor de la línea para el correspondiente instante de tiempo de la primera columna.

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. Esta pequeña diferencia, nos supuso en su momento un pequeño problema al intentar hacer la visualización, ya que ahora tenemos en el eje "x" de la gráfica los parámetros ha representar y en el eje "y" la variable independiente que en este caso es la profundidad. Esto parece poca cosa, sin embargo, en un principio no encontrábamos un tipo de visualización que nos permitiera hacer eso, puesto que las visualizaciones `AnnotatedTimeLine` y `LineChart` no servían.

La elección final fue la visualización **`scatterChart`** del API de Google Charts, que está pensada para mapear la correlación entre conjuntos de números, representada mediante puntos. Al principio, esta opción no parecía la adecuada, puesto que nosotros necesitamos dibujar líneas para describir la evolución de las medidas en función de la profundidad, sin embargo, encontramos la solución en el atributo `lineSize` de la visualización `scatterChart` que nos permite incluir líneas que unen los puntos (si el valor de este atributo es 0 sólo se muestran los puntos, a partir de 1 se dibujan líneas más gruesas cuanto mayor sea el valor del parámetro).

Para rellenar el `dataTable` que nutre esta visualización, se requieren dos o más columnas; la primera contiene los valores usados para el eje "x" y las siguientes los valores del eje "y". Cada una de las columnas se muestra en un color diferente.

A continuación mostramos un diagrama en la Figura 6, con todas las clases importantes usadas en el modelado de los gráficos, así como una descripción más detallada de cada una de las clases.

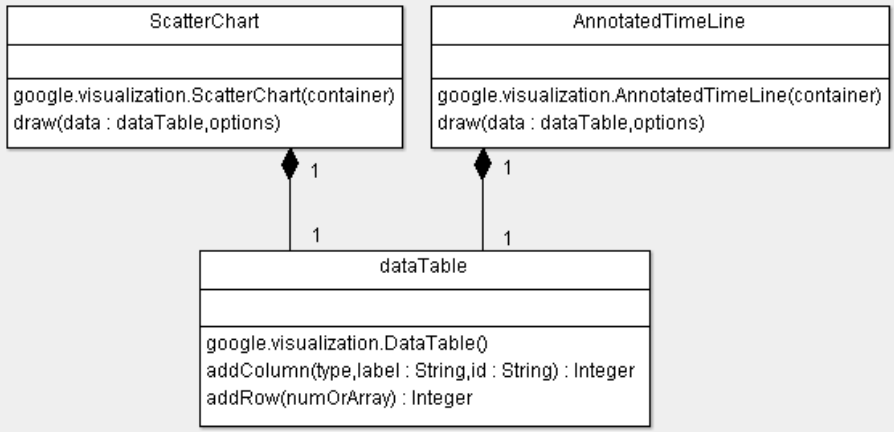


Figura 6: Diagrama de clases de los gráficos

Clase ScatterChart

Es un tipo de visualización que mapea la correlación entre conjuntos de datos. El gráfico se visualiza en el navegador usando SVG o VML. El formato de los datos ha de ser el siguiente: se requieren dos o más columnas que han de ser de carácter numérico. Los valores de la primera columna serán los correspondientes al eje X, los valores de la siguientes columnas se usarán en el eje Y (cada columna se mostrará con un color diferente).

- **Constructor**
  → google.visualization.ScatterChart(container)
  Crea un nueva visualización de tipo ScatterChart dentro del contenedor HTML en cuestión, que generalmente suele ser un elemento DIV.

- **Métodos**
  → draw(data: dataTable, options?)
  Dibuja el gráfico a partir de los datos que se le pasan en el parámetro data (el formato del *dataTable* tendrá que ser el adecuado para este tipo de gráfico) y con la configuración que le pasemos a través del parámetro opcional *Options* . Estas opciones se pasan con la sintaxis literal de los objetos javascript.

Clase AnnotatedTimeLine

Es un tipo de gráfico interactivo que representa líneas de datos con dependencia del tiempo. El gráfico se visualiza en el navegador usando Flash. El formato de los datos ha de ser el siguiente: se pueden mostrar una o más líneas en el gráfico. Cada fila representa una posición X en el gráfico, es decir para cada instante de tiempo tenemos uno o varios datos en una fila y tendremos tantas filas como puntos en el gráfico queramos. La primera columnas será siempre de tipo date o datetime y

especifica el valor X del punto en el gráfico. Luego cada línea se describe con tres columnas adicionales (las dos últimas son opcionales), de las cuáles la primera es la importante, ya que es el valor del eje Y, es decir el valor de la línea en el instante correspondiente a la primera columna.

- **Constructor**

→ google.visualization.AnnotatedTimeLine(container)

Crea un nueva visualización de tipo AnnotatedTimeLine dentro del contenedor HTML en cuestión, que generalmente suele ser un elemento DIV.

- **Métodos**

→ draw(data: dataTable, options?)

Dibuja el gráfico a partir de los datos que se le pasan en el parámetro data (el formato del *dataTable* tendrá que ser el adecuado para este tipo de gráfico) y con la configuración que le pasemos a través del parámetro opcional *Options* (cada tipo de visualización tiene unas opciones diferentes) . Estas opciones se pasan con la sintaxis literal de los objetos javascript.

Clase DataTable

Representa una tabla de valores cambiantes de dos dimensiones.

- **Constructor**

→ google.visualization.DataTable ()

Crea una instancia vacía de DataTable, que podemos poblar mediante los metos *addColumn* y *addRow*.

- **Métodos**

→ addColumn(type, label: String, id: String): number

Añade una nueva columna al dataTable y devuelve el índice de la nueva columna. El parámetro *type* define el tipo de datos que contiene la columna (puedes ser String, Number, Boolean, Date, DateTime o Timeofday), el parámetro *label* es la etiqueta de la columna y *id* su identificador.

→ addRow(numOrArray): number

Añade una nueva fila al dataTable y devuelve el índice de la nueva fila. El parámetro *numOrArray* puede ser un único valor o un vector de valores, que poblarán las tabla.

4.3.2. Formularios y XMLHttpRequest

Por otra parte, necesitamos un mecanismo para saber que parámetros o medidas quiere representar el usuario en su gráfico, así como el intervalo de tiempo que quiere visualizar. Para ello se ha programado un formulario en HTML, que contiene varios campos, para que el usuario pueda introducir el intervalo de tiempo y seleccionar las medidas que quiere representar (el campo de los parámetros permite

seleccionar más de una medida, para que varias puedan ser representadas en el mismo gráfico). Estas medidas incluyen para el gráfico histórico: la temperatura, el pH, la conductividad, el nivel de oxígeno y la turbidez del agua. Además se cuenta con una función JavaScript que nos permite recoger las opciones elegidas por el usuario y usarlas para realizar la consulta correspondiente a la base de datos mediante el lenguaje xQuery.

Sin embargo, hay que tener en cuenta algo muy importante y es que hay que esperar a tener el resultado de la consulta a la base de datos para poder crear el gráfico; sino se producen problemas, porque la información de la que se nutre la visualización puede no estar disponible en el momento de cargar el gráfico. Para solucionar este problema, se hace uso de **XMLHttpRequest**. Se trata de una interfaz empleada para realizar peticiones http a servidores Web. Como formato para los datos transferidos se puede usar cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON, HTML y codificaciones particulares específicas.

La interfaz se representa como una clase, para utilizarla, la aplicación cliente debe crear una nueva instancia mediante el constructor adecuado. Este constructor se llama `createXhrObject()`. Es posible realizar peticiones asíncronas al servidor, esto se consigue definiendo una función que se ejecutará cuando se complete la petición, esta función es `onreadystatechange`; poniendo en su interior las líneas de código necesarias para que el navegador cree y dibuje nuestro gráfico, nos aseguramos de que sólo lo haga en el momento en el que la consulta a la base de datos ya haya obtenido un resultado y por lo tanto contemos con la información necesaria y no haya problemas. Otro método del objeto XMLHttpRequest necesario para que la petición en sí sea enviada es el método `open` en el que se especifican como parámetros: el modo de solicitar los datos, que puede ser mediante “GET” (implica que los parámetros de la petición se codifican en la URL) o “POST” (implica que los parámetros de la petición se codifican en las cabeceras de Http), en nuestro caso usamos “GET”; el siguiente parámetro es la URL a la que se envía la petición y por último se especifica un valor que será `true` si queremos que la petición se gestione asíncronamente o `false` si queremos que se haga de forma síncrona.

En la Figura 7, podemos ver el diagrama de la clase XMLHttpRequest y a continuación se detallan sus métodos y atributos.

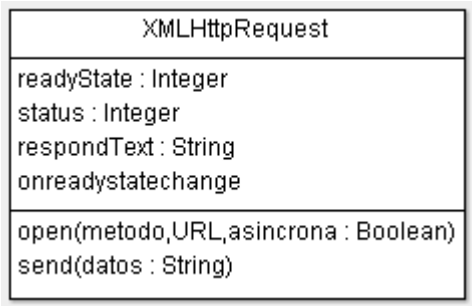


Figura 7: Diagrama de la clase XMLHttpRequest

Clase XMLHttpRequest

Nos permite la transferencia de datos en formato XML o texto plano desde los scripts del navegador (JavaScript) a los del servidor (Java) e inversamente.

- **Atributos**
  → readyState: Integer
  Devuelve el estado actual del objeto *XMLHttpRequest*, cada vez que cambia el valor de *readyState*, se lanza la función indicada en *onreadystatechange*.
  → status: Integer
  Devuelve el código del estado HTTP de la transmisión devuelto por el servidor web. Es conveniente comprobar este dato antes de acceder a los datos mediante *respondText*.
  → responseText: String
  Devuelve el texto del documento (en texto plano) descargado del servidor en una petición con XMLHttpRequest.
  → onreadystatechange: function()
  Asigna la función que se ejecutará cada vez que *readyState* cambie de valor.

- **Métodos**
  → open(método, URL, asíncrono)
  Prepara una conexión HTTP a través del objeto *XMLHttpRequest* (con un método y una URL especificados) e inicia todos los atributos del objeto. Para realizar la conexión deberemos usar el método *send* después de *open*.
  → send(datos)
  Envía la petición con los datos pasados por parámetro como cuerpo de la petición a través del objeto *XMLHttpRequest*. El parámetro *datos* puede ser una referencia al DOM de un documento o una cadena de caracteres.

4.3.2. Consultas xQuery

Como ya hemos comentado el lenguaje de consulta usado para interrogar a la base de datos es xQuery.

A continuación, en la Figura 8, podemos ver un ejemplo de las consultas xQuery que utilizamos para la creación de los gráficos en la aplicación de monitorización:

```
for $b in collection('/db/tsp/rs2m/netinf/readings/99')//Observation
where number($b/GSample)=-964
return concat('data3.addRow([',replace($b/Temperature/value/text(),
',', '.'),',', -(replace($b/Depth/value/text(), ',', '.')),']);')
```

Figura 8: Ejemplo de consulta xQuery

La variable "$b" es el índice o contenedor de lo que buscamos (si pusiéramos return $b la consulta nos devolvería todo el documento XML). En la expresión de la Figura 8, se utilizan las expresiones:

- FOR: en la que tenemos que especificar la localización del directorio donde se van a buscar los documentos y el elemento raíz de los documentos XML. En este caso los documentos XML utilizados para los gráficos de perfil están en el directorio 99 y el elemento raíz de todos ellos es Observation.
- WHERE: para filtrar la información que nos interese en cada caso. En concreto en esta consulta se filtra la información para que sólo nos muestre los datos referentes a una muestra concreta.
- RETURN: devuelve como resultado de la consulta lo especificado esta cláusula. En concreto las utilizadas en estas consultas son un poco especiales pues se han hecho de forma que el resultado sea directamente el código necesario para crear el dataTable solicitado por Google Charts para dibujar la visualización.

En la expresión RETURN se hace uso de las funciones concat y replace que sirven respectivamente para concatenar cadenas de caracteres y para remplazar un carácter concreto en un elemento de la consulta por otro carácter distinto (esta

función se usa para cambiar las comas de los valores decimales por puntos, ya que así lo requiere Google Charts).

En la Figura 9, podemos ver el resultado de realizar la consulta de la Figura 8, directamente en eXist-DB mediante su xQuery SandBox (es una especie de playground en el que puedes hacer consultas sobre tus colecciones). Se comprueba que los resultados obtenidos son los esperados.

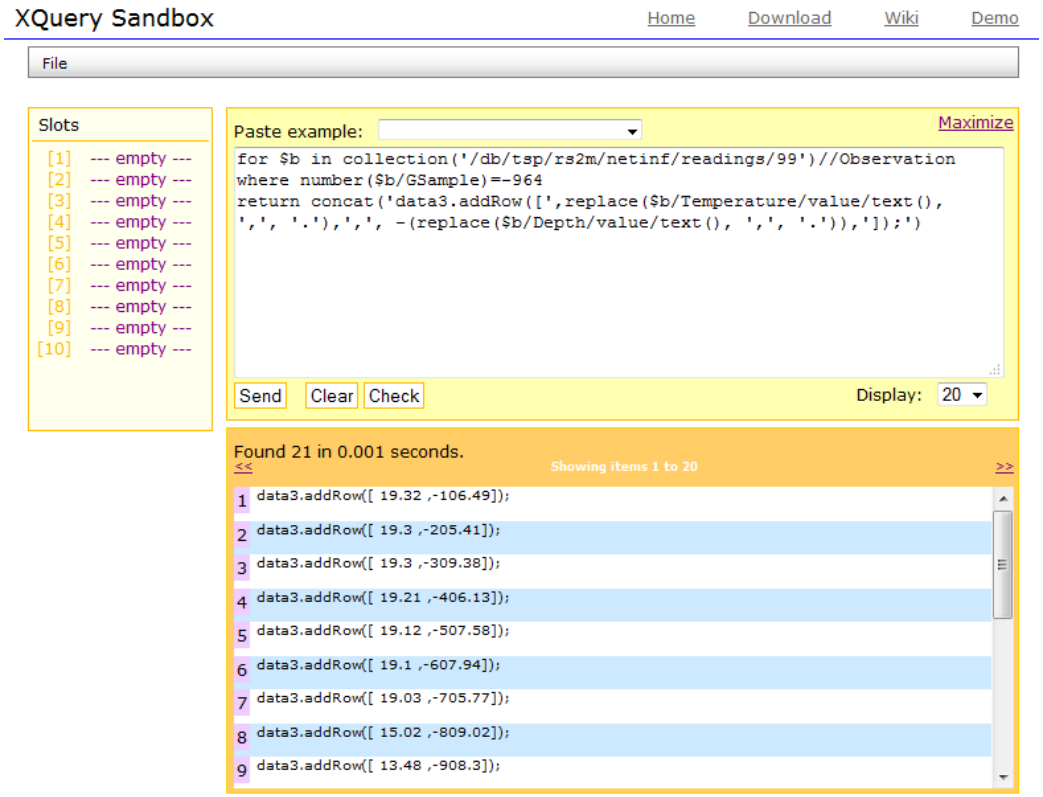


Figura 9: xQuery Sandbox de eXist-DB

Sin embargo, cuando realizamos las consultas desde la aplicación hay que añadirles algunas expresiones para que NetInf las trate correctamente, como podemos observar en la Figura 10, en la que se han añadido dos líneas de código a la consulta, una al principio y otra al final.

```
http://localhost:8080/NetInf/api/rest/light-mobesens/search?query=
for $b in collection('/db/tsp/rs2m/netinf/readings/99')//Observation
where number($b/GSample)=-964
return concat('data3.addRow([',replace($b/Temperature/value/text(), ',',
'.'),',', -(replace($b/Depth/value/text(), ',', '.')),']);')
&collection=/db/tsp/rs2m/netinf/readings/99
```

Figura 10: Consulta xQuery desde la aplicación

1.5. 5. GESTOR DE DATOS

El Gestor de Datos está compuesto por una serie de clases que nos permiten describir la información de los sensores y que se comunica por un lado con la base de datos mediante el servicio de mensajes java (JMS) y por otro lado con la aplicación de monitorización pasándole los datos que nutren el cuadro de medidas mediante los Managed Beans.

En la Figura 11 se muestra el diagrama de clases del Gestor de Datos y a continuación la descripción de los atributos y métodos de cada una de estas clases.

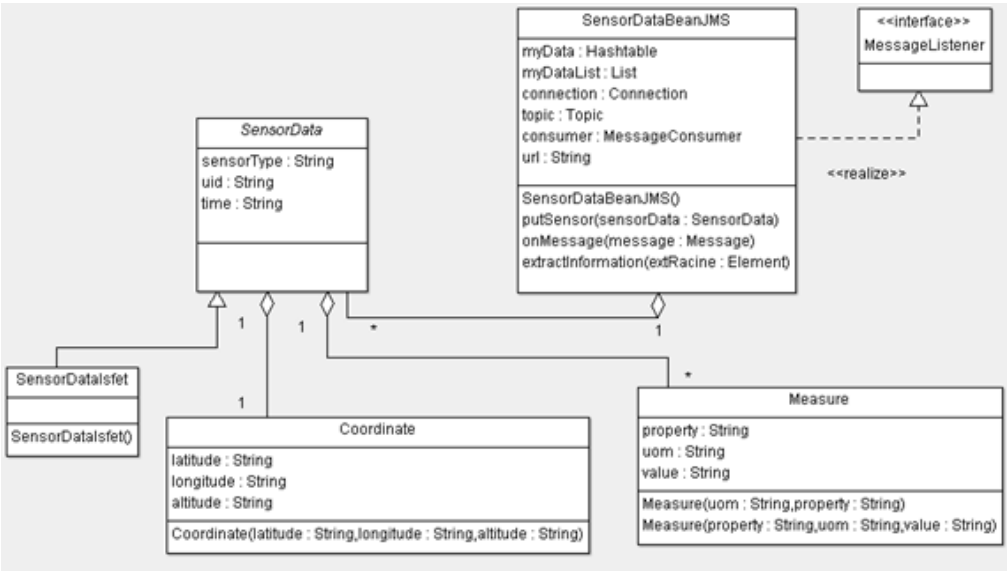


Figura 11: Diagrama de clases del gestor de datos

Clase Coordinate

Coordenadas de un punto geográfico (de un sensor).

- **Atributos**
  → latitude: String
  Latitud de la coordenada geográfica.
  → longitude: String
  Longitud de la coordenada geográfica.

→ altitud: String
Altitud de la coordenada geográfica (0 sería el nivel del mar)

- **Constructores**
  → Coordinate(latitude: String, longitude: String, depth: String)
  Crea una instancia de la clase Coordinate con los atributos que se le pasan como parámetro.

Clase Measure

Medida de un sensor.

- **Atributos**
  → property: String
  Tipo de propiedad que está midiendo el sensor.
  → uom: String
  Unidad de medida de la propiedad.
  → value: String
  Valor de la medida.

- **Constructores**
  → Measure(property: String, uom: String)
  Crea una instancia de la clase Measure con los atributos que se le pasan como parámetro (sólo el tipo de propiedad y la unidad de medida).
  → Measure(property: String, uom: String, value: String)
  Crea una instancia de la clase Measure con los atributos que se le pasan como parámetro (tipo de propiedad, unidad de medida y valor de la medida).

Clase SensorData

Es una clase abstracta que modela toda la información referente a un sensor.

- **Atributos**
  → sensorType: String
  Tipo de sensor.
  → uid: String
  Identificador único del sensor.
  → time: String
  Instante en el que se ha tomado la medida.
  → measure: Measure []
  Vector de medidas.
  → coordinate: Coordinate
  Coordenadas del sensor.

Clase SensorDataBean

Se trata de un managed bean donde se crean los diferentes sensores y que contiene los métodos para comenzar y parar la simulación.

- **Atributos**
  → threads[]: Thread_isfet_push
  Vector que contiene los diferentes sensores (o más bien los hilos de ejecución de los sensores).

- **Métodos**
  → createTreads()
  Crea hilos de ejecución para 4 sensores diferentes.
  → start(event: ActionEvent)
  Pone en marcha los hilos de ejecución.
  → stop(event: ActionEvent)
  Para los hilos de ejecución.

Clase SensorDataIsfet

Esta clase hereda de SensorData. Es básicamente para construir los sensores (ya que SensorData es una clase abstracta y no puede tener constructores).

- **Constructores**
  → SensorDataIsfet(uid: String, time: String, mTemperature: String, Mph: String, mCa: String, Mnh4: String, Mno3: String, latitude: String, longitude: String, altitude: String)
  Crea el sensor con todos los atributos que le pasamos como parámetros y especifica que el tipo de sensor es “ISFET-Ta205”.

1.6. *6. SIMULADOR*

Su función es simular el funcionamiento en tiempo real de la aplicación de monitorización, como se ha explicado previamente en la sección 4.1 de la memoria del PFC.

En la Figura 12, se presenta el diagrama de clases del Simulador, incluyendo una descripción detallada de los métodos y atributos de cada una de sus clases.

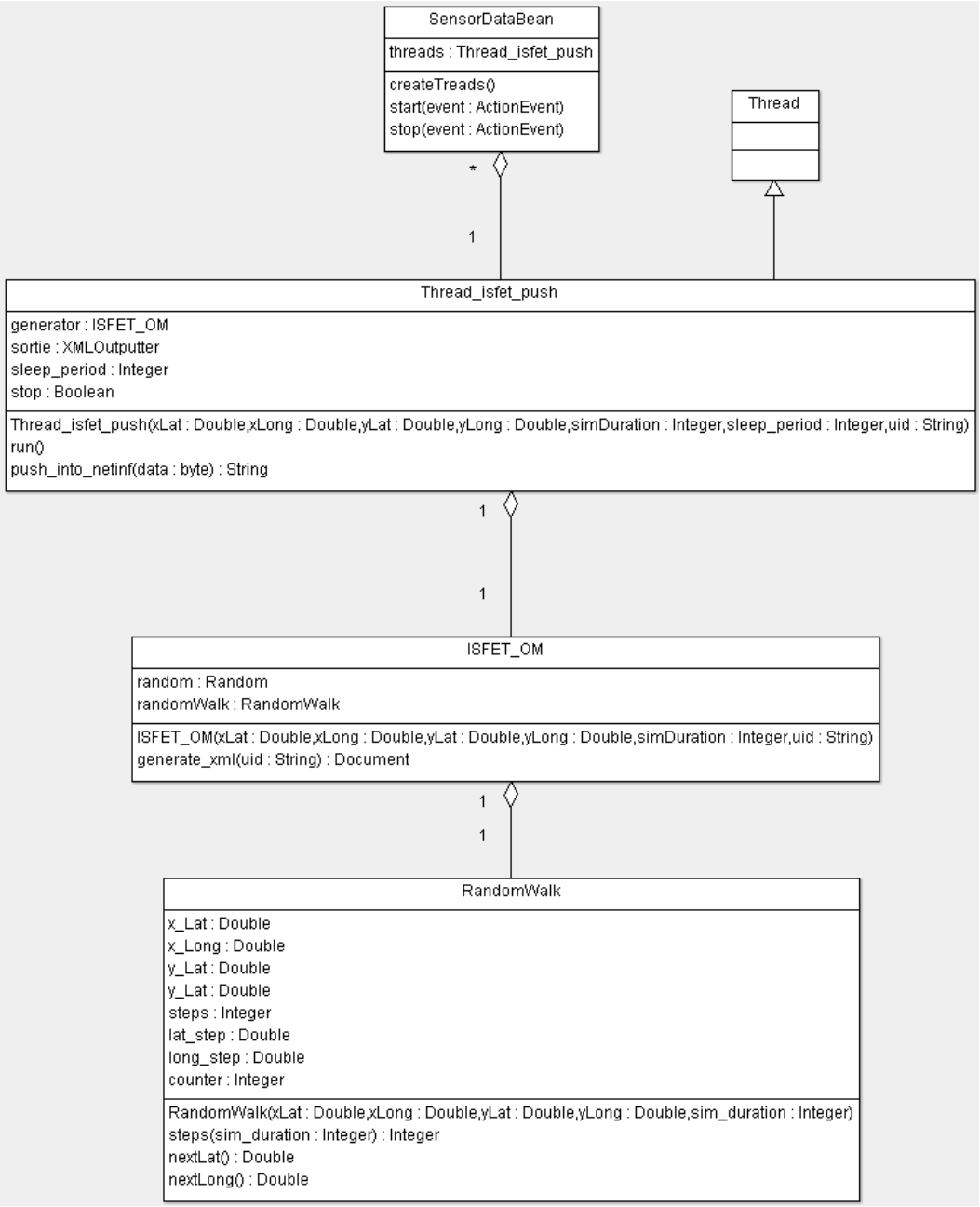


Figura 12: Diagrama de clases del simulador

Clase SensorDataBeanJMS

Se trata de un managed bean que contendrá la lista con todos los sensores, a su vez es la clase donde se inicia la conexión al servicio JMS.

- **Atributos**
  → myData: Hashtable <String, SensorData>
  Hashtable para almacenar la información de los sensores, donde la clave será el identificador del sensor.

→ myDataList : List
Lista sincronizada de los sensores.
→ connection : Connection
Conexión con el proveedor JMS.
→ session: Session
Sesión para producir y consumir mensajes. Se crea a partir de una conexión.
→ topic : Topic
Tema del intercambio de mensajes.
→ consumer: MessageConsumer
Es un objeto que se crea a partir de una sesión y sirve para recibir mensajes.
→ url: String
URL del servidor JMS.

- **Constructor**
→ SensorDataBeanJMS()
Se crea la conexión al proveedor JMS y se comienza la comunicación.

- **Métodos**
→ putSensor(sensorData: SensorData)
Actualiza la lista de sensores.
→ onMessage(message: Message)
Función que se ejecuta cuando llega un mensaje JMS. Implementa la interfaz MessageListener. Crea un objeto Document a partir del mensaje recibido para poder acceder a los elementos a partir de la raiz del documento.
→ extractInformation(extRacine: Element)
Inserta en la lista de sensores un nuevo sensor a partir de los datos recibidos en el mensaje JMS.

Clase RandomWalk

Es la clase con la que imprimimos movimiento a los sensores en la simulación. Se programa un "paseo" del sensor entre dos coordenadas geográficas.

- **Atributos**
→ x_Lat: double
Latitud en la que empieza el paseo.
→ x_Long: double
Longitud en la que empieza el paseo.
→ y_Lat: double
Latitud en la que acaba el paseo.
→ y_Long: double
Longitud en la que acaba el paseo.
→ lat_step: double
Paso del movimiento (en latitud).

→ long_step: double
Paso del movimiento (en longitud).
→ long_step: double
Paso del movimiento (en longitud).
→ steps: int
Número de pasos que va a tener la simulación
→ counter: int (inicializado a 0)
Contador que sirve para ir calculando las nuevas posiciones del sensor.

- **Constructor**
→ RandomWalk(xLat: double, xLong: double, yLat: double, yLong: double, sim_duration: int)
Crea un objeto de la clase RandomWalk con los puntos de comienzo y finalización del paseo y la duración de la simulación.

- **Métodos**
→ steps (sim_duration: int): int
Nos devuelve el número de pasos que tendrá la simulación.
→ nextLat(): double
Devuelve la latitud de la siguiente posición del sensor.
→ nextLong(): double
Devuelve la longitud de la siguiente posición del sensor.

Clase Thread_isfet_push

Hereda de la clase Thread. Es por tanto un hilo de ejecución customizado para la simulación.

- **Atributos**
→ generator: ISFET_OM
Almacena el objeto generador del documento XML para cada sensor.
→ sortie: XMLOutputter
Almacena un documento de tipo JDOM como una secuencia de bytes.
→ sleep_period: int
Tiempo de pausa de los thread. Está inicializado a 500 ms.
→ stop: boolean
Decide si hay que generar y enviar los documentos o no.

- **Constructor**
→ Thread_isfet_push(xLat: double, xLong: double, yLat: double, yLong: double, sim_duration: int, sleep_period: int, uid: String)
Crea un thread para un sensor.

- **Métodos**
→ run()
Método que se ejecuta cuando empieza el thread.

→ push_into_netinf(data: byte[]): String

Envia el documento XML en forma de secuencia de bytes a NetInf (quien a su vez lo envia mediante el servicio JMS a SensorDataBeanJMS).

Clase ISFET_OM

Almacena los datos para la simulación del movimiento de un sensor.

- **Atributos**
  → random: Random
  Número aleatorio.
  → randomWalk: RandomWalk
  Almacena el paseo de un sensor.

- **Constructor**
  → ISFET_OM(xLat: double, xLong: double, yLat: double, yLong: double, sim_duration: int, uid: String)
  Crea un objeto de tipo ISFET_OM.

- **Métodos**
  → generate_xml(uid: String): Document
  Genera el documento XML con los datos del sensor.

1.7. 7. COMPORTAMIENTO

En esta sección vamos a explicar el comportamiento de la aplicación mediante diagramas de secuencia, en lo que se refiere a la interacción de los usuarios (tanto administrador como usuario final) con la aplicación.

En la Figura 13, vemos el diagrama de secuencia que describe las acciones que tiene que realizar el administrador para poner en funcionamiento todo el sistema:

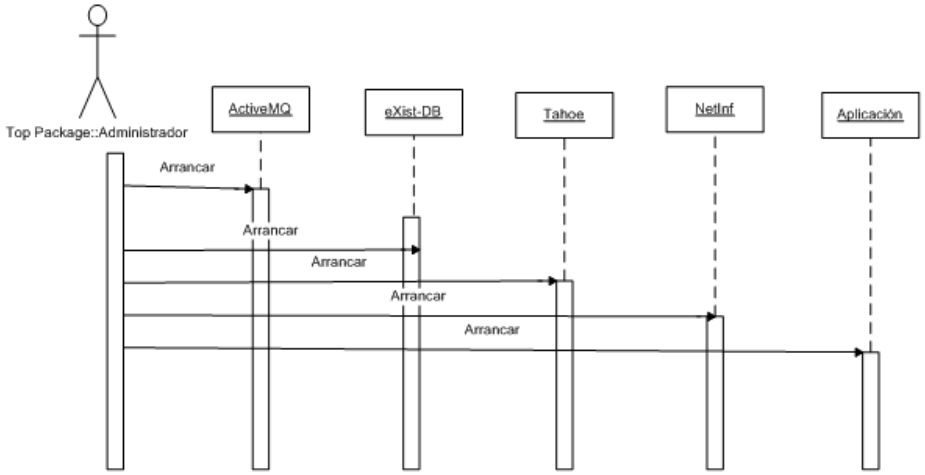


Figura 13: Diagrama de secuencia de arranque

Como vemos en la Figura 13, el administrador tiene que realizar una serie de tareas antes de que el usuario pueda acceder a la aplicación. En primer lugar tendrá que arrancar la instancia de ActiveMQ que nos permite la comunicación JMS entre NetInf y el Gestor de Datos. Luego tendrá que activar eXist-DB y Tahoe-LAFs para posibilitar el funcionamiento de NetInf que internamenta usa estas dos herramientas. Por último, el administrador deberá lanzar las instancias de NetInf y de la Aplicación de monitorización.

Por otra parte, comentamos el funcionamiento del simulador en lo que se refiere a comunicaciones entre las distintas partes. En la Figura 14, se presenta el diagrama de secuencia de la simulación.

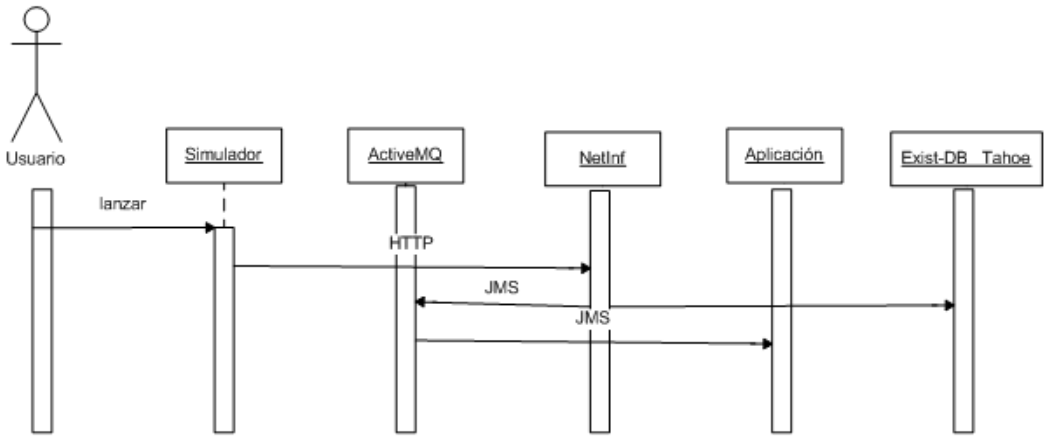


Figura 14: Diagrama de secuencia de la simulación

En primer lugar para que empiece la simulación, el usuario tendrá que poner en marcha el Simulador mediante la interfaz gráfica que hemos comentado en el punto 3 de este Anexo. Una vez activado el **Simulador**, éste se dedica a crear nuevos datos para los sensores y enviárselos en forma de documentos XML a NetInf vía HTTP. Una vez los datos están en **NetInf,** internamente se introducen en la base de datos mediante **eXist-DB** y **Tahoe**. A su vez, NetInf envía los datos al proveedor del servicio de mensajes Java, **ActiveMQ**, del cual los recupera la **Aplicación** (en concreto el Gestor de Datos en la parte del servidor) ya que está suscrita a dicho servicio.

Se han realizado una serie de pruebas para comprobar que la aplicación de monitorización funciona correctamente y también para comprobar el buen funcionamiento del simulador y de su comunicación con la aplicación y la base de datos.

Se han llevado a cabo varios ensayos para ver cuanta información es capaz de crear y enviar el simulador, que nos permiten corroborar a su vez que tanto la base de datos como el proveedor del servicio de mensajes Java, ActiveMQ, están a la altura de las circunstancias y que podrán soportar la carga de datos esperada una vez el proyecto Mobesens esté funcionando a pleno rendimiento con los sensores físicos tomando y enviando sus medidas.

Para la realización de dichos ensayos se han tomado los siguientes parámetros fijos:

- Número de sensores: 4
- Duración de la simulación: 10 minutos
- Tamaño de los documentos XML: 4kbytes

Variando en cada caso la frecuencia con la que el simulador crea y envía los documentos XML que contienen las medidas de los sensores. En la tabla de la Figura 15 se presentan los resultados obtenidos.

Frecuencia de envío (s)	Número de XML enviados	Número de XML recibidos en la aplicación	Número de XML recibidos en la base de datos
120	20	20	20
90	28	28	28
60	40	40	40
30	76	76	76

Figura 15: Tabla de resultados

A la vista de los resultados podemos observar que efectivamente para cada frecuencia se envían y se reciben, tanto en la aplicación como en la base de datos, el número de documentos XML correctos, teniendo en cuenta que hay cuatro sensores diferentes y que la duración de las distintas simulaciones es de diez minutos. También hay que decir, que a la vista de las estampas de tiempo de cada documento XML recibido, se ha sacado la conclusión de que el tiempo de creación y envío de cada documento por parte del simulador es de aproximadamente 3 segundos.

Estos resultados son adecuados para la integración de la aplicación en el proyecto Mobesens, ya que se ha verificado que funciona correctamente con las anteriores simulaciones y se espera que en la fase final del proyecto Mobesens, cuando los sensores físicos estén emplazados en el medio acuático, éstos realicen medidas cada varios minutos y las envíen a la base de datos.

Por otra parte se ha comprobado que la aplicación de monitorización funciona correctamente. La actualización del mapa y el cuadro de medidas con las nuevas informaciones que van llegando se realiza casi instantáneamente. Las elecciones del usuario en cuanto a la creación de los gráficos son tratadas correctamente, ya que tanto las consultas a la base de datos como la propia visualización de los gráficos es coherente con estas elecciones. A su vez se ha constatado que el acceso a la base es prácticamente instantáneo, puesto que los gráficos aparecen en la pantalla inmediatamente después de hacer click en el botón “Draw Chart”. También se ha verificado que la navegación interna de la aplicación (el acceso a cada una de sus partes) se lleva a cabo según el diagrama de navegación propuesto en el diseño.

ANEXO B. CONTEXTO TECNOLÓGICO Y HERRAMIENTAS UTILIZADAS

En este Anexo se pretende dar una visión más detallada y completa de las tecnologías, lenguajes de programación y herramientas utilizadas en la realización del proyecto, así como las razones de su elección. Se empezará por las tecnologías seleccionadas para el desarrollo de aplicaciones web, luego se detallarán otras tecnologías empleadas en el proyecto y por último las herramientas utilizadas.

1.8. *1. Tecnologías para el desarrollo de aplicaciones web*

1.1. JAVASCRIPT

Se trata de un lenguaje de programación interpretado orientado a objetos. Fue desarrollado originalmente por Brendan Eich de Netscape y con el paso del tiempo ha pasado a ser un estándar de la ECMA.

Su uso principal es en aplicaciones del lado del cliente. Los scripts o programas JavaScript se integran en las páginas web junto al código HTML. Está provisto de una implementación del DOM, lo que nos permite cambiar dinámicamente el contenido estructurado de un documento HTML haciendo así las páginas web mucho más interactivas. En la actualidad todos los navegadores modernos tienen la capacidad de interpretar el código JavaScript incrustado en las páginas web.

En JavaScript, cualquier tipo de dato es un objeto constituido por una colección de atributos. Todos son extensiones del objeto base Object y cada uno tiene un prototipo que se puede extender con nuevos atributos.

En el proyecto se ha utilizado JavaScript [15] para la mayor parte de la aplicación de monitorización del lado del cliente, ya que tanto el API de Google Maps como el de Google Charts están escritos en JavaScript y además es inherente a la utilización de Ajax. También se utiliza para la validación de los formularios.

1.2. JAVA SERVER FACES (JSF)

Es una tecnología y framework para crear aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones J2EE. Es la tecnología que utiliza ICEfaces en conjunción con Ajax para crear la suite de componentes que nos hacen mucho más sencilla la creación de la aplicación de

monitorización. Además nos permite la utilización de los Managed Beans que nos dan acceso a las clases Java definidas como tales del simulador y del gestor de datos.

JSF tiene las siguientes características principales:

- Proporciona un conjunto de componentes para la interfaz de usuario, incluyendo los elementos estándares de HTML para representar un formulario. Estos componentes se obtendrán de un conjunto básico de clases base que se pueden utilizar para definir componentes nuevos.
- Proporciona un modelo de Managed Beans para enviar eventos desde los controles de la interfaz de usuario del lado del cliente a la aplicación del servidor.
- Utiliza la tecnología JSP (Java Server Pages) como la tecnología que permite hacer el despliegue de las páginas.
- Utiliza un sencillo fichero de configuración para el controlador en formato xml
- Es extensible, pudiendo crearse nuevos elementos de la interfaz o modificar los ya existentes.
- Y lo que es más importante: forma parte del estándar J2EE. En efecto, hay muchas alternativas para crear la capa de presentación y control de una aplicación web java, como Struts y otros frameworks, pero solo JSP forma parte del estándar.

1.3. AJAX (Asynchronous JavaScript and XML)

Se trata de una técnica de desarrollo web para crear aplicaciones interactivas. En realidad es una combinación de cuatro tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- Document Object Model (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interaccionar dinámicamente con la información presentada.
- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor.

En la Figura 16 podemos ver el modelo de una aplicación trabajando con Ajax. Al comienzo de la sesión, el navegador en vez de cargar la página web directamente, carga una máquina Ajax responsable por una parte de presentar el

interfaz gráfico al usuario y por otra de comunicarse con el servidor en nombre del usuario, de tal forma que la interacción del usuario con la aplicación se lleva a cabo asíncronamente, es decir independientemente de la comunicación con el servidor.

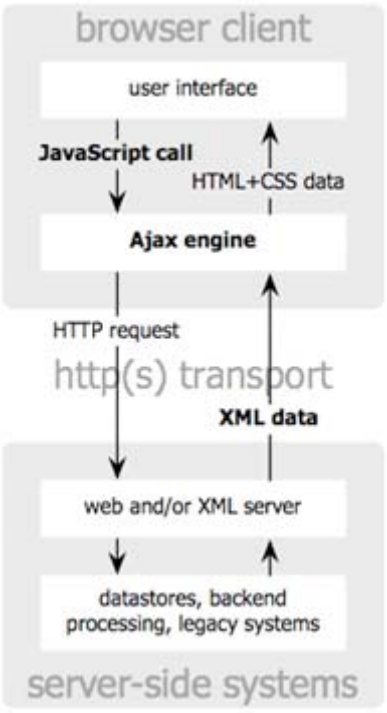


Figura 16: Modelo de aplicación usando Ajax

Por tanto, en vez de que la aplicación tenga que estar constantemente pidiendo información al servidor (como pasaría en una aplicación normal), la tecnología Ajax nos permite realizar cambios sobre las páginas sin necesidad de recargarlas. Con lo cual aumenta la interactividad, velocidad y usabilidad de las aplicaciones.

Para la creación de la aplicación de monitorización era necesario el uso de Ajax debido a la necesidad de visualizar la posición y medidas de los sensores en tiempo real. Aunque en realidad nosotros no hemos tenido que manejar esta tecnología puesto que el marco de trabajo ICEfaces nos aisla de gestionar el Ajax. Sin embargo si se ha utilizado la implementación más básica de Ajax que es el objeto XMLHttpRequest para la creación de los gráficos.

1.9. *2. Otras tecnologías*

2.1. JAVA

Se trata de un lenguaje de programación orientado a objetos. Fue desarrollado por Sun Microsystems a principios de los años 90 y en 2007 se liberó la mayoría de sus tecnologías bajo la licencia GNU GPL.

La idea de la programación orientada a objetos es diseñar el software de forma que sea reutilizable entre distintos proyectos. Para ello se usan unas entidades genéricas llamadas objetos que combinan datos y las operaciones que se pueden realizar sobre ellos (funciones o métodos). De esta forma separando las cosas que cambian de las que permanecen inalterables, se consigue una estructura de datos independiente y reutilizable.

La característica principal de Java que además constituye su premisa inicial es "Write Once, Run Anywhere", es decir, "Escríbelo una vez, ejecútalo en cualquier sitio". Esto se refiere a la capacidad de las aplicaciones o programas Java de ejecutarse en cualquier tipo de hardware, gracias a un lenguaje independiente de la plataforma y un entorno de ejecución, la Java Virtual Machine (JVM), ligero y gratuito para las plataformas más populares. El código fuente escrito en lenguaje Java se compila para generar un "bytecode" (código máquina simplificado y específico de la plataforma Java) que luego es ejecutado en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino que es el que interpreta y ejecuta el código.

La elección de Java como lenguaje para la aplicación del lado del servidor ya estaba hecha a mi llegada al proyecto Mobesens y se debe principalmente a las propias características del lenguaje nombradas anteriormente, así como la posibilidad que nos brinda de crear aplicaciones web dinámicas a través de las Java Server Pages. Además el hecho de que la base de datos está íntegramente programada en Java también ha sido un factor importante dada la interacción que la aplicación ha de tener con ésta.

Como entorno de programación Java se ha elegido Eclipse. Se trata de un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

Nosotros utilizaremos la versión de Eclipse llamada Galileo (versión 3.5) que salió el 24 de Junio de 2009.

2.2. XML (eXtensible Markup Language)

El lenguaje de marcado extensible (XML) es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C), que deriva de SGML (ISO 8879). No es un lenguaje en particular, sino una manera de definir lenguajes para necesidades específicas. Es un lenguaje muy similar a HTML pero su función principal es describir datos y no mostrarlos como es el caso de HTML, para ello intenta estructurar la información de la manera más abstracta y reutilizable posible, pero siempre cumpliendo una serie de definiciones básicas y reglas de formato para que el documento pueda ser considerado como "bien formado".

XML es un formato que permite la lectura de datos a través de diferentes aplicaciones. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. En la actualidad tiene un papel muy importante ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. Esto lo ha hecho el candidato ideal para ser utilizado en el proyecto, ya que es el referente para el intercambio de información estructurada en diferentes plataformas, que es uno de los objetivos a conseguir en el proyecto Mobesens.

2.3. JAVA MESSAGE SERVICE (JMS)

Se trata de la solución de Sun Microsystems para los sistemas de mensajes entre aplicaciones Java. Permite a dichas aplicaciones crear, enviar, recibir y leer mensajes a través de una comunicación confiable de manera síncrona o asíncrona.

El JMS [16] soporta dos tipos de comunicaciones:

- Punto a punto: en la que sólo hay dos extremos de la comunicación, uno que envía el mensaje y otro que lo recibe.

- Publicador/Suscriptor: en la que una aplicación publica su mensaje en el servicio JMS y lo reciben todas las aplicaciones que estén suscritas a dicho servicio.

Una aplicación JMS consta de los siguientes elementos:

● **Cliente JMS:** son clientes del servicio JMS todas las aplicaciones que envían o reciben mensajes.

● **Mensajes:** los mensajes que se intercambian.

● **Objetos administrados:** son el punto al que se comunican los clientes JMS para enviar o recibir mensajes. Implementan las interfaces JMS y se sitúan en el espacio de nombres de JNDI (Java Naming and Directory Interface) para que los clientes puedan solicitarlos.

En el proyecto utilizamos el Servicio de Mensajes Java para realizar la comunicación asíncrona entre la base de datos y la aplicación del lado del servidor, ya que una vez llega la información de los sensores a NetInf es necesario que ésta la mande a la aplicación sin esperar a que sea solicitada.

Como proveedor del Servicio de Mensajes Java usamos Apache ActiveMQ dado que es gratuito, de código abierto y soporta JMS 1.1.

1.10. *3. Herramientas utilizadas*

3.1. APACHE TOMCAT

Para el desarrollo de nuestra aplicación web, vamos a necesitar un servidor web que nos permita procesar cualquier aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales, y síncronas o asíncronas con el cliente.

Dado que en la construcción de la aplicación haremos uso de JSP (Java Server Pages), que es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo; vamos a utilizar Apache Tomcat (también llamado Jakarta Tomcat) como servidor web, ya que éste funciona como un contenedor de servlets e implementa las especificaciones de los servlets y de las Java Server Pages (JSP) de Sun Microsystems.

Apache Tomcat presenta algunas ventajas como son:

- Puede ser usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.
- Funciona en cualquier sistema operativo que disponga de la máquina virtual Java, dado que fue escrito en Java.
- Es gratuito y fácil de instalar.

3.2. ICEFACES

Para el desarrollo de la aplicación web, se pretende hacer uso de la tecnología Ajax, que nos permite crear aplicaciones interactivas o RIA (Rich Internet Application). Además dado que se trata de una herramienta de monitorización, surge la necesidad de que algunas partes de la aplicación web estén disponibles para el usuario en tiempo real (o casi-real).

Se propone pues la utilización del entorno de trabajo ICEfaces [9]. Se trata de un framework de código abierto para construir aplicaciones web con AJAX tipo RIA. ICEfaces permite al programador aislarse totalmente de Ajax, ya que es el propio framework el que genera el código Ajax. No hacen falta etiquetas especiales: se ponen los controles en la pantalla e ICEFaces se encarga de enviar sólo la información necesaria entre cliente y servidor.

A continuación nombramos algunas de las características diferenciadoras que nos hacen elegir ICEfaces frente a otros frameworks:

- Experiencia de usuario enriquecedora: crea una experiencia de usuario superior además de utilizar las ventajas de aplicaciones Java EE. Esto se consigue gracias a los componentes que vienen incluidos dentro de la distribución de ICEfaces.
- Está basado en código abierto: ICEfaces es un framework basado en Ajax bajo licencia de código abierto. La comunidad de desarrolladores de ICEfaces incluye cerca de 20.000 desarrolladores en 36 países.
- Basado en estándares: ICEfaces es una solución basada en Java, así que los desarrolladores pueden continuar trabajando de la misma forma que lo hacen. Hay multitud de plugins desarrollados para que ICEfaces sea integrado con multitud de IDEs Java.
- El Ajax es transparente: ICEfaces aporta a los programadores un desarrollo con mínimo esfuerzo en la sección JSF.
- Compatibilidad: ICEfaces soporta todos los servidores de aplicaciones, aporta plugins para los distintos IDEs y efectos javascript de librerías de cualquier empresa que haya desarrollado Ajax del mercado.
- Seguridad: ICEfaces es una de las soluciones Ajax más seguras del mercado. Es compatible con SSL, previene los scripts de cross-site, inyección de código malicioso. Es una solución Ajax basada en servidor, la cual no utiliza datos de usuarios, además es especialmente efectivo en la prevención de fallos en los submits de los formularios y el ataque SQL por inyección.
- Escalabilidad y clustering: El servidor asíncrono HTTP (AHS) aporta una alta escalabilidad para aplicaciones ICEfaces y pueden ser utilizadas por un gran número de usuarios concurrentes, además aporta despliegue en clúster (un requisito crítico que algunas soluciones no aportan).
- Carga de páginas incremental con edición de secciones y sin recargas de página completas.
- Se preserva el contexto del usuario durante la actualización de la página, incluyendo posición del foco y scroll.
- En aplicaciones de tiempo real, las recargas de páginas son asíncronas.

3.3. GOOGLE CHARTS

Existen muchas APIs diferentes que nos permiten realizar gráficos e integrarlos en nuestra página web. En un principio se estudiaron varias posibilidades

como JS Charts, Emprise JavaScript Charts y el API de Google Charts. Finalmente las dos primeras fueron descartadas, la una porque sólo ofrecía un conjunto de gráficos posibles muy reducido (de líneas, de barras y circulares) y la otra porque era de pago y en todo el proyecto se apuesta por tecnologías y APIs open source que den más libertad al programador; por tanto, la elección final fue el API de Google Charts, dadas sus siguientes ventajas:

- Consta de una rica galería de visualizaciones, que pueden ser de dos tipos: gráficos de imagen, usando una simple petición URL a un servidor Google de gráficos; o bien gráficos interactivos, usando una librería de JavaScript desarrollada por Google. Para nuestra aplicación se optó por utilizar gráficos interactivos que proporcionan más posibilidades y opciones al usuario, así como aumentan la usabilidad y diseño de la aplicación.

- Se trata de una API gratuita y de código abierto, además de ser sencilla de usar.

- Es capaz de leer e interpretar datos que proceden de una gran cantidad de fuentes de datos distintas o formatos diferentes.

Algunas de las funcionalidades que incluye este API de Google Charts son:

- Especificar algunas opciones de nuestra visualización, como pueden ser el tamaño, el color, el formato de los títulos o anotaciones, etc.

- Manejar eventos de las visualizaciones, ya sean debidos a la propia carga del gráfico o a acciones de los usuarios (por ejemplo, que un usuario haga click sobre el gráfico).

- Mejorar el tiempo de carga de las librerías necesarias para cada visualización, por medio de dos métodos: la carga dinámica y la autocarga.

- Visualizar los errores que pueda haber en las peticiones de datos (por ejemplo si se produce un error en la consulta a la base de datos) en la aplicación.

- Implementar nuestra propia fuente de datos con la que nutrir nuestras visualizaciones. Este proceso implica la utilización de un protocolo propio del API de Google Charts que se llama “wire protocol”, se trata de un protocolo de petición-respuesta en el que dichas peticiones y respuestas tienen que tener un formato concreto.

3.4. GOOGLE MAPS

El Google Maps API, es una interfaz de programación de aplicaciones que permite insertar Google Maps en tus propias páginas o aplicaciones web con JavaScript. Proporciona diversas utilidades para manipular mapas y añadir contenido personalizado al mapa mediante diversos servicios, permitiendo crear sólidas aplicaciones de mapas.

Ésta herramienta ha sido la elegida para nuestra aplicación, ya que nos permite realizar la visualización de los sensores y su movimiento sobre un mapa de Google Maps, de una manera sencilla. Las razones de la elección de esta herramienta y no otra son las siguientes:

- Ha sido desarrollada por completo en JavaScript.

- Es una herramienta de código abierto (open source), es decir, podemos acceder en cualquier momento a todas sus funciones, librerías, etc. programadas en JavaScript, para comprender su funcionamiento interno y poder así utilizar sin problemas dichas funciones y librerías.

- Se trata de un servicio gratuito, disponible para cualquier sitio web que sea gratuito para el consumidor, como es nuestro caso. Sólo se requiere solicitar una clave para tu dominio (ya que cada clave es única para cada dominio) e incluirla en una línea de código en la página o aplicación web.

- Google Maps es un servicio universalmente conocido, cuya utilización se ha extendido mucho desde su creación, con lo cual es de suponer que a cualquier usuario le resulte sencillo e intuitivo el manejo de nuestra aplicación.

- Es una API rica en posibilidades y funcionalidades que nos van a permitir crear mapas personalizados para nuestra aplicación web.

Para la creación de nuestra aplicación web, se ha utilizado la versión 2 del API, a pesar de que ya existe una nueva versión, API v.3, que ha sido diseñada para que se cargue rápido, especialmente en los navegadores para móviles (como los dispositivos basados en Android o iPhone). Esto es debido a que al ser la 3 una versión más reciente tiene un conjunto más reducido de funciones que la anterior.

Algunas de las funcionalidades del API de Google Maps son:

- Visualizar diferentes tipos de mapa, como vista normal o satélite (imágenes satelitales de Google Earth) o una combinación de ambas.

- Hacer zoom in o zoom out, dependiendo de la resolución que queramos de la vista del mapa.

- Mostrar “ventanas de información” (aunque sólo se puede mostrar una por mapa), que muestran contenido HTML en una ventana flotante sobre el mapa.

- Manejar y administrar todo un sistema de eventos, en el que el API de Google Maps nos permite definir eventos personalizados para los objetos del API y mecanismos de detección y respuesta a estos eventos, siempre interaccionando con el DOM.

- Añadir a nuestro mapa una serie de elementos de interfaz de usuario que permiten que el usuario interactúe con el mapa. Estos elementos se denominan controles. Podemos incluir variaciones de ellos en nuestra aplicación o incluso crear nuestros propios controles personalizados. Algunos de estos controles son los que nos permiten cambiar de tipo de mapa, o el acercamiento y desplazamiento en el mapa.

- Crear, añadir o eliminar a nuestro mapa superposiciones, éstas son objetos del mapa vinculados a coordenadas de latitud y longitud, por lo que se mueven al arrastrar o aplicar el acercamiento sobre el mapa. Representan objetos que se añaden al mapa para designar puntos (marcadores), líneas (polilíneas) o zonas (polígonos). El uso más importante que daremos en nuestra aplicación a estas superposiciones será crear marcadores para cada uno de los sensores presentes en el mapa.

- El API de Google Maps nos proporciona también una serie de servicios como: análisis de datos y código XML, codificación geográfica, uso de objetos panorámicos de Street View, integración con Google Earth, búsqueda local, superposiciones KML y GeoRSS, superposiciones de tráfico e indicaciones.

3.5. EXIST-DB

Como sistema de almacenamiento para los datos del proyecto Mobesens se utiliza una base de datos nativa XML, que al igual que otro tipo de bases de datos, soporta transacciones, acceso multi-usuario, lenguaje de consulta, etc., pero se caracteriza por estar especialmente diseñada para almacenar documentos XML.

Para el manejo y administración de esta base de datos se utiliza eXist-DB. 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 estas prácticas.

Se va a utilizar la última versión publicada de eXist-DB que es la 1.4.0. Esta versión mejora el procesamiento XQuery gracias a un manejo más robusto de los

documentos en memoria. Ofrece una más rápida y costumizable indexación de texto, mediante la integración de Lucene en el mecanismo XQuery. También proporciona reescritura de URLs ligera y un entorno de trabajo MVC; además de soporte para XProc, XForms y otros módulos XQuery.

3.6. TAHOE-LAFS

En nuestra base de datos, necesitamos un sistema de almacenamiento de los datos. Vamos a usar Tahoe-LAFS, que es un sistema de almacenamiento de datos en nube, gratuito y abierto. Distribuye los datos en múltiples servidores de tal manera que, incluso si alguno de los servidores falla o es invadido por un atacante, el sistema completo de ficheros continúa funcionando correctamente, incluyendo la preservación, privacidad y seguridad de los datos.

Proporciona seguridad independiente del proveedor. Esto quiere decir, que el proveedor del servicio nunca tiene la posibilidad de leer o modificar los datos, de tal manera que no tenemos que preocuparnos de que se cometan errores que provoquen el que nuestros datos sean visibles para otros clientes o para el público en general o de que los propios empleados del servicio de almacenamiento violen la política de privacidad por descuido, avaricia o mera curiosidad.

ANEXO C. MANUAL DE INSTALACIÓN

Para el correcto funcionamiento de la aplicación, hay que realizar las siguientes instalaciones en el equipo:

- Apache Tomcat
- eXist-DB
- VMware
- Tahoe-LAFs
- ActiveMQ
- Entorno Eclipse
- ICEfaces

A continuación vamos a describir la instalación de dichas herramientas. En primer lugar debemos instalar Apache Tomcat ya que es el servidor web elegido para desplegar la aplicación. A continuación instalaremos eXist-DB y Tahoe-LAFS ya que son las herramientas que usa NetInf (interfaz de comunicación con la base de datos) para gestionar la base de datos. En el caso de Tahoe-LAFS la instalación en Windows da problemas. Es por eso que se requiere la instalación previa de un software de virtualización, como es VMware, que nos permita disponer del sistema operativo Linux sobre el que realizar la instalación de Tahoe-LAFS. También es necesaria la instalación de un proveedor del servicio de mensajes de Java (JMS), para esta labor se ha elegido Apache ActiveMQ por ser gratuito y de código abierto.

Por último y aunque no es necesario para el despliegue de la aplicación, vamos a explicar como se realiza la instalación de los entornos de trabajo Eclipse y ICEfaces, pues son los que se han usado en la realización del proyecto. Hay multitud de entornos de desarrollo de programación que se podrían usar, pero se ha elegido Eclipse dado que nos permite programar en Java, desarrollar aplicaciones web y el hecho de que yo ya conocía este entorno de trabajo por haberlo utilizado previamente en alguna asignatura de la carrera. A su vez una vez instalado el entorno Eclipse deberíamos instalar el marco de trabajo ICEfaces. Existe una versión completa de ICEfaces, pero también está disponible como plug-in para diferentes interfaces de desarrollo como Eclipse.

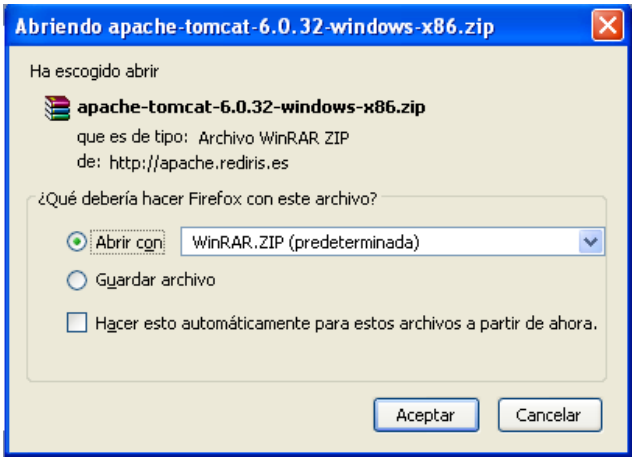
Al final de la sección de instalación de Eclipse se dan las indicaciones necesarias para desplegar y ejecutar la aplicación web de monitorización y la aplicación de comunicación con la base de datos (NetInf), a partir de los archivos .war incluidos en el cd del PFC.

1. *APACHE TOMCAT*

Es el servidor web elegido para desplegar la aplicación de monitorización.

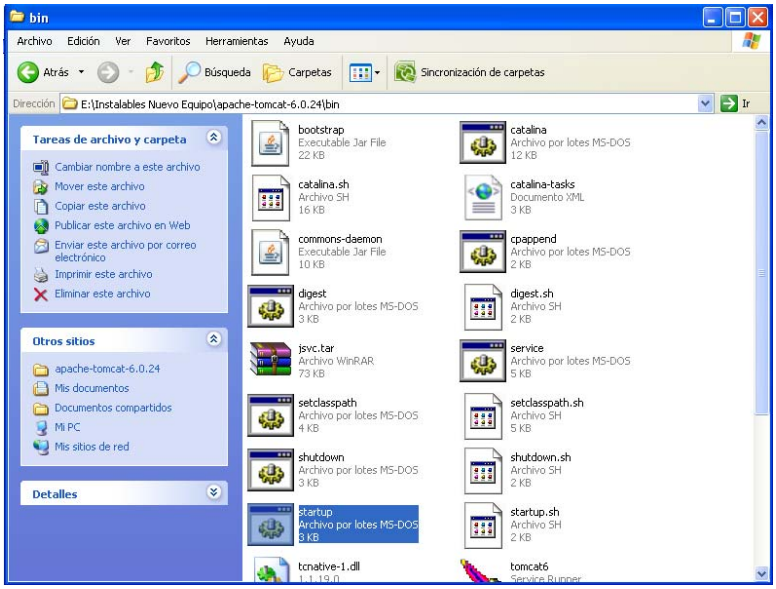
INSTALACIÓN

1º. Descargar el Apache Tomcat de la página de apache:
http://tomcat.apache.org/download-60.cgi
Seleccionamos la versión 6.0 con la opción Windows 32-bits:



Obteniendo un fichero .zip : apache-tomcat-6.0.32-windows-x86.zip

2º Descomprimir este archivo en la ubicación deseada, en nuestro caso:
E:\Instalables Nuevo Equipo
Creándose entonces una carpeta llamada apache-tomcat-6.0.32, en dicha ubicación. Para arrancar tomcat sólo tenemos que hacer doble click en el archivo startup que se encuentra en: apache-tomcat-6.0.32/bin/



2. EXIST-DB

Previamente a la instalación de eXist, tenemos que asegurarnos de que el JDK 1.5 o una versión mayor haya sido instalada en el sistema operativo; porque durante la instalación de eXist, se nos pide especificar el path al directorio donde se encuentra el JDK. Nosotros hemos descargado el jdk-6u18-windows-i586 de la página de oracle:

http://www.oracle.com

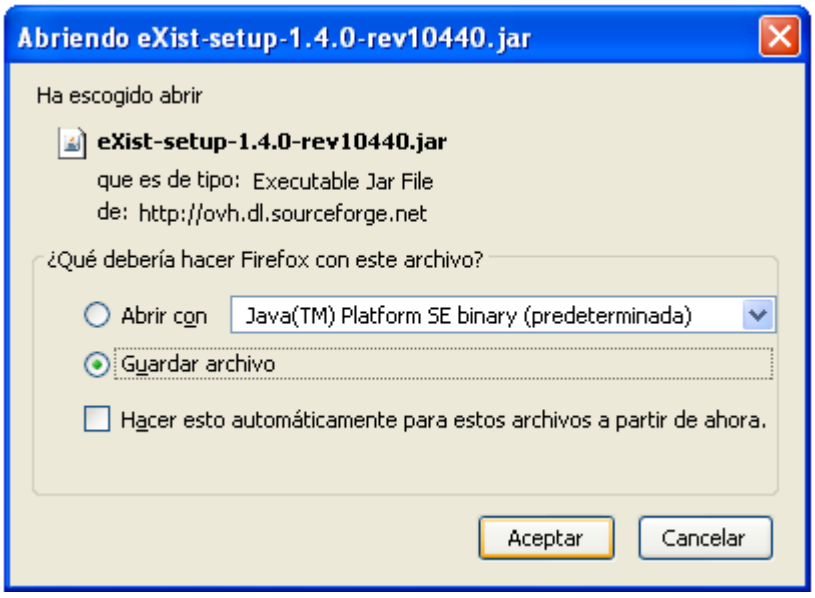
Una vez descargado el fichero sólo hay que hacer doble click sobre él y se lanza el instalador. Luego simplemente seguir los pasos especificados para completar el proceso de instalación.

INSTALACIÓN

1º. Descargar eXist-DB de la página de eXist:

http://exist.sourceforge.net/download.html

Seleccionamos la opción .jar, que incluye una herramienta de instalación gráfica que nos guía a través del proceso de instalación y que determina automáticamente los ajustes del sistema correctos para eXist (como por ejemplo los caminos y variables de entorno).

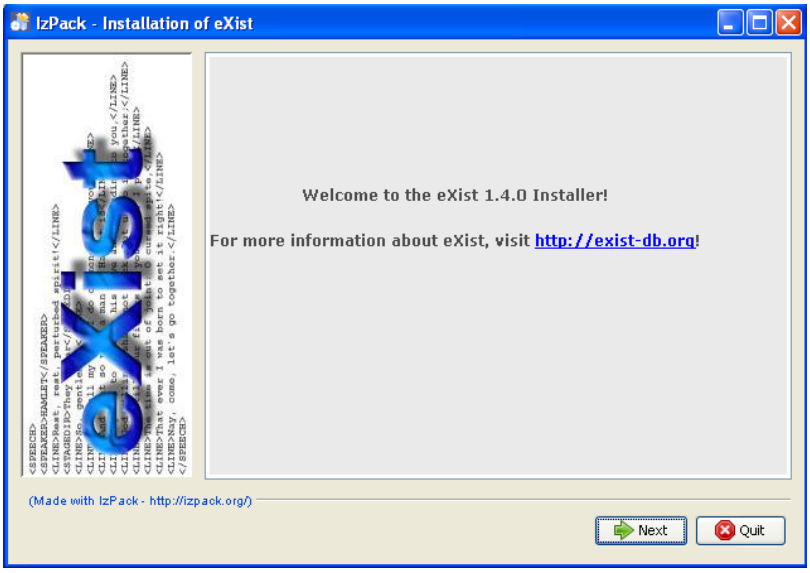


Obteniendo un fichero: eXist-setup-1.4.0-rev10440.jar

2º. Copiamos este fichero en la ubicación deseada, en nuestro caso:

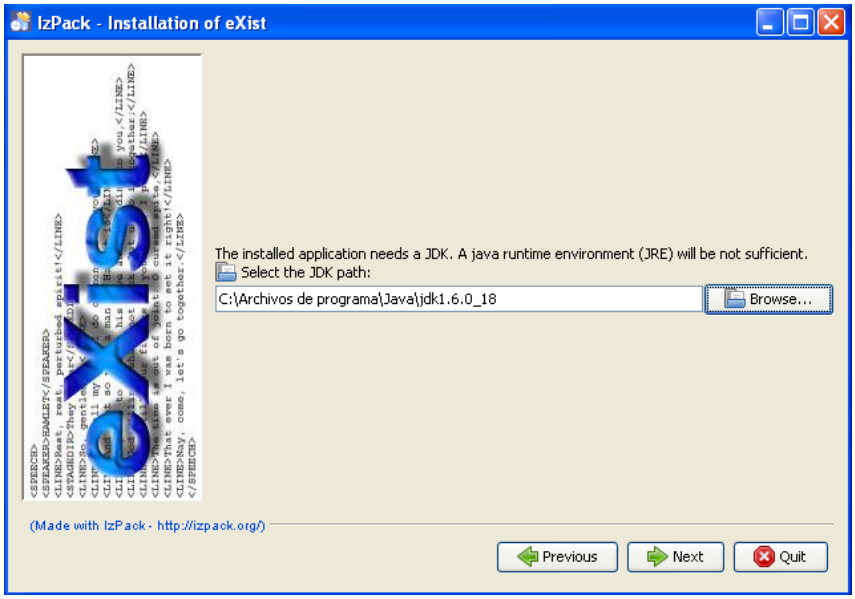
E:\Instalables Nuevo Equipo\exist

3º. Hacemos doble click sobre el fichero anterior. Esto lanzará el instalador y simplemente hay que seguir los pasos para completar el proceso de instalación.
3.1. Pantalla de inicio del instalador.

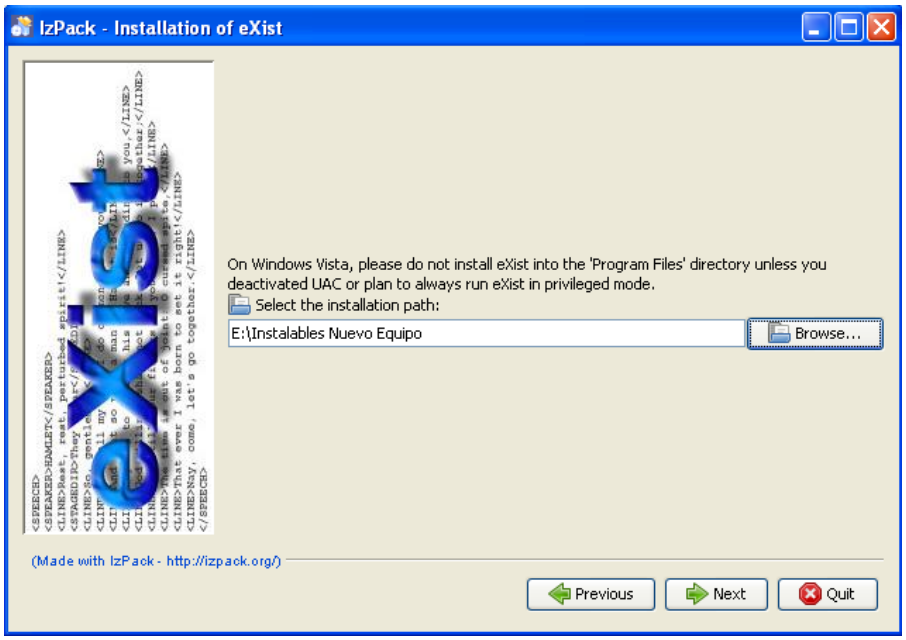


3.2. El instalador nos pide que especifiquemos el path al jdk que hemos instalado previamente y que en nuestro caso se encuentra en:

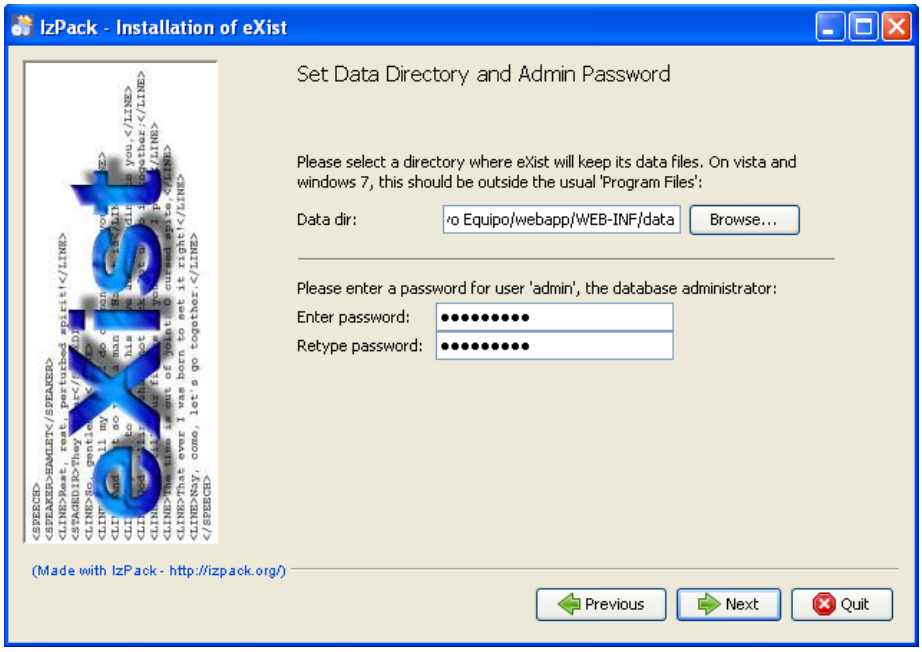
C:\Archivos de programa\Java\jdk1.6.0_18



3.3. Elegimos el directorio en el que queremos instalar eXist.



3.4. Se nos pide seleccionar el directorio en el que eXist va a guardar sus ficheros de datos (dejamos la que nos ofrecen por defecto), así como una contraseña para el administrador de la base de datos (nosotros).



Una vez completada la instalación, tenemos que lanzar eXist, tenemos varias formas de hacer esto:

- Si el instalador ha creado accesos directos, simplemente hacemos doble click en el icono *eXist Database Startup* o seleccionamos dicho objeto desde el menú de inicio del escritorio.
- Para lanzar eXist manualmente, tenemos que:
  1. Abrir la ventana de comandos DOS de Windows.

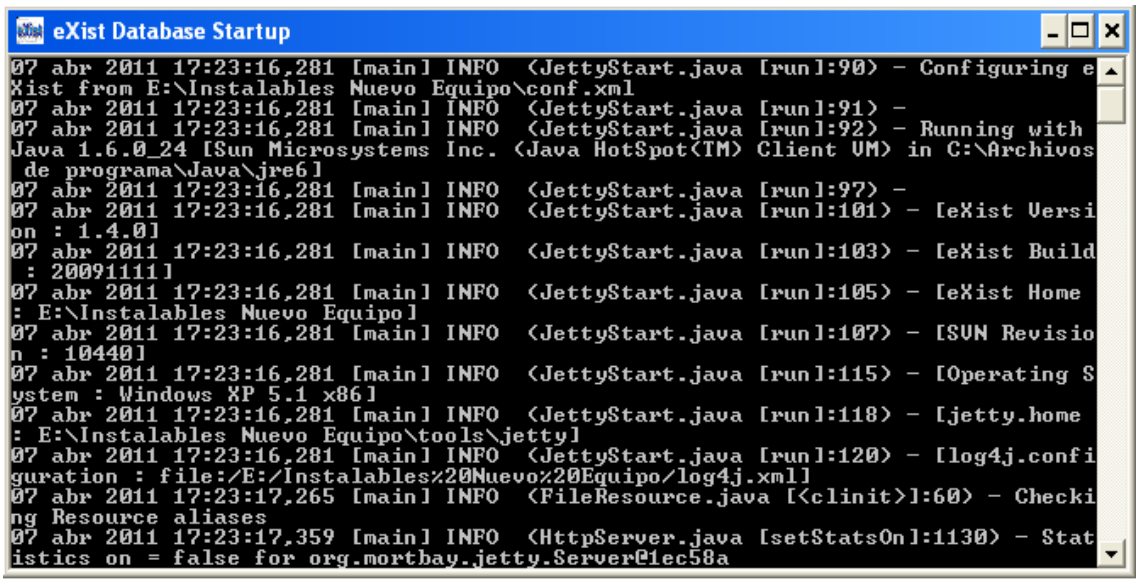
2.Posicionarnos en el directorio en el que se haya instalado eXist.

3.Escribir el siguiente comando: *bin/startup.sh*

- En caso de que las opciones anteriores no hayan lanzado eXist, podemos cargarlo directamente con la rutina de arranque:

java -Xmx128M -Djava.endorsed.dirs=lib/endorsed -jar start.jar jetty

Si eXist ha arrancado adecuadamente, en la consola deberíamos ver lo siguiente:



En la configuración por defecto, eXist corre dentro de un servidor de aplicaciones servido por un servidor web Jetty preconfigurado. Todas las interfaces están provistas como servlets. Para comprobar que el servidor está corriendo, podemos introducir la siguiente URL en nuestro navegador web:

http://localhost:8080/exist/index.xml

deberíamos de ver las siguiente página web:

Es importante notar que la página anterior no es la página de inico de eXist, sino una copia local incluida con la instalación de eXist. Estos ficheros locales incluyen documentación de eXist, librerías de funciones, recursos y ejemplos de XQuery.

A partir de aquí ya podemos utilizar eXist en nuestras aplicación web y registrarnos en la página de eXist con nuestra contraseña (la que hemos introducido al hacer la instalación) para acceder a nuestras colecciones de datos y hacer pruebas de consultas XQuery sobre ellas usando el XQuery Sandbox, etc.

¡IMPORTANTE! eXist utiliza por defecto el puerto 8080 para el servidor Jetty, esto nos causa un problema al intentar poner en marcha la aplicación, ya que nuestro servidor web Apache Tomcat también utiliza el puerto 8080. La solución será cambiar el puerto del servidor Jetty. Para ello editaremos el fichero de configuración:

%HOME_EXIST%/tools/jetty/etc/jetty.xml

cambiando por un nuevo valor (por ejemplo 8888)

```
<SystemProperty name="jetty.port" default="8080"/>
```

por

```
<SystemProperty name="jetty.port" default="8888"/>
```

También podría cambiarse el puerto cada vez que se quiere arrancar eXist, escribiendo lo siguiente en la ventana de commandos de Windows:

Java –Djetty.port=8888 –jar start.jar jetty

Sin embargo, es mejor modificar directamente el fichero de configuración como hemos explicado antes, ya que así el cambio es “permanente” y no tenemos

que estar indicando que puerto queremos usar para Jetty cada vez que lanzamos eXist.

3. VMWARE

A lo largo del proyecto ha surgido la necesidad de trabajar en algunas ocasiones con el sistema operativo Linux (para la instalación de Tahoe, dado que la versión para Windows no funcionaba bien y también para introducir algunos ficheros XML con los datos utilizados en los gráficos históricos y de perfil, en la base de datos). Para evitarnos realizar particiones del disco duro de nuestro equipo, se decidió utilizar un software de virtualización como es VMware.

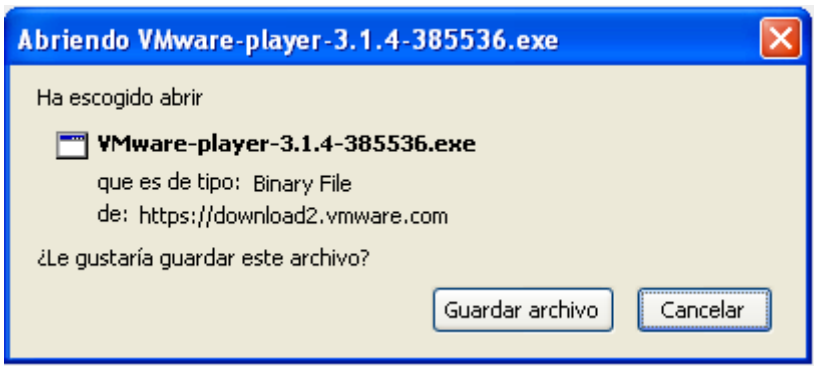
Se trata de una aplicación que nos permite instalar sistemas operativos adicionales, conocidos como "sistemas invitados", dentro de otro sistema operativo "anfitrión", cada uno con su propio ambiente virtual. Entre los sistemas operativos soportados (en modo "anfitrión") se encuentran GNU/Linux, Mac OS X, Microsoft Windows, y Solaris/OpenSolaris y dentro de ellos es posible virtualizar los sistemas operativos FreeBSD, GNU/Linux, OpenBSD, OS/2 Warp, Windows, Solaris, MS-DOS y muchos otros.

En nuestro caso el sistema operativo "anfitrión" será Windows XP y el "invitado" será Linux.

INSTALACIÓN

1º. Podemos descargar VMware de la página:
http://downloads.vmware.com/d/info/desktop_downloads/vmware_player/3_0

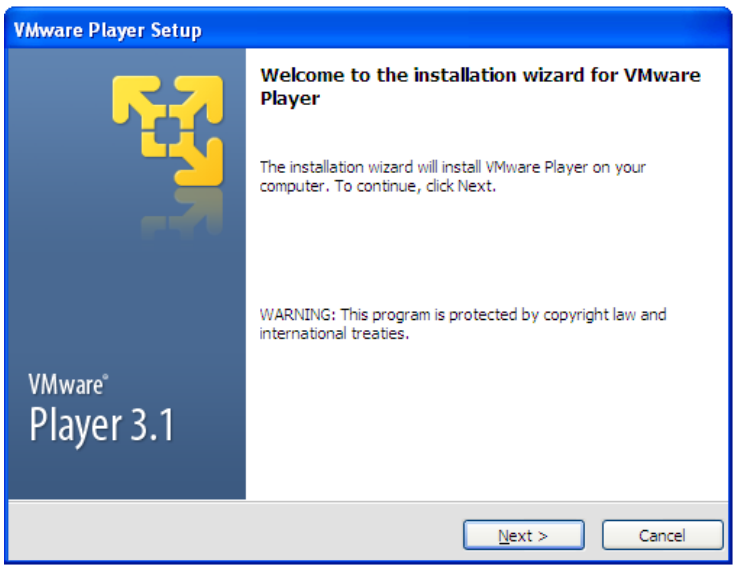
Hay que registrarse (simplemente introducir nombre y apellidos y dirección de correo electrónico), para poder descargar gratuitamente. Una vez hecho esto, elegimos la opción de descarga del VMware player 3.1.4 para Windows de 32 y 64 bits:



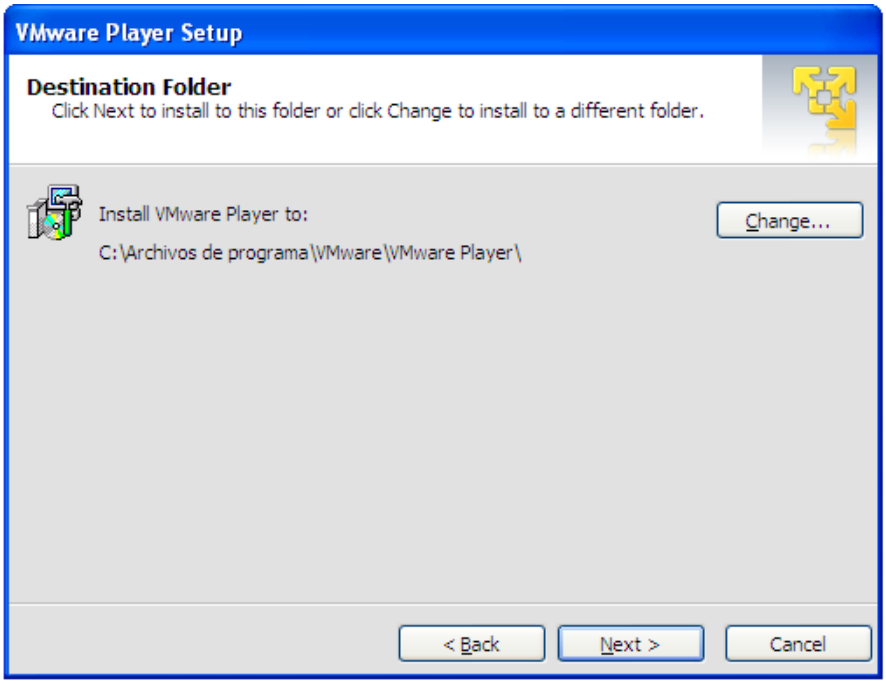
y guardamos el archivo en la ubicación deseada, en nuestro caso:

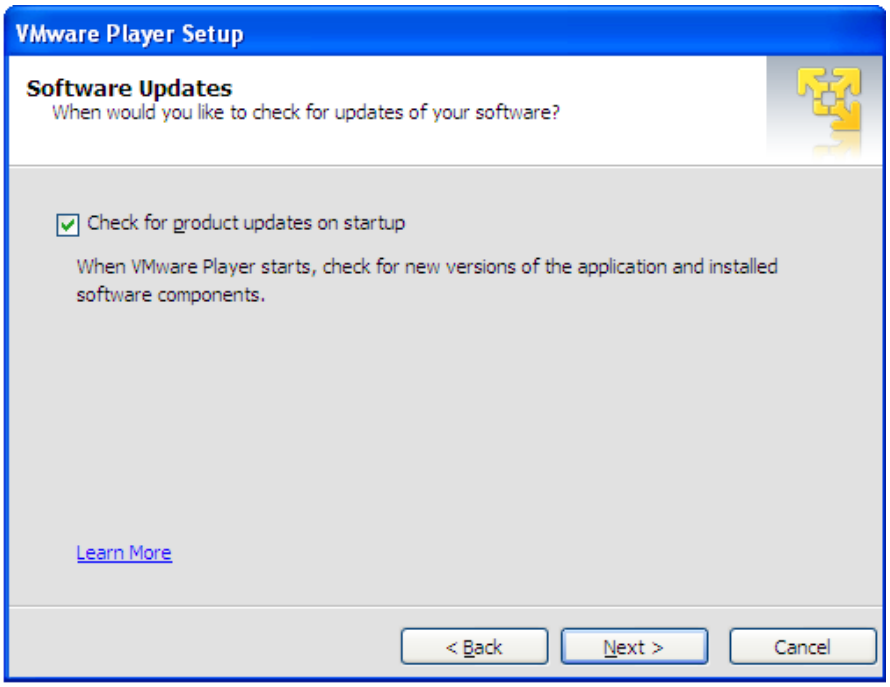
E:\Instalables Nuevo Equipo\VMware

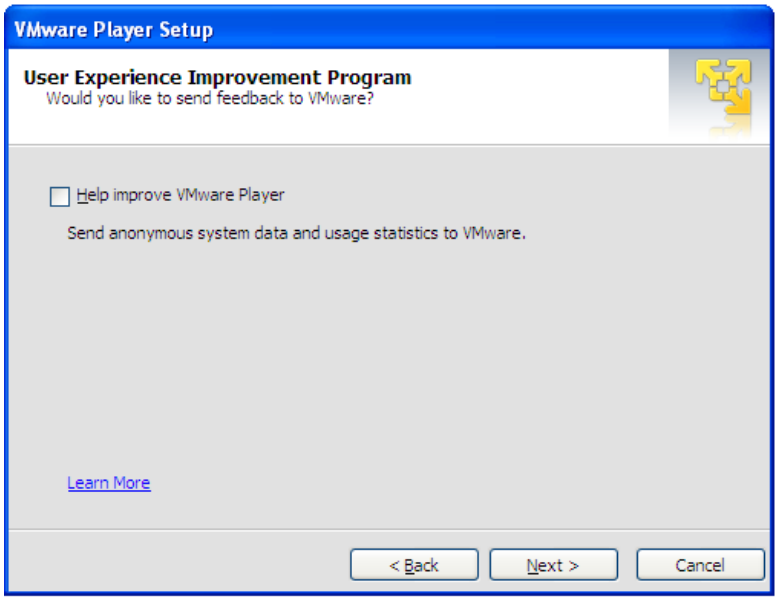
2º. Este archivo es un ejecutable, haciendo doble click sobre él, lanzamos el instalador:

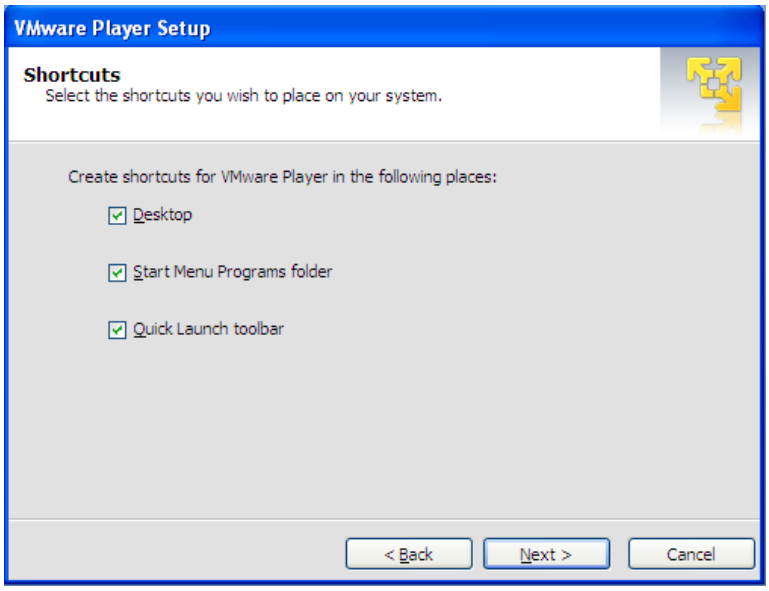


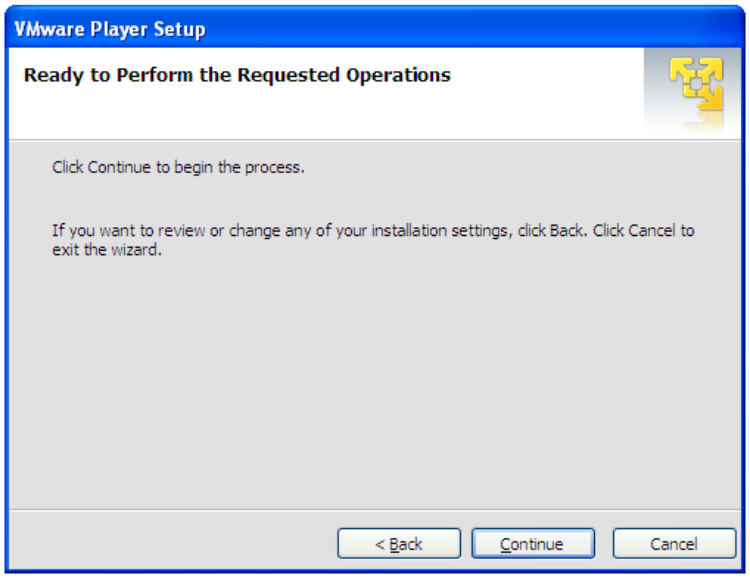
Y seguimos los pasos del proceso de instalación.

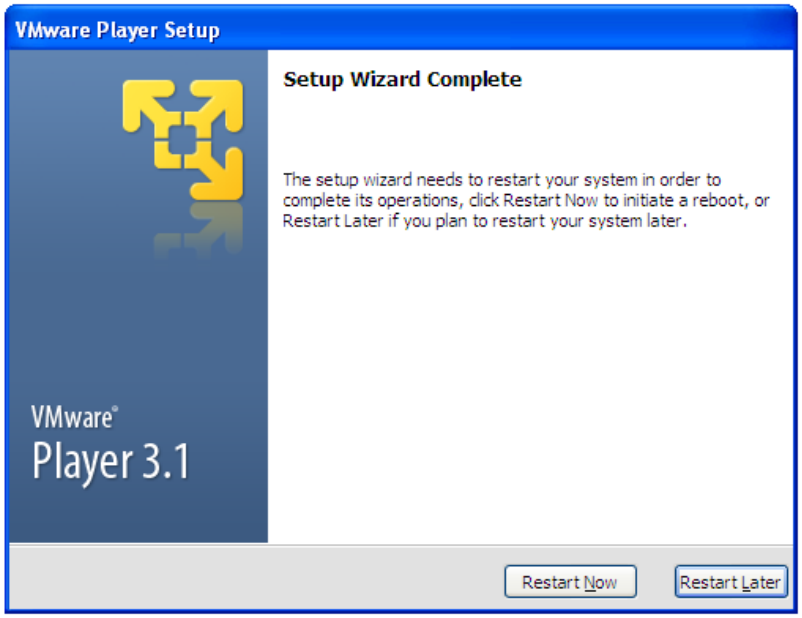



VMware Player Setup
Software Updates
When would you like to check for updates of your software?
Check for product updates on startup
When VMware Player starts, check for new versions of the application and installed software components.
Learn More
< Back
Next >
Cancel


VMware Player Setup
User Experience Improvement Program
Would you like to send feedback to VMware?
Help improve VMware Player
Send anonymous system data and usage statistics to VMware.
Learn More
< Back
Next >
Cancel

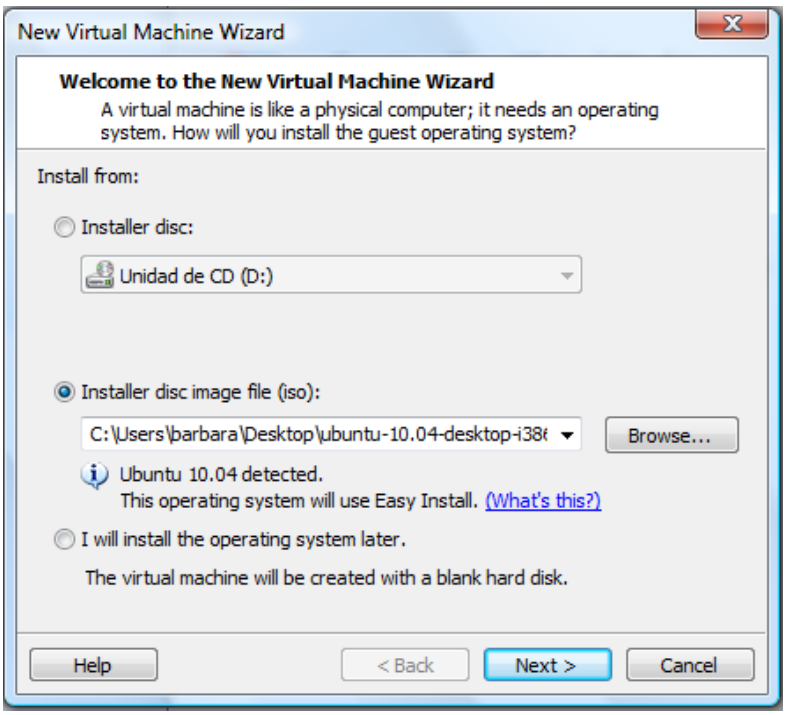

VMware Player Setup
Shortcuts
Select the shortcuts you wish to place on your system.
Create shortcuts for VMware Player in the following places:
Desktop
Start Menu Programs folder
Quick Launch toolbar
< Back
Next >
Cancel


VMware Player Setup
Ready to Perform the Requested Operations
Click Continue to begin the process.
If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
< Back
Continue
Cancel

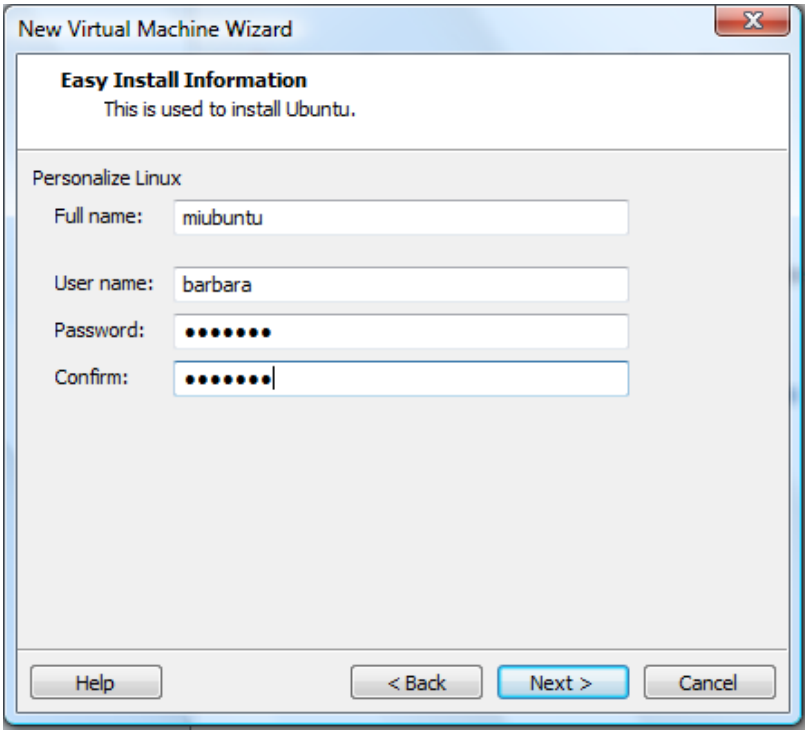


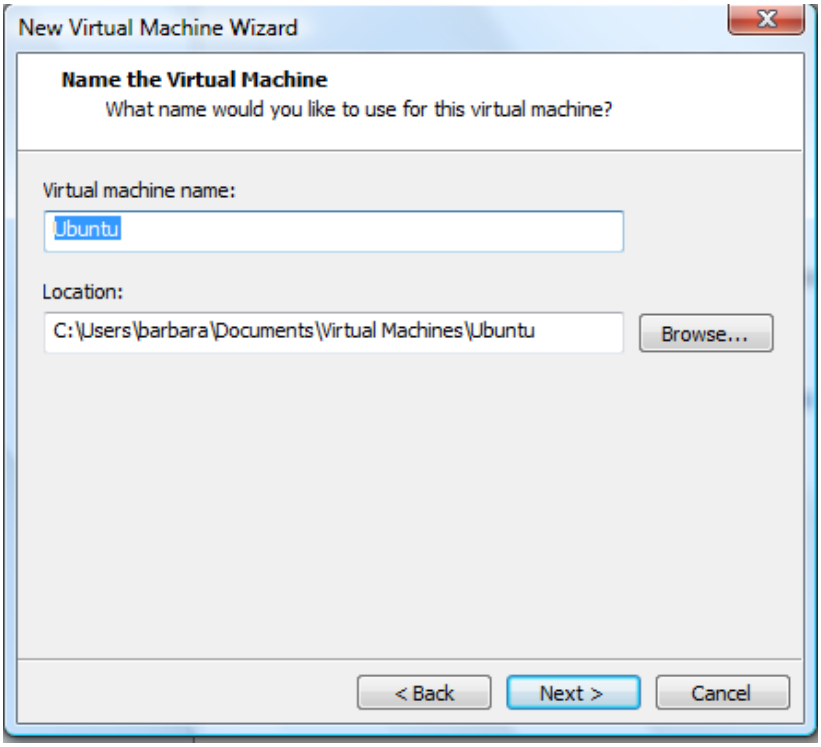
3º. Una vez instalado VMware, lo abrimos (desde el shortcut que se nos ha creado en el escritorio) y tenemos que aceptar el acuerdo de la licencia y ya nos aparece la ventana de inicio del VMware.

Tendremos que crear una nueva máquina virtual, elegimos pues la opción "Create a New Virtual Machine" y seguimos las instrucciones para crear una nueva máquina virtual. El primer paso es seleccionar la imagen de disco duro que será usada como disco de arranque. Tendremos que descargar una imagen de Ubuntu y guardarla en nuestro equipo. Para en este paso de la creación de una nueva máquina virtual, seleccionar la opción "Installer disc image file" y poder elegir esta imagen de Ubuntu previamente descargada:

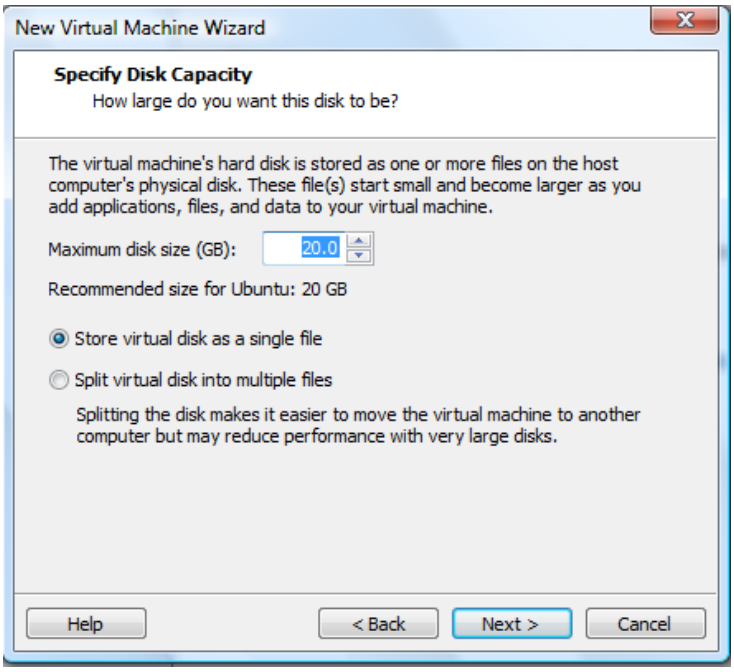


Luego se nos pide que le demos un nombre a la nueva máquina virtual, así como que definamos el nombre de usuario y la contraseña para iniciar la sesión en Ubuntu:

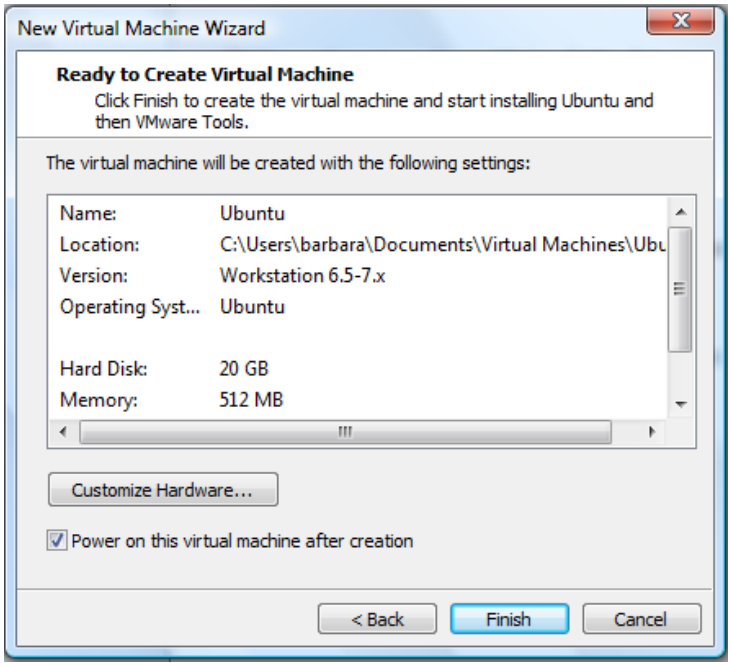




Seleccionamos el espacio en disco que vamos a dedicar a la máquina virtual y la opción que dice “Store virtual disk as a single file” para que el disco se guarde como un solo archivo y no como varios.



Finalizamos la instalación:



4. TAHOE-LAFS

Se llevará a cabo la instalación de Tahoe-LAFS en Linux, dado que, a pesar de que Tahoe-LAFS está soportado en Windows, no hemos sido capaces de instalarlo correctamente en ese sistema operativo. Sin embargo, en Linux, la instalación es más sencilla, ya que Tahoe-LAFS está más enfocado a trabajar en este sistema

operativo, o al menos la documentación de instalación es más extensa y nos permite que el proceso de instalación se realice de manera más simple.

Por tanto, tened en cuenta que todo el proceso de instalación descrito a continuación ha de ser realizado desde el Linux (Ubuntu) que hemos instalado previamente y al que tenemos acceso a través de VirtualBox (también instalado con anterioridad).

INSTALACIÓN EN LINUX

La instalación de Tahoe-LAFS requiere tener Python instalado en el equipo (esto es debido a que Tahoe-LAFS ha sido programado en Python), para comprobar que versión de Python tenemos instalada (si es que tenemos alguna) podemos escribir lo siguiente en la ventana de comandos:

python —version

La mínima versión de Python requerida para que Tahoe-LAFS funcione es la 2.4.

1º. En caso de no tener ninguna versión de Python instalada en el equipo, podemos descargar una desde la página de Python:

http://www.python.org/download/releases/2.6.6/

elegimos la opción Bzipped source tar ball (2.6.6) (sig) para Linux y la descargamos:

2º. Descomprimimos el archivo, escribiendo en la ventana de comandos:

gunzip -c Python-2.6.6.tar.bz2 | tar xf –

así el archivo queda descomprimido en el directorio Python-2.6.6.

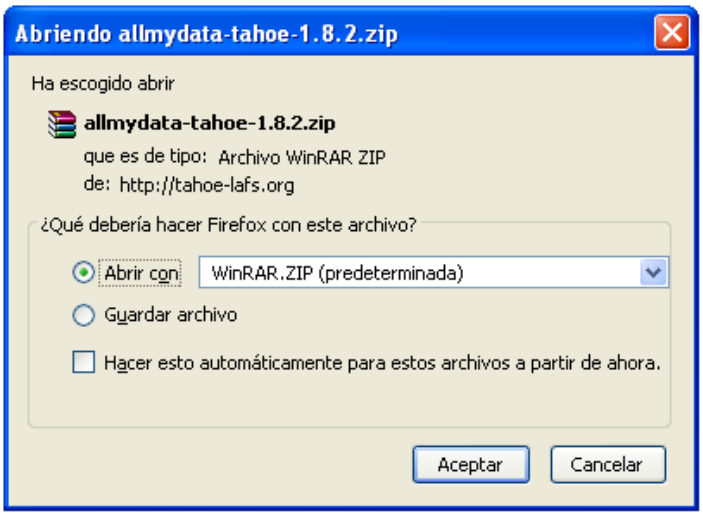
Para ejecutar el comando setup (el que realiza la instalación de Python) debemos situarnos siempre en el directorio raíz, en este caso, el directorio donde hemos descomprimido el archivo Python-2.6.6.tar.bz2. Nos situamos pues en dicho directorio, escribiendo en la ventana de comandos:

cd Python-2.6.6

y ejecutamos el siguiente comando:

python setup.py install

Una vez hecho esto, ya tenemos instalado Python.
Ahora, para instalar Tahoe-LAFS:
1º. Descargar Tahoe-LAFS de la página:
http://tahoe-lafs.org/source/tahoe-lafs/releases/allmydata-tahoe-1.8.2.zip



obteniendo un archivo allmydata-tahoe-1.8.2.zip

2º. Descomprimir el archivo en la ubicación deseada, por ejemplo el escritorio de nuestra máquina virtual Ubuntu.
3º. Para instalar Tahoe, escribimos el siguiente comando en la línea de comandos:

sudo apt-get install tahoe-lafs

Una vez instalado Tahoe, debemos crear los nodos *introducer* y *client* , escribiendo los siguientes comandos:

tahoe create-introducer introducer-tahoe
tahoe create-client client-tahoe

para poner en marcha el introducer, escribimos:

tahoe start introducer-tahoe

este comando crea un fichero introducer.furl en la carpeta introducer-tahoe, que tendremos que copiar en cada cliente (carpeta client-tahoe).
Para poner en marcha el cliente, escribimos:

tahoe start client-tahoe

5. APACHE ACTIVEMQ

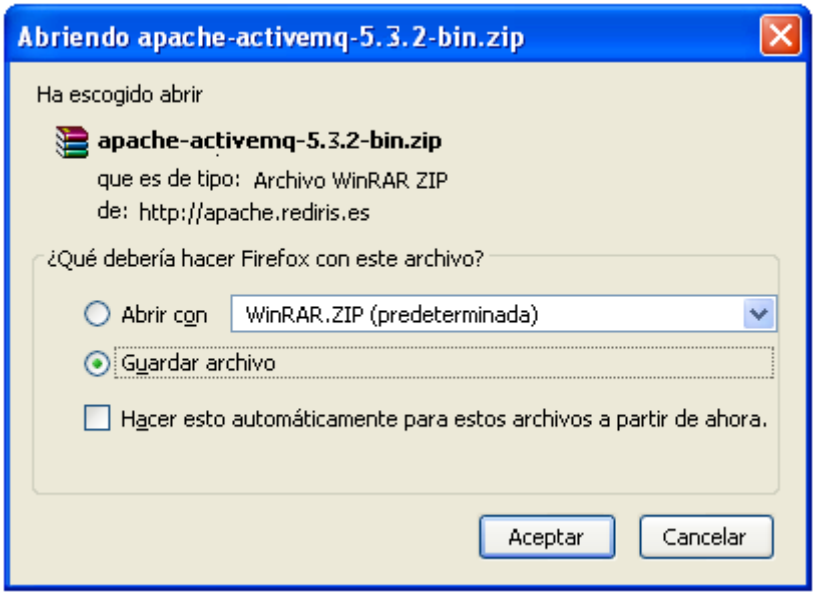
En el estado actual de la aplicación, es necesario utilizar un sistema de mensajes para hacer posible la comunicación entre la aplicación del lado del servidor (en concreto el gestor de datos) y la base de datos (NetInf). La solución de Sun

Microsystems para estos sistemas de mensajes es el servicio de mensajes de java (JMS), que permite la comunicación asíncrona entre cliente y servidor. En concreto se utiliza la modalidad Publicador/Suscriptor, en la que la aplicación A publica su mensaje en el servicio JMS y lo reciben todas las aplicaciones que estén suscritas al servicio JMS al que se envió el mensaje.

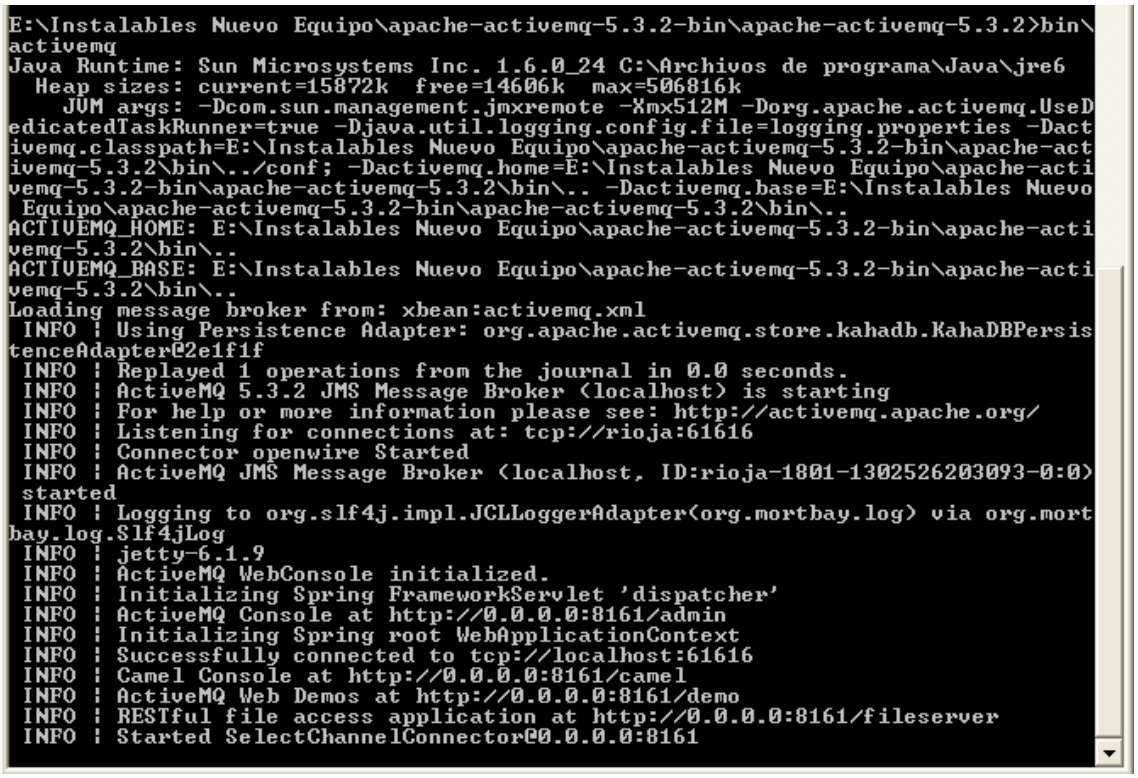
Se utiliza Apache ActiveMQ como proveedor de este servicio de mensajes, ya que es de código abierto, gratuito y soporta JMS 1.1.

INSTALACIÓN

1º. Podemos descargar una versión de ActiveMQ de la página:
http://activemq.apache.org/download.html
elegimos la opción binaria de ActiveMQ 5.3.2 para Windows:



Obteniendo un archivo apache-activemq-5.3.2-bin.zip
2º. Descomprimimos el archivo en la ubicación deseada, en nuestro caso:
E:\Instalables Nuevo Equipo
3º. Para iniciar ActiveMQ, debemos ejecutar un archivo llamdo activemq que se encuentra en:
E:\Instalables Nuevo Equipo\apache-activemq-5.3.2-bin\apache-activemq-5.3.2\bin
Para ello, abrimos la consola de comandos de Windows, nos colocamos en el directorio bin, especificado por el camino anterior y escribimos activemq.
Si ActiveMQ se ha iniciado correctamente, veremos lo siguiente en la consola:

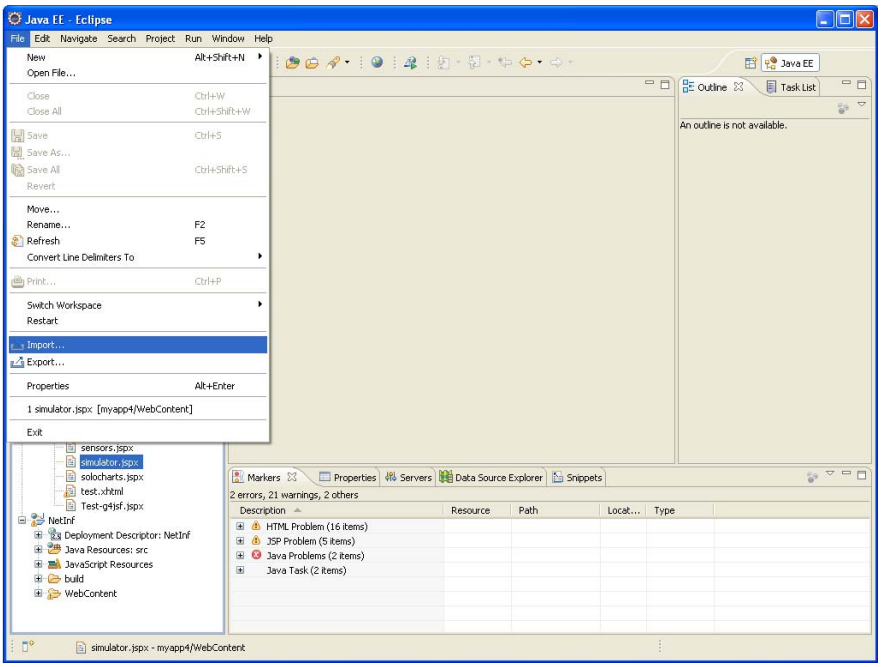


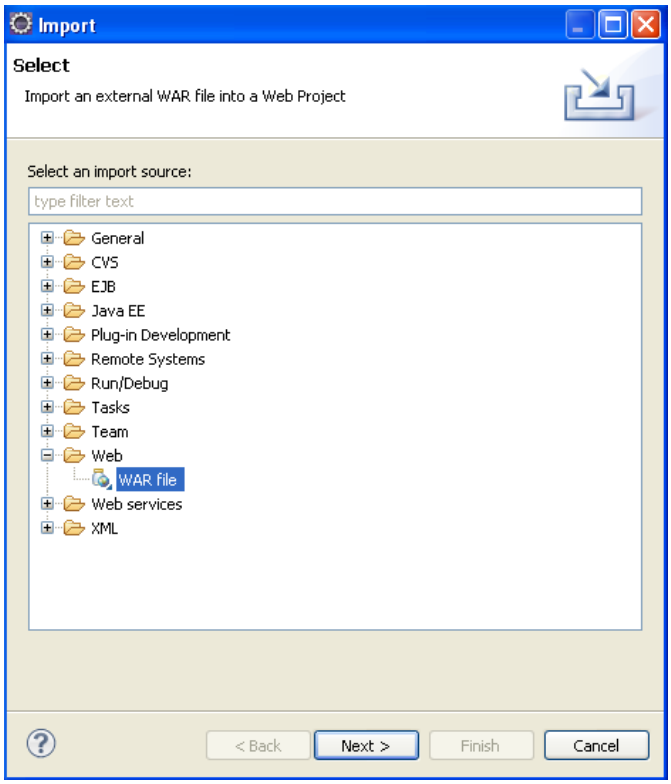
Una vez iniciado ActiveMQ ya podemos utilizar sus servicios en nuestra aplicación.

6. APLICACIÓN DE MONITORIZACIÓN

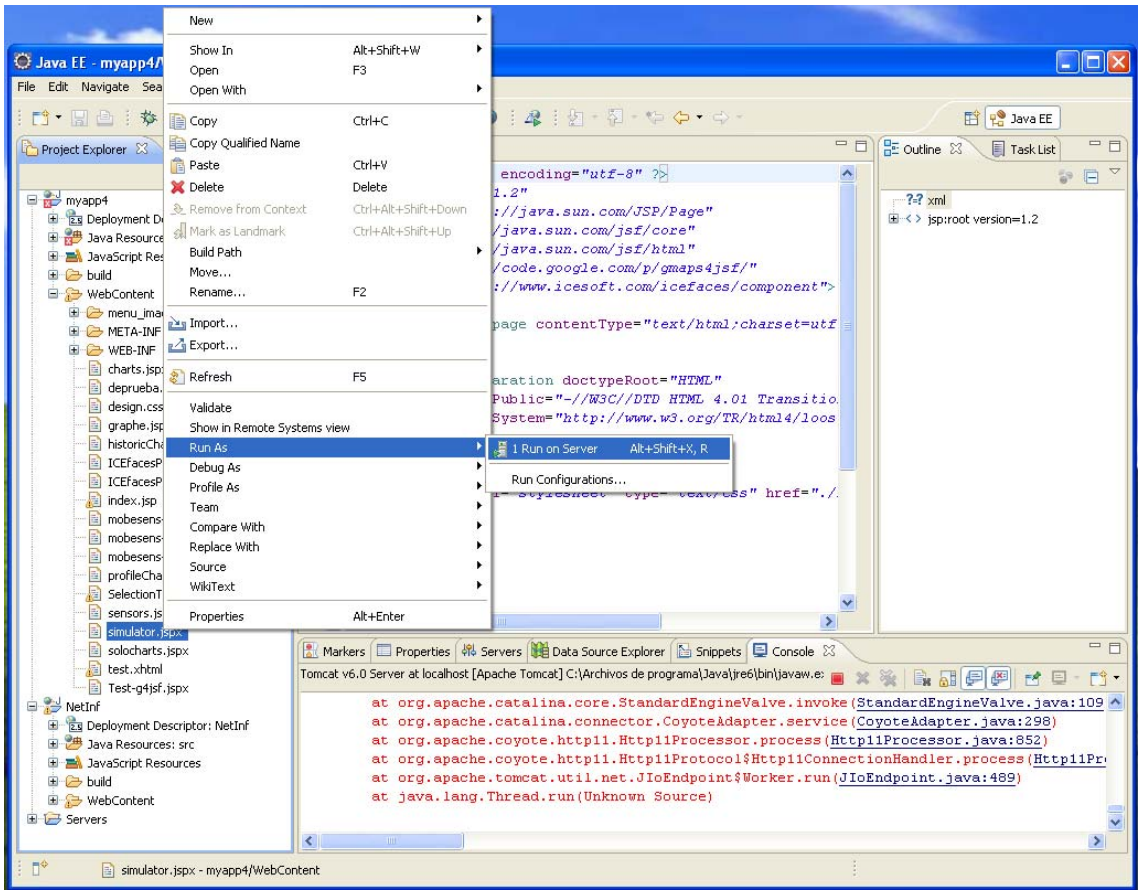
Los proyectos que se han creado a lo largo del PFC son del tipo aplicación web (se trata de un conjunto de páginas HTML, JSP, servlets, recursos y fichero archivo fuente que pueden ser administrados como una sola entidad). A la hora de importar o exportar este tipo de proyectos, lo que se hace es empaquetarlos en un tipo de archivo llamado Web Archive (WAR). Por tanto, estos son los ficheros de los que dispondremos para instalar la aplicación, en concreto los ficheros: myapp.war y netInf.war q corresponden respectivamente a la aplicación de monitorización y la aplicación de comunicación con la base de datos. Estos ficheros están incluidos en el CD adjunto a la memoria del PFC.

Dado que se ha trabajado con la herramienta Eclipse, a continuación procedemos a explicar como se desplegaría la aplicación en este marco de trabajo. Para importar proyectos ya creados anteriormente, seleccionaremos la opción Import del menu File, se abrirá una ventana en la que tenemos que elegir el tipo de archivo a importar, en nuestro caso, nuestros proyectos son .war, así que seleccionamos la opción Web→WAR file y elegimos la ubicación en la que está situado el proyecto a importar.





Una vez importados los .war, estos nos aparecerán en el espacio de proyectos y podremos acceder tanto a los ficheros que contienen las clases Java como a los .jspx que contienen la aplicación de monitorización del cliente. Para ejecutarlos sólo tenemos que hacer click derecho en el fichero a ejecutar y seleccionar la opción Run As → Run on Server (asegurándonos de seleccionar Apache Tomcat como el servidor sobre el que queremos correr nuestra aplicación) como vemos en la siguiente Figura.



A la hora de apagar tomcat sólo tenemos que hacer doble click en el archivo shutdown que se encuentra en: apache-tomcat-6.0.32/bin/ o bien apagarlo directamente desde eclipse, mediante el botón Terminate (botón cuadrado y rojo) que está situado en la pestaña Console.

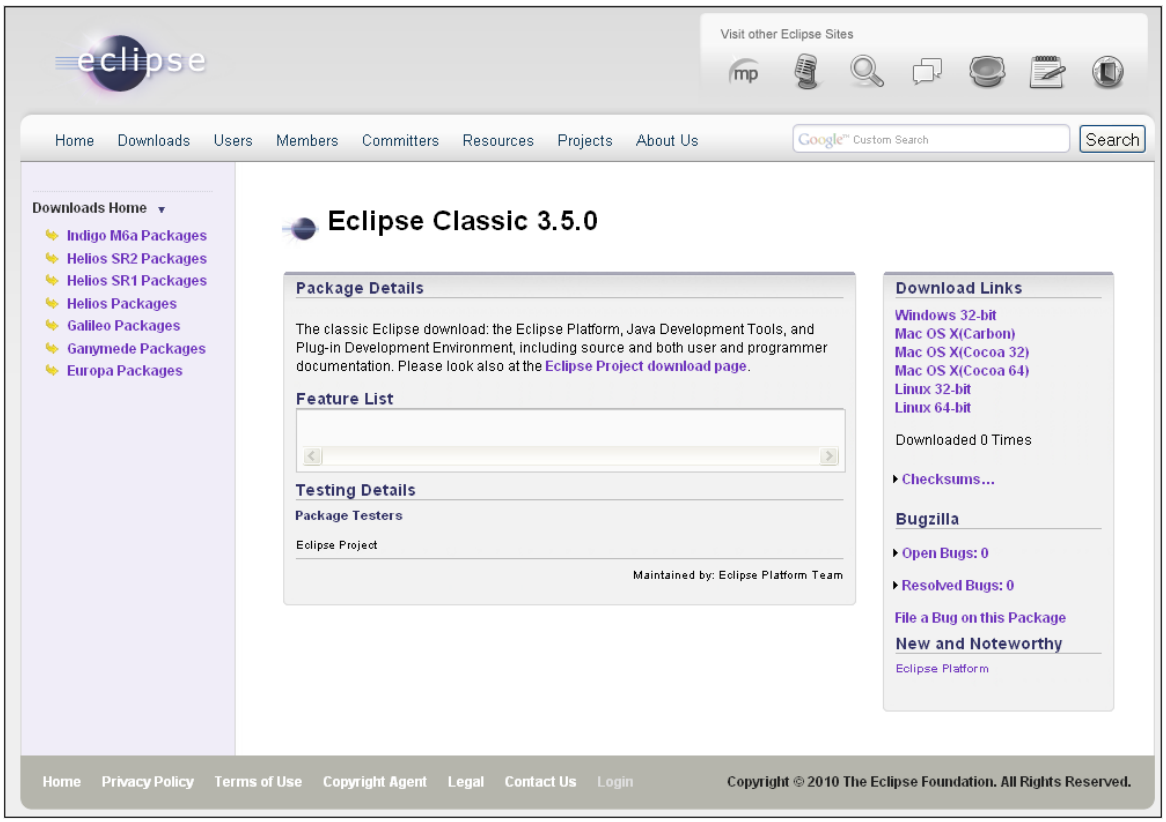
7. ENTORNO ECLIPSE

Eclipse es un entorno de desarrollo de programación, requiere la instalación previa de un JRE (Java Runtime Environment) en el equipo, para poder ejecutarse. Se recomienda la versión Java SE 5 o mayor. En nuestro caso, el JRE ya viene incluido en la versión de Galileo que nos descargamos.

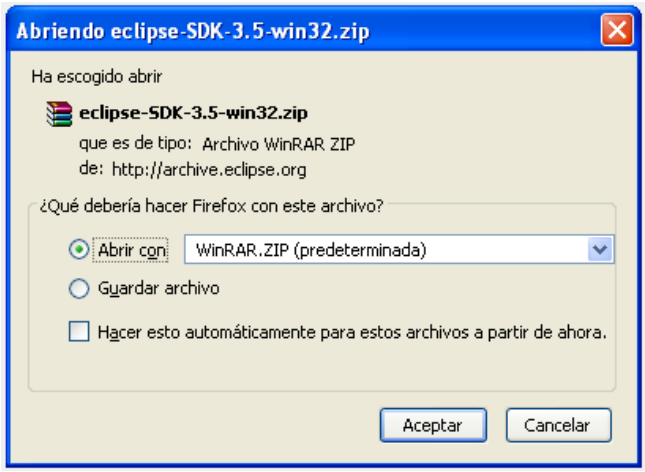
INSTALACIÓN

1º. Descargar el SDK de la página de eclipse:

http://www.eclipse.org/downloads/packages/eclipse-classic-350/galileor



Seleccionamos la opción para Windows 32-bit:

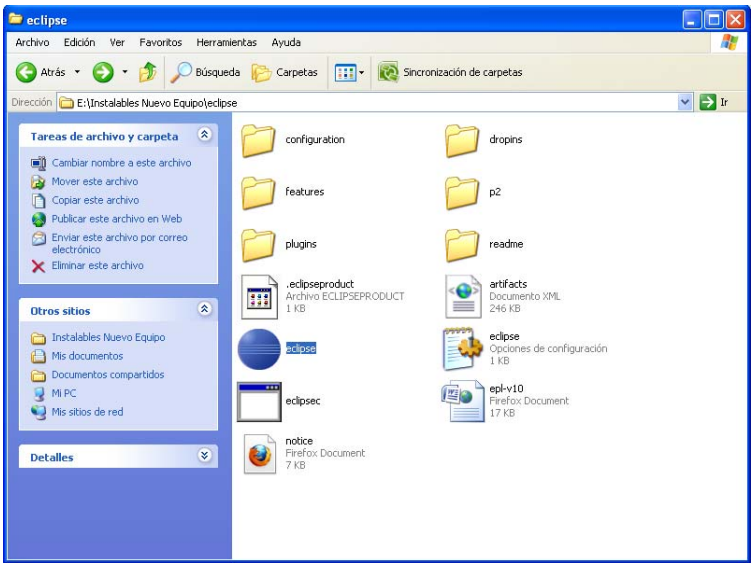


Obteniendo un fichero zip: eclipse-SDK-3.5-win32.zip

2º Para instalar eclipse, sólo tenemos que descomprimir este archivo en la ubicación deseada, en nuestro caso:

E:\Instalables Nuevo Equipo

Creándose entonces una carpeta llamada eclipse en dicha ubicación. En esta carpeta tenemos el icono de eclipse que lanza la aplicación:



A partir de aquí Eclipse ya está instalado en nuestro equipo, sólo tendremos que seleccionar una ubicación para guardar nuestro Workspace y empezar a programar.

8. ICEFACES

Para poder correr ICEfaces en nuestro equipo, necesitamos tener previamente instalados:

- Una plataforma Java (JEE) con versión 1.4.2 o mayor. En nuestro caso, hemos instalado previamente el SDK 3.5 y podemos comprobar que nuestra versión de java es 1.6.0_24, introduciendo en la ventana de comandos lo siguiente:

  java –versión

- Un servidor web que soporte servlets y Java Server Pages (JSP), se recomienda Tomcat versión 6 o mayor, ya que los códigos ICEfaces han sido extensamente probados en Tomcat. En nuesto caso, hemos instalado previamente el apache-tomcat versión 6.0.32 como se indica en el punto anterior del manual de instalación.
- Un navegador web. Se ha comprobado la compatibilidad de ICEfaces con los siguientes navegadores:

Vendor	Product	Version
Apple	Safari	1.3, 2.x, 3.x, 4.x
Google	Chrome	1.0
Microsoft	Internet Explorer	6.x+, 7.0, 8.0
Mozilla	Firefox	1.x+, 2.0, 3.0
Opera	Opera	9.x+

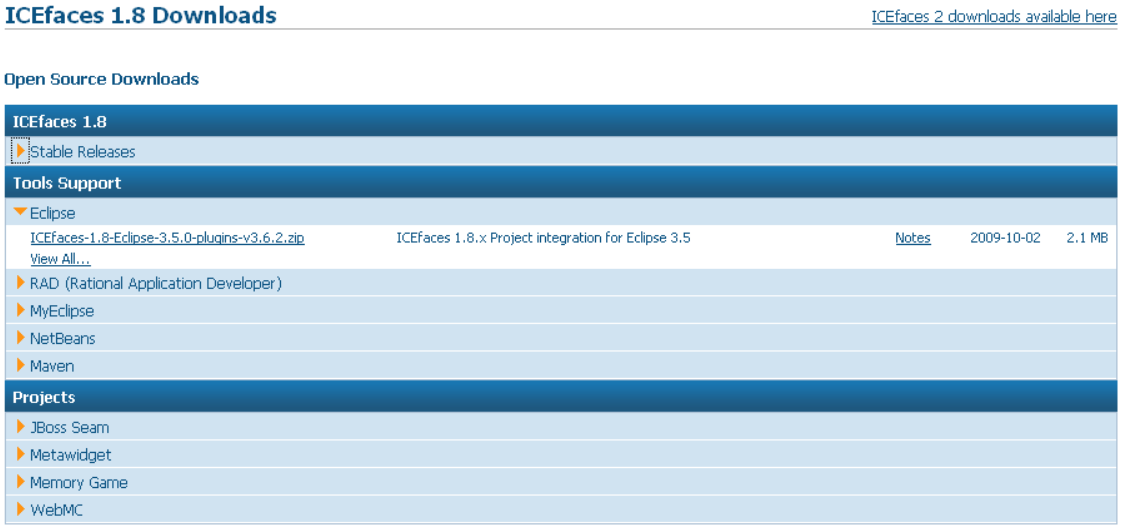
Nosotros usaremos Firefox 3.6.8, con lo cual no debería haber ningún problema.

ICEfaces está disponible como una versión completa (en edición normal o en edición empresa) o bien como plug-in para diferentes interfaces de desarrollo. En nuestro caso, como usamos Eclipse, elegiremos el plug-in de ICEfaces 1.8 para Eclipse 3.5

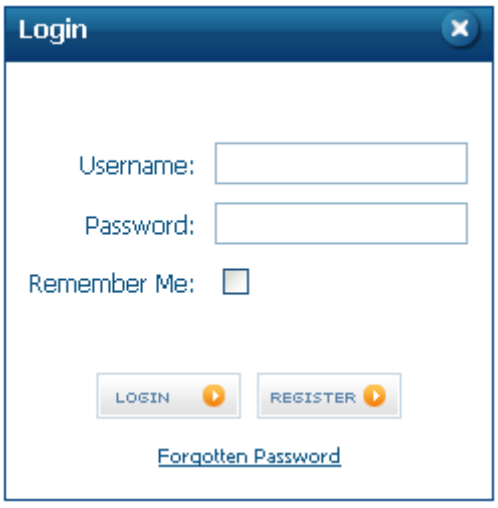
INSTALACIÓN

1º. Descargar el plug-in de ICEfaces 1.8 para Eclipse 3.5, desde la página de ICEfaces:

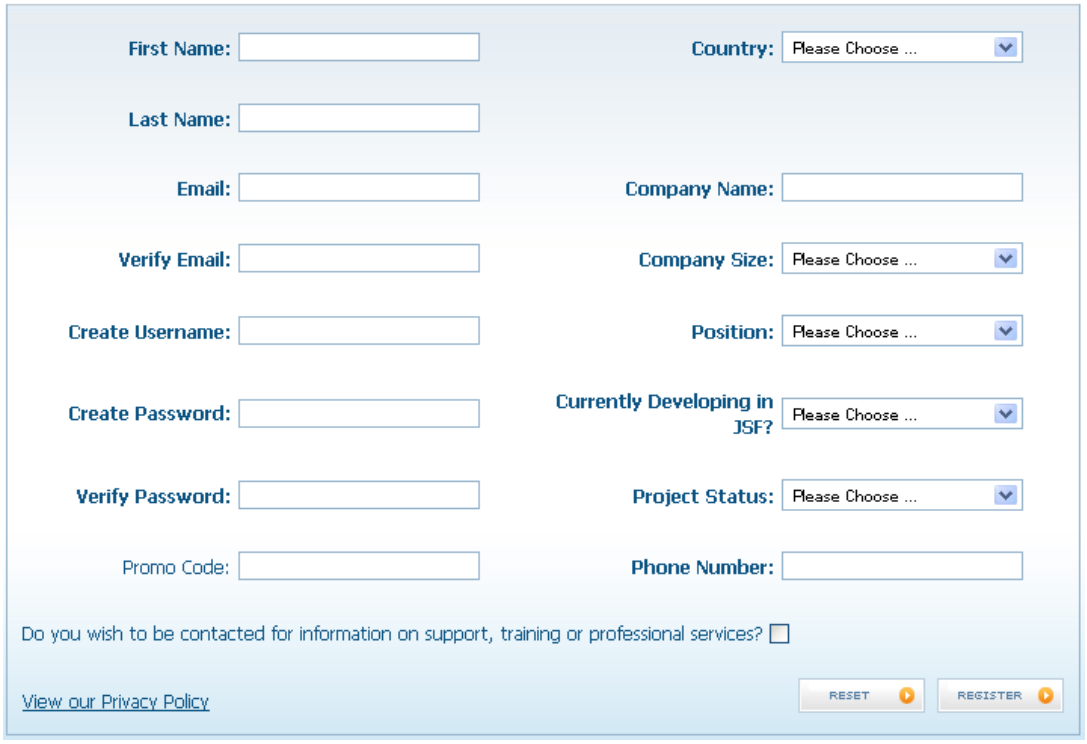
http://www.icefaces.org/main/downloads/os-downloads-1-8.iface



A la hora de realizar la descarga, se nos pide un nombre de usuario y contraseña:



para conseguir dicho nombre de usuario y contraseña hay que llevar a cabo un registro, rellenando el siguiente formulario:



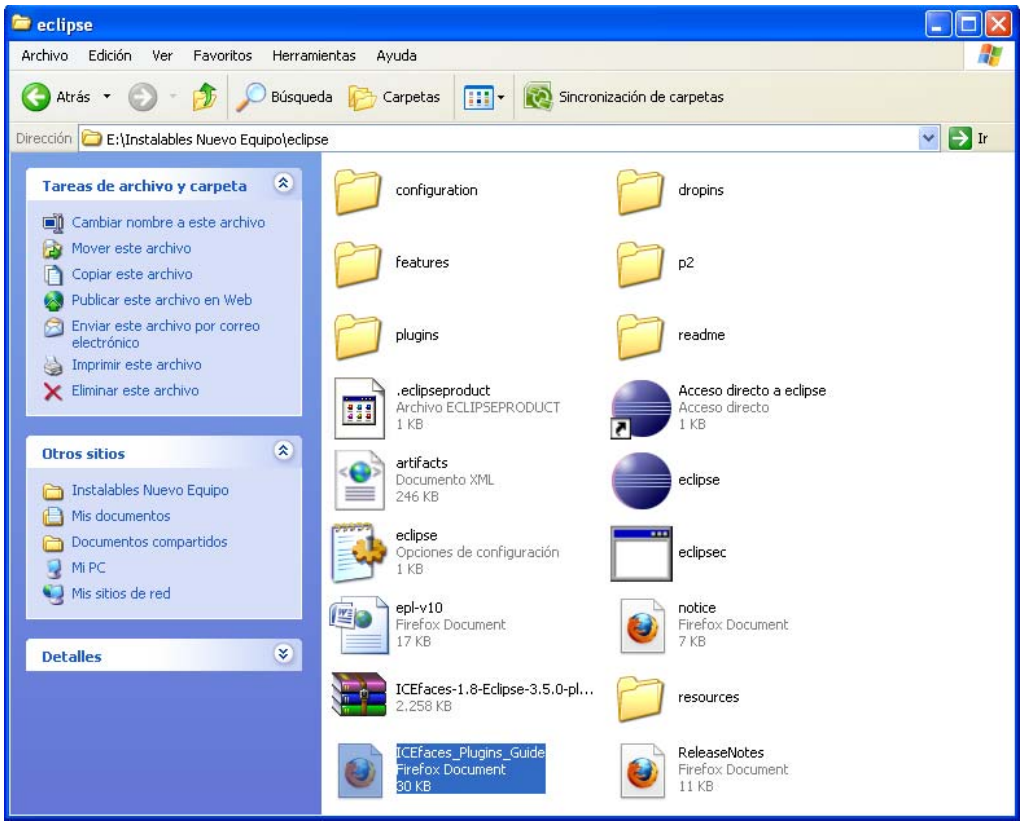
Una vez hecho esto podremos realizar la descarga, obteniendo un archivo .zip: ICEfaces-1.8-Eclipse-3.5.0-plugins-v3.6.2.zip

2º. Cerramos Eclipse (en caso de que lo tuviésemos abierto).
3º. Copiamos el archivo .zip dentro del directorio raíz de Eclipse, que en nuesto caso es:

E:\Instalables Nuevo Equipo\eclipse

4º. Descomprimimos el ZIP en este mismo directorio.

A partir de aquí, ya podemos abrir Eclipse y crear nuestro primer proyecto con ICEfaces. Al descomprimir el contenido del archivo ICEfaces-1.8-Eclipse-3.5.0-plugins-v3.6.2.zip, encontramos un documento Firefox (html) llamado ICEfaces_Plugins_Guide que contiene una guía de cómo crear un proyecto ICEfaces paso a paso.


eclipse
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Carpetas Sincronización de carpetas
Dirección E:\Instalables Nuevo Equipo\eclipse
Ir
Tareas de archivo y carpeta
Cambiar nombre a este archivo
Mover este archivo
Copiar este archivo
Publicar este archivo en Web
Enviar este archivo por correo electrónico
Imprimir este archivo
Eliminar este archivo
Otros sitios
Instalables Nuevo Equipo
Mis documentos
Documentos compartidos
Mi PC
Mis sitios de red
Detalles
configuration
dropins
features
p2
plugins
readme
.eclipseproduct
Archivo ECLIPSEPRODUCT
1 KB
Acceso directo a eclipse
Acceso directo
1 KB
artifacts
Documento XML
246 KB
eclipse
eclipse
Opciones de configuración
1 KB
eclipsec
epl-v10
Firefox Document
17 KB
notice
Firefox Document
7 KB
ICEfaces-1.8-Eclipse-3.5.0-pl...
2.258 KB
resources
ICEfaces_Plugins_Guide
Firefox Document
30 KB
ReleaseNotes
Firefox Document
11 KB

ANEXO D. MANUAL DE USUARIO

Está aplicación ha sido creada para la monitorización de sensores en medios acuáticos que permita a los usuarios controlar el estado y evolución de estos medio ambientes. Se trata de una aplicación web de fácil uso y manejo. A continuación se describen las pautas a seguir para su utilización.

1.11. 1. Mapa y cuadro de medidas

En la **¡Error! No se encuentra el origen de la referencia.** podemos ver la página de Inicio de la aplicación de monitorización:

-- 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 17: Página principal de la aplicación

En esta página principal se muestra un mapa de la zona a monitorizar en el que podemos observar la localización de los sensores en tiempo real. Podemos interaccionar con este mapa como con cualquiera de Google Maps. Las siguientes acciones pueden ser realizadas sobre el mapa:

● Hacer zoom in o zoom out (acercar o alejar la imagen) y desplazarnos por el mapa en cualquier dirección mediante el control situado en la esquina derecha del mapa.

● Cambiar la vista del mapa a las siguientes modalidades:

- Modo Mapa: mapa político de calles y carreteras (modo por defecto).

- Modo Satélite: mapa de imágenes satélite.
- Modo Híbrido: mapa de calles y carreteras sobre el fondo de imágenes satélite (combinación de los dos modos anteriores).

● Conocer el identificador y coordenadas concretas de cualquiera de los sensores haciendo click sobre ellos. Esto abre una pequeña ventana de información en la que se muestra dicha información. Para cerrar esta ventana sólo hay que hacer click sobre el aspa que hay en la esquina superior derecha de la ventana.

Además del mapa podemos observar en esta página principal una tabla en la que aparecen valores, es el cuadro de medidas, en el que se muestran en tiempo real los valores de los parámetros del agua medidos por los sensores, así como otras informaciones sobre los sensores como su tipo, identificador y coordenadas geográficas.

Para navegar por la aplicación tenemos los botones: MAP, HISTORIC CHART y PROFILE CHART, que nos dirigen en cualquier momento a la parte de la aplicación que deseemos ver.

1.12. *2. Gráficos históricos*

Para dirigirnos a la parte de los gráficos históricos sólo hay que hacer click en el botón HISTORIC CHART, que nos lleva a la página mostrada en la Figura 18.

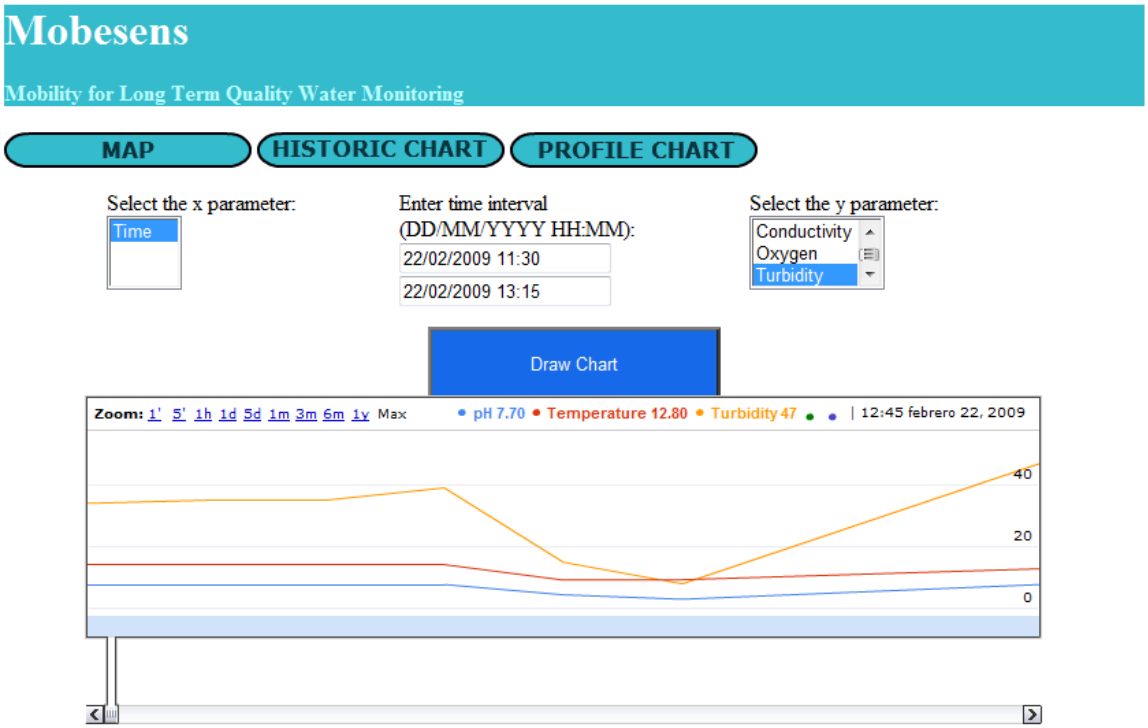


Figura 18: Página de los gráficos históricos

Para crear un gráfico histórico sólo tenemos que:

- Seleccionar los parámetros, que queremos visualizar en función del tiempo, del menú “Select y parameter”. Para seleccionar varios parámetros sólo hay que hacer click en todos aquellos que se desee pulsando al mismo tiempo la tecla CTRL del teclado.
- Introducir el intervalo de tiempo a mostrar en los campos destinados a ello siguiendo el formato indicado (DD/MM/YYYY HH:MM), es decir poner la fecha como Día/Mes/Año Hora:Minuto.
- Hacer click en el botón DRAW CHART.

El gráfico histórico es interactivo, podemos hacer zoom mediante la barra deslizante situada debajo del gráfico. Además si pasamos el ratón por encima de las líneas dibujadas en el gráfico, que corresponden a los diferentes parámetros del agua, podemos ver el valor exacto de las medidas para un instante concreto de tiempo.

1.13. *3. Gráficos de perfil*

Para dirigirnos a la parte de los gráficos de perfil hacemos click en el botón PROFILE CHART y se nos presenta la página de la Figura 19.

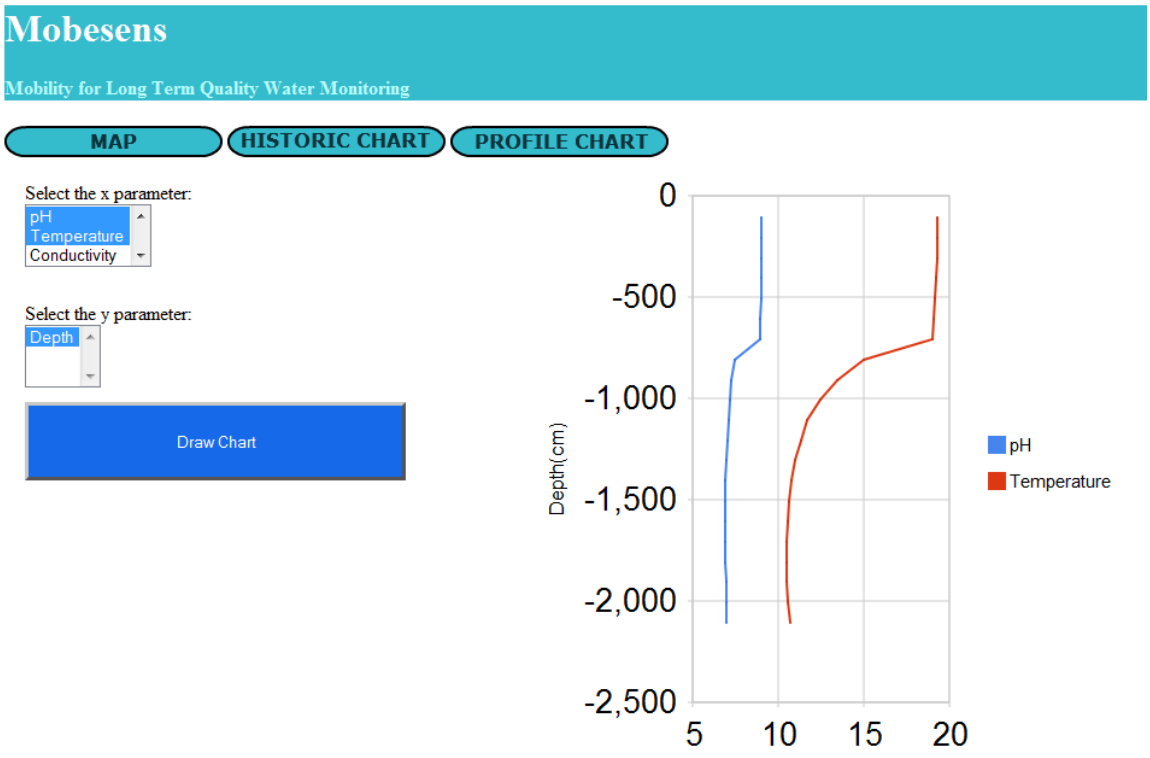


Figura 19: Página de los gráficos de perfil

Como en el caso de los gráficos históricos, para crear el gráfico de perfil tenemos que:

- Seleccionar los parámetros que queremos visualizar en función de la profundidad mediante el menú “Select x parameter”.
- Hacer click en el botón DRAW CHART.