Facultad
de
Ciencias

VISUALIZANDO DATOS METEOROLÓGICOS: DESARROLLO DE UN CLIENTE Y SERVIDOR WMS

(Viewing meteorological data: development of a WMS client and server)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Alba Muñiz Revuelta

Director: Patricia López Martínez

Co-Director: Daniel San Martín

Febrero - 2016

INDICE

RESUMEN

La empresa *Predictia* tiene desarrollado un servidor WMS (Web Map Service) con una serie de funcionalidades que permiten crear diferentes capas para la representación de datos meteorológicos sobre mapas. Estas funcionalidades se dividen en diferentes tipos que dependen del estilo de visualización que se desea utilizar. Para la visualización basada en capas "vectoriales" se utiliza Mapnik y se accede a una fuente externa para abastecerse de los datos, mientras que para la visualización basada en capas "raster" se utiliza también Mapnik pero se accede a los datos a través de ficheros NetCDF (Network Common Data Form).

El trabajo que se ha realizado se divide en dos partes:

- Por el lado del servidor se han añadido dos funcionalidades nuevas. La primera, utilizando Java para leer ficheros NetCDF, consiste en generar un fichero JSON a partir de los datos obtenidos de estos ficheros, con la información necesaria para representar de forma animada datos de viento. La segunda, implementada en Python y usando también ficheros NetCDF, tiene por objeto la generación de capas que permiten la visualización de los datos del viento en diferentes formatos, así como de las zonas de altas y bajas presiones. Esta última funcionalidad se ha desarrollado de forma genérica, de manera que pueda ser fácilmente extendida con otros tipos de estilos de representación.

- Por el lado del cliente se ha desarrollado un cliente web que utiliza cada una de las funcionalidades expuestas anteriormente, dibujando las capas obtenidas a través del servidor sobre un mapa. Este cliente es bastante configurable, pudiendo el usuario elegir el tipo de mapa de fondo, las capas que se quieren representar, la fecha en la que se hizo la predicción (Runtime), la fecha/hora de la predicción que se desea visualizar, etc. El cliente proporciona también un meteograma para poder visualizar los datos en forma de serie temporal.

Las tecnologías y lenguajes que se han utilizado para el desarrollo de este proyecto son JavaScript, WMS, Python, HTML, CSS, Java, así como bastantes librerías relacionadas con estas tecnologías.

Palabras Clave: Meteorología, aplicación web, cliente, servidor, fichero NetCDF, WMS.

ABSTRACT

The *Predictia* Company has developed a WMS (Web Map Service) server with several functionalities that allow to create different layers to represent meteorological predictions in maps. These functionalities are divided in different types depending on the visualization style. For "vectorial" layers, the server uses Mapnik and the data are obtained from an external database. Mapnik is also used for the "raster" layers but the data are obtained from NetCDF (Network Common Data Form) files.

The work accomplished during the project is divided in two parts:

- On the server side, two new features have been added. The first, based on Java and NetCDF files manipulation, consists in generating a JSON file from data obtained from NetCDF files, which can represent in an animated way the wind prediction. The second, implemented in Python and based also on NetCDF files, aims to generate layers that allow displaying wind data in different formats, as well as areas of higher and lower pressures. This last feature has been developed in a generic way, so that it can be easily adapted to other types of predictions.

- On the client side, a web client has been developed that uses the new server functions to draw the obtained layers on a map. This client is quite configurable so his user can choose the type of background map, the layers to be represented in the map, the runtime control and the prediction time. The client also provides a meteogram to display data as a graph.

Several technologies and languages have been used for the development of this project such as JavaScript, WMS, Python, HTML, CSS, Java, and many libraries related to these technologies.

Keywords: Meteorology, web application, client, server, NetCDF file, WMS.

1. INTRODUCCIÓN Y OBJETIVOS

1.1. INTRODUCCIÓN

Hoy en día existe un interés generalizado en las predicciones meteorológicas tanto por motivos de ocio como laborales. La necesidad de conocer la previsión del tiempo para la preparación y realización de acciones y actividades abarca un amplio abanico de posibilidades, que van desde los deportes que dependen de las olas o el viento para su realización, la recolección de cosechas o vendimias, la pesca, pruebas tecnológicas en ambientes controlados, la determinación de la táctica a seguir en una carrera de coches, o el desafío de una escalada. Pero si solo se proporcionasen los datos numéricos obtenidos de los diferentes aparatos de medición, únicamente un experto sería capaz de interpretarlos.

Comprender datos meteorológicos de manera no gráfica, para personas sin conocimientos meteorológicos, es casi imposible. Para facilitar la comprensión de estos datos, se crearon las representaciones meteorológicas que todos conocemos, a través de la información de tiempo en los informativos de televisión o a través de la visita a alguna página web de predicción meteorológica como puede ser la de la AEMET [1] (Agencia Estatal de Meteorología).

Predictia [2] ha creado un servicio web que aporta una serie de información, o más exactamente, proporciona capas de información con datos meteorológicos que se pueden utilizar posteriormente para mostrar estos datos de manera gráfica sobre un mapa, con la posibilidad de visualizar diferentes datos a la vez y de poder configurar lo que queremos ver o no.

Para implementar esta funcionalidad se ha creado un servidor WMS [3], que es quién proporciona una serie de funciones para la creación de las diferentes capas de predicción que posteriormente se usarán en los clientes o simplemente para el procesamiento de datos meteorológicos, de manera que sea el cliente el que cree la capa con los datos obtenidos del servidor.

Las capas que genera actualmente el servicio pueden tener dos formatos, Vectorial o Raster, y en ambos casos se utiliza Mapnik [4] para su implementación. Tanto Vectorial como Raster son formas de almacenamiento de la información. El formato Vectorial es una forma de codificación de la información mediante formas geométricas (una curva, un polígono, un punto, etc. con sus atributos) mientras que la codificación Raster almacena los valores de la información pixel a pixel en una rejilla (una imagen, por ejemplo). En la Ilustración 1 se muestra la diferencia entre una representación y otra.

Mapnik es una herramienta para desarrollar aplicaciones cartográficas que facilita la creación de las capas que se van a representar sobre los mapas. Por su rendimiento, es muy utilizada en el desarrollo de servicios web de alta demanda como pueden ser los proyectos OpenStreetMap, CartoDB o MapBox.

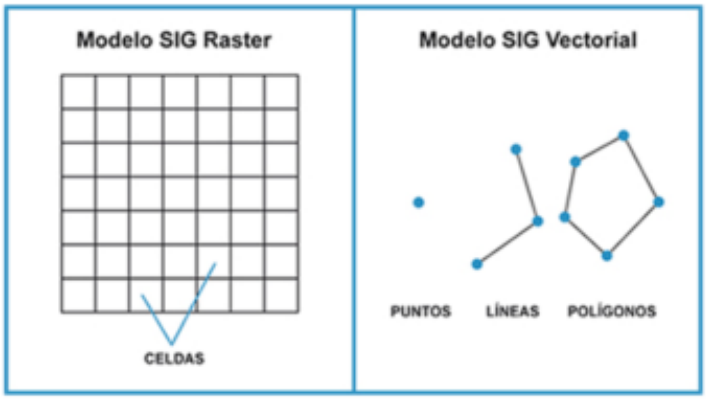

Ilustración 1 Modelo Raster y Modelo Vectorial[1]

Para demostrar la utilidad de la información proporcionada por el servidor WMS desarrollado por Predictia y como parte del trabajo abordado a lo largo del proyecto, se ha desarrollado un cliente WMS que expone las capas obtenidas del servidor sobre un mapa para identificar la localización de la predicción más fácilmente.

1.2. OBJETIVO

El objetivo principal del proyecto consiste en contribuir a la visualización de datos meteorológicos a través de la extensión de la funcionalidad de un servidor WMS y del desarrollo de un cliente web que explote la información proporcionada por el servidor. De esta forma se podrán representar los datos meteorológicos seleccionados de manera muy visual.

Para conseguir este objetivo se definen los siguiente sub-objetivos:

- Por el lado del servidor añadir dos funcionalidades nuevas:
    - La primera consiste en procesar los datos de las variables *u* (componente zonal) y *v* (componente meridional) del viento que nos proporcionan los ficheros NetCDF [5], para crear un fichero JSON que permita desde un cliente crear una capa de viento animada para un área geográfica determinada. La lectura de los datos será desarrollada en Java.
    - La segunda consiste en la generación de varias capas, en este caso para la representación del viento con dos formatos de iconos diferentes y para la identificación de altas y bajas presiones. Esta funcionalidad será implementada en Python y basada en manipulación de ficheros NetCDF. Esta funcionalidad se ha de desarrollar de manera que los cambios necesarios para añadir nuevos tipos de capas sean lo más fáciles posible.
- Por el lado del cliente implementar un cliente web que facilite la visualización de las capas proporcionadas por el servidor al pintarlas sobre una interfaz con un mapa, para que sea más fácil la localización de la información. En esta interfaz

[1] http://sig.cea.es/tipos_SIG

se añadirán una serie de controles para poder seleccionar el tipo de mapa base, las capas que se desea visualizar, la fecha en la que se realizó la predicción y/o la fecha/hora de la predicción. El cliente además, deberá generar una capa, la capa de viento animada, utilizando el JSON creado por el servidor.

La interfaz de la aplicación cliente deberá ser muy configurable, pudiendo el usuario modificar el aspecto y capacidades de la propia interfaz simplemente modificando los datos de un fichero de configuración textual. De esta manera, el usuario de la aplicación podrá decidir dónde aparece cada control en la interfaz, si aparece o no, qué estilos de capa va a permitir mostrar o cuáles de ellas se van a poder seleccionar, entre otras cosas.

También por el lado del cliente se proporcionará una interfaz con un meteograma para poder visualizar los datos en forma de serie temporal, también altamente configurable a través de su correspondiente fichero de configuración.

Todo el código implementado se deberá desarrollar de la manera más genérica y modular posible, para que en el caso de añadir nuevas funcionalidades, este trabajo sea lo más fácil posible.

1.3. ORGANIZACIÓN DE LA MEMORIA

En el capítulo 2, Tecnologías y Materiales, se explica que tecnologías y herramientas se han usado durante el desarrollo del proyecto y en qué consiste cada una de ellas. Asimismo, se explica la metodología que se ha seguido para el desarrollo del proyecto.

En el capítulo 3, Requisitos, se documenta la primera fase del ciclo de vida de un proyecto de software. Se listan los requisitos que se van a desarrollar durante el proyecto, tanto funcionales como no funcionales. Mediante diagramas de casos de uso se muestra el comportamiento del sistema en base a interacciones entre los actores y el sistema. Se muestra la especificación de alguno de los casos de uso no triviales mediante plantillas. Se expone una tabla en la que se especifica qué trabajo se ha realizado en cada iteración y a qué requisitos corresponde.

En el capítulo 4, Arquitectura, se explica qué tipo de arquitectura se ha seguido para desarrollar el proyecto. Mediante el diagrama de componentes y conectores se expone qué componentes tiene el sistema, cómo se relacionan y el papel que tiene cada uno de ellos. Y mediante un diagrama de despliegue se muestra el esquema de cómo se ha desplegado el proyecto una vez finalizado.

En el capítulo 5, Implementación, se explica cómo se ha implementado el proyecto mostrando fragmentos de código e imágenes del resultado final.

En el capítulo 6, Pruebas, se exponen los distintos tipos de pruebas que se han realizado al proyecto y cómo se han desarrollado cada una de ellas.

En el capítulo 7, Conclusiones y Trabajos Futuros, se hace un pequeño resumen de las conclusiones sacadas tras finalizar el proyecto y qué trabajos se pueden plantear en un futuro para añadir o mejorar el proyecto.

Se finaliza con el apartado de referencias.

2. TECNOLOGÍAS Y MATERIALES

A continuación se expone un pequeño resumen de las tecnologías, herramientas y metodología utilizadas para el desarrollo de este proyecto.

2.1. TECNOLOGÍAS

2.1.1. JAVA

Java [6] es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra, siempre y cuando la plataforma esté dotada de una máquina virtual Java.

2.1.2. PYTHON

Python [7] es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

2.1.3. JAVASCRIPT

JavaScript [8] (abreviado comúnmente "**JS**") es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web, permitiendo introducir mejoras en la interfaz de usuario y páginas web dinámicas.

2.1.4. CSS

Hoja de estilo en cascada o **CSS** [9] (siglas en inglés de *cascading style sheets*) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML.

2.1.5. HTML

HTML [10], siglas de ***HyperText Markup Language*** («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, entre otros.

2.1.6. WMS

El **servicio Web Map Service (WMS)** es un servicio de mapas con interfaz estándar OpenGIS Web que ofrece una sencilla interfaz HTTP para solicitar imágenes de mapas georegistrados de una o más bases de datos geoespaciales distribuidas. Una petición WMS define la capa y el área geográfica de interés para ser procesado.

2.1.7. LEAFLET

Leaflet [11] es una librería JavaScript de código abierto para la construcción de aplicaciones de mapas web. Funciona de manera eficiente en todas las principales plataformas de escritorio y móvil, se puede ampliar con una gran variedad de plugins y tiene una API bien documentada.

2.1.8. D3js

D3.js [12] (**D3** for **Data-Driven Documents**) es una librería de JavaScript para producir visualizaciones de datos dinámicos e interactivos en los navegadores web. Hace uso de SVG, HTML5 y CSS. D3.js permite un gran control sobre el resultado visual final.

2.1.9. MATPLOTLIB + BASEMAP

Matplotlib [13] es una librería para el trazado de datos en 2D en los mapas de Python.

BaseMap no realiza ningún trazado por sí mismo, sino que proporciona las herramientas necesarias para transformar coordenadas de una de las 25 proyecciones cartográficas diferentes.

Matplotlib se utiliza por lo tanto para trazar contornos, imágenes, vectores, líneas o puntos en las coordenadas transformadas.

2.1.10. JUnit

JUnit [14] es un framework que permite realizar la ejecución de pruebas de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase es el esperado. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

2.1.11. SPRING

Spring [15] es un framework para el desarrollo de aplicaciones empresariales y contenedor de inversión de control, de código abierto para la plataforma Java.

Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web sobre la plataforma Java EE. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (Enterprise JavaBean).

2.1.12. MAVEN

Maven [16] es una herramienta de construcción de software que utiliza ficheros POM para describir el proyecto software a construir, sus dependencias de otros módulos y componentes externos. Este permite compilar, ejecutar pruebas o realizar distribuciones, tratando de forma automática las dependencias del proyecto.

2.2. HERRAMIENTAS

2.2.1. SUBLIME TEXT

SublimeText [17] es un editor de texto y de código fuente.

2.2.2. ECLIPSE

Eclipse [18] proporciona entornos de desarrollo y plataformas para casi todos los lenguajes de programación. Es famoso por sus IDE para Java, C / C ++, JavaScript y PHP construidos sobre plataformas extensibles para la creación de entornos de desarrollo de escritorio, Web y de nube.

2.2.3. SUBVERSION

Apache Subversion [19] (abreviado frecuentemente como **SVN**, por el comando *svn*) es una herramienta de control de versiones de código abierto basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio.

2.2.4. MAGIC DRAW

MagicDraw UML [20] es una herramienta de modelado UML desarrollada por NoMagic. Con esta herramienta se puede modelar datos, desarrollar código en varios lenguajes de programación y sigue el estándar UML 2.3.

2.2.5. NetCDF Tools

NetCDF Tools es una herramienta gratuita para visualizar datos NetCDF.

2.3. METODOLOGÍA

Para este proyecto se ha seguido una metodología iterativa incremental [26], que como muestra la Ilustración 2, es un proceso de desarrollo de software que se basa en realizar cada una de las fases del desarrollo software a pequeña escala repetidamente hasta completar el proyecto.

Esta metodología es una de las más utilizadas en la actualidad. El proyecto se planifica en bloques de menor complejidad que se denominan iteraciones. En cada una de las iteraciones se realizan todas las etapas del desarrollo software, las cuales se inician con el análisis de requisitos y finalizan con la fase de pruebas de forma que cada una de ellas aporte beneficios al proyecto de manera incremental.

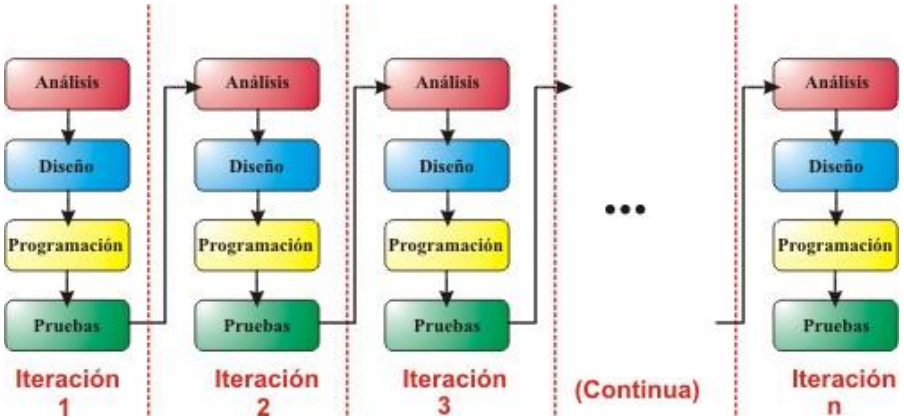

Ilustración 2 Desarrollo iterativo[2]

Lo que se busca con esta metodología es facilitar la labor de desarrollo. Si el proyecto es complejo y no se va comprobando los resultados poco a poco, se deberá revisar el proyecto entero y seguramente modificar mucho código, si al realizar la fase de pruebas al final el sistema falla. De esta manera al hacerlo en bloques de menor complejidad e ir probando el funcionamiento al final de cada bloque, en caso de error, será fácil localizar donde está el fallo y no será necesario modificar mucho trabajo del ya realizado.

Una de las maneras de decidir en qué orden se van a ir implementado los requisitos es la de priorizar objetivos en función del valor que le dé a cada requisito el cliente.

Más adelante se establecen dichas iteraciones y cuál ha sido el trabajo desarrollado en cada una de ellas en el caso de este proyecto.

[2] https://www.mindmeister.com/es/198054122/modelos-de-proceso-de-ingenier-a-de-software

3. REQUISITOS

En este apartado se lista cada uno de los requisitos funcionales desde el punto de vista tanto del servidor como del cliente, así como la parte del meteograma. También se aborda la especificación de los requisitos no funcionales.

La elicitación de requisitos se llevó a cabo mediante una serie de reuniones con el jefe de proyecto, en las que se expusieron las nuevas funcionalidades que querían añadir a su sistema, las cuales iban a ser implementadas en este proyecto.

3.1. LISTA DE REQUISITOS

3.1.1. REQUISITOS DEL SERVIDOR

ID	Nombre	Descripción
RFS001	Obtener Datos	El sistema deberá obtener los datos necesarios de un fichero NetCDF.
RFS002	Procesar Datos	El sistema deberá seleccionar los datos adecuados dependiendo del tipo de capa.
RFS003	Crear Capa	El sistema deberá crear una capa de visualización de los datos obtenidos.
RFS004	Crear Fichero JSON	El sistema deberá crear un fichero JSON a partir de ficheros NetCDF con los datos necesarios dependiendo del tipo de capa.

3.1.2. REQUISITOS DEL CLIENTE

ID	Nombre	Descripción
RFC001	Mostrar Mapa	El sistema deberá mostrar un mapa.
RFC001.1	Elegir estilo del Mapa	El usuario podrá elegir entre diferentes tipos de mapas.
RFC001.2	Elegir capa a mostrar	El usuario podrá elegir las diferentes capas a mostrar sobre el mapa.
RFC001.3	Hacer zoom en el mapa	El usuario podrá hacer zoom en el mapa.
RFC001.4	Moverse por el mapa	El usuario podrá moverse por el mapa.
RFC001.5	Seleccionar fecha/hora de predicción	El usuario podrá seleccionar la fecha/hora de la predicción que desea visualizar.
RFC001.6	Predicciones en movimiento	El usuario podrá observar cómo van cambiando las predicciones en el tiempo.
RFC001.7	Seleccionar fecha/hora de realización de predicción	El usuario podrá seleccionar la fecha/hora en que se realizó la predicción.
RFC001.8	Cambiar opacidad de la capas	El usuario podrá cambiar la opacidad de cada capa.
RFC001.9	Control de movimiento	El sistema deberá mostrar botones de play, pause, avance y retroceso de las predicciones.
RFC001.10	Cambiar velocidad de reproducción	El usuario podrá cambiar la velocidad de la reproducción.
RFC001.11	Crear capa animada	El sistema deberá crear una capa animada del viento con los datos obtenidos de un

		fichero JSON.
RFC001.12	Adelantar capa	El usuario podrá adelantar una de las capas seleccionadas.
RFC002	Configurar Interfaz Mapa	El usuario podrá cambiar la configuración de la interfaz a través de un fichero de configuración.
RFC002.1	Configurar dirección del fichero WMS	El usuario podrá configurar la dirección de donde obtener los datos.
RFC002.2	Configurar localización del mapa	El usuario podrá configurar la posición inicial del mapa.
RFC002.3	Configurar zoom del mapa	El usuario podrá configurar el zoom inicial del mapa.
RFC002.4	Configurar posición de la leyenda	El usuario podrá configurar la posición de las leyendas.
RFC002.5	Configurar posición del menú	El usuario podrá configurar la posición del menú de capas.
RFC002.6	Configurar la expansión del menú	El usuario podrá configurar si el menú inicialmente está expandido.
RFC002.7	Configurar visibilidad de la leyenda	El usuario podrá configurar si la leyenda se ve o no.
RFC002.8	Configurar capas iniciales	El usuario podrá configurar las capas activas inicialmente.
RFC002.9	Configurar formato de las capas	El usuario podrá configurar el formato de las capas.
RFC002.10	Configurar posición del control	El usuario podrá configurar la posición del control de tiempos.
RFC002.11	Configurar valores de tiempos	El usuario podrá configurar el rango de tiempos a mostrar (ej.: 100 primeras fechas).
RFC002.12	Configurar valores por defecto de los tiempos	El usuario podrá configurar los tiempos por defecto a mostrar.
RFC002.13	Configurar visibilidad del selector de fecha	El usuario podrá configurar la visibilidad del selector de las fechas en la que se realizaron las predicciones.
RFC002.14	Configurar fechas en las que se realizaron las predicciones	El usuario podrá configurar el rango de fechas en la que se realizaron las predicciones a mostrar (ej. 100 primeras fechas).
RFC002.15	Configurar valores por defecto de la fecha en la que se realizaron predicciones	El usuario podrá configurar la fecha en la que se realizaron las predicciones por defecto a mostrar.
RFC002.16	Configurar periodo	El usuario podrá configurar el periodo entre las fechas/hora en las que se realizaron las predicciones.
RFC002.17	Configurar formato del selector de fechas en las que se realizaron las predicciones	El usuario podrá configurar formato del selector de fechas en las que se realizaron las predicciones.

RFC003	Mostrar Meteograma	El sistema deberá mostrar datos en forma de gráfico.
RFC004	Configurar Interfaz Meteograma	El usuario podrá cambiar la configuración del meteograma a través de un fichero.
RFC004.1	Configurar márgenes del gráfico	El usuario podrá configurar los márgenes.
RFC004.2	Configurar número de valores en arista	El usuario podrá configurar el número de valores por arista.
RFC004.3	Configurar la visualización de cada arista	El usuario podrá configurar la visualización de cada arista.
RFC004.4	Configurar la localización de cada arista	El usuario podrá configurar la localización de cada arista.
RFC004.5	Configurar formato de cada arista	El usuario podrá configurar el formato de cada arista.
RFC004.6	Configurar características representación de los datos	El usuario podrá configurar las características de representación de los datos dependiendo del tipo de éste.

3.1.3. REQUISITOS NO FUNCIONALES

ID	Descripción	Tipo
RNFC001	El sistema deberá visualizarse de manera similar independientemente del navegador que se use.	Compatibilidad
RNFC002	El sistema deberá permitir ampliaciones y modificaciones.	Mantenibilidad
RNFC003	El sistema deberá tener una interfaz intuitiva.	Usabilidad
RNFS004	El sistema deberá implementar la generación del fichero JSON en Java.	Implementación
RNFS005	La generación de capas deberá ser implementada con Python.	Implementación

3.2. CASOS DE USO

Mostramos aquí el comportamiento del sistema mediante el modelado de las interacciones entre los actores y el sistema a través de casos de uso, tanto para la funcionalidad del lado del servidor como de la aplicación cliente.

3.2.1. CASOS DE USO SERVIDOR

En la Ilustración 3 se muestra el esquema de los casos de uso del servidor.

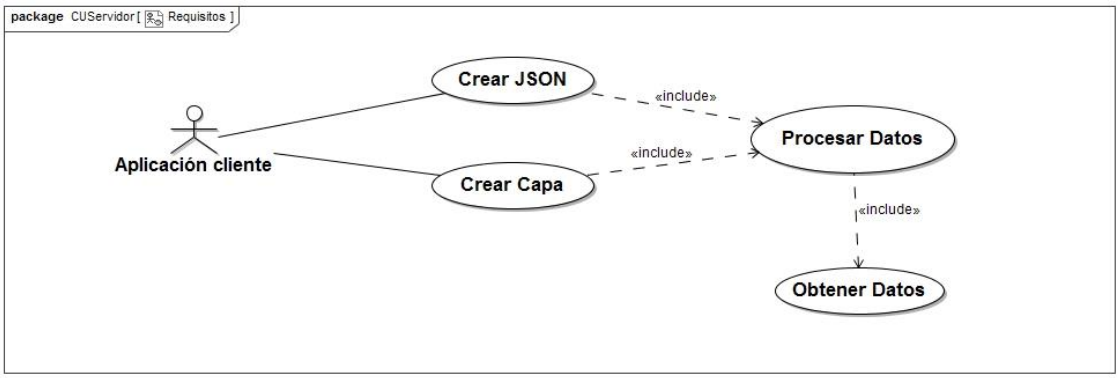

Ilustración 3 Casos de Uso del Servidor

3.2.2. CASOS DE USO CLIENTE

El modelo de casos de uso del cliente se muestra a continuación, comenzando por el diagrama de casos de uso a alto nivel (Ilustración 4) y a continuación el modelado de cada uno de los casos de uso de alto nivel a más bajo nivel, a través de las ilustraciones 5, 6 y 7.

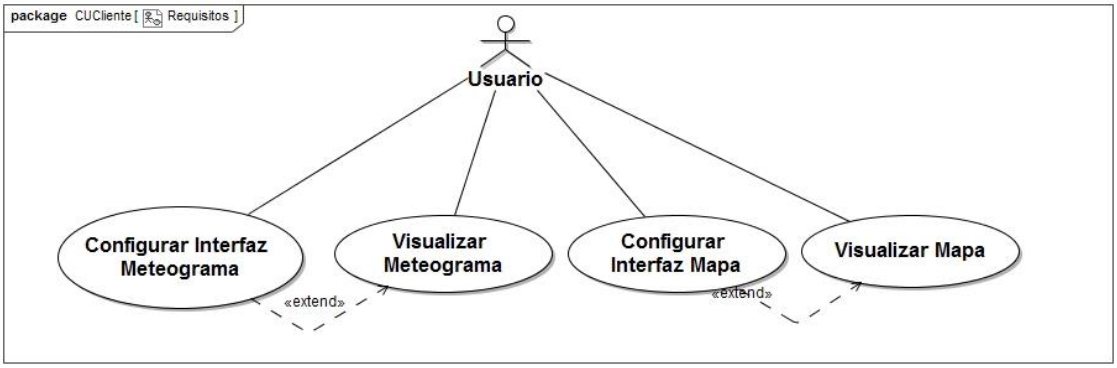

Ilustración 4 Casos de uso de Alto Nivel de la aplicación cliente

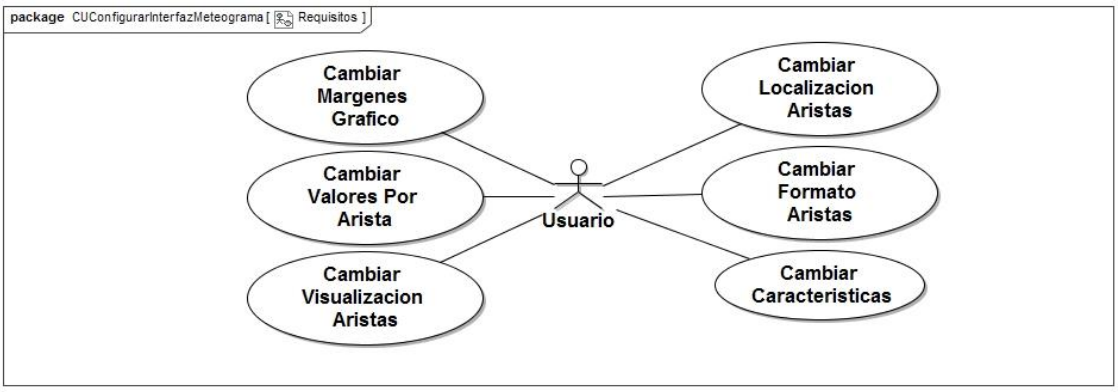

Ilustración 5 Caso de uso Configurar Interfaz Meteograma

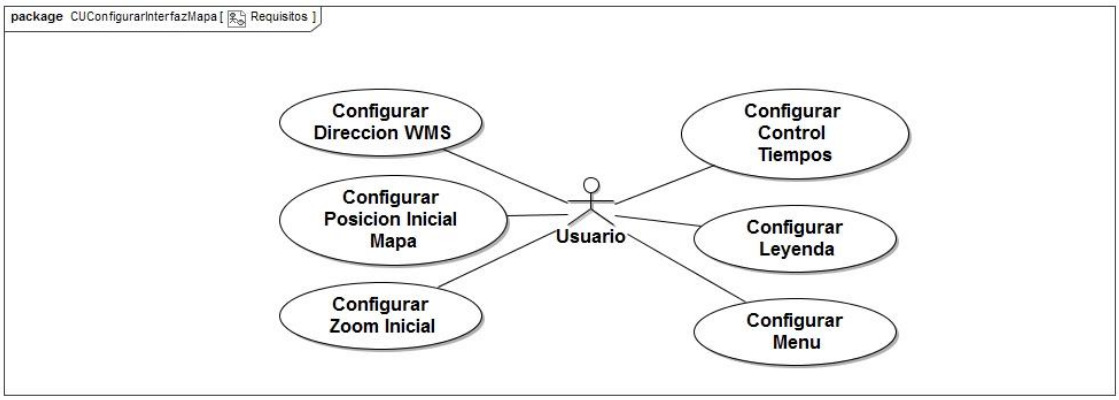

Ilustración 6 Caso de uso Configurar Interfaz Mapa

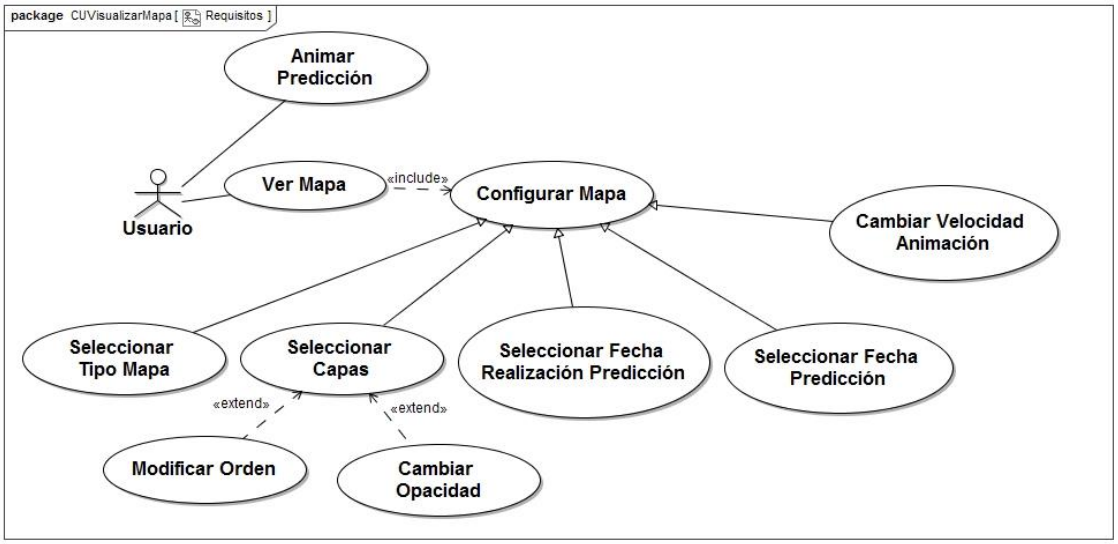

Ilustración 7 Caso de uso Visualizar Mapa

3.3. ESPECIFICACIÓN CASOS DE USO

Se presenta la especificación de alguno de los casos de uso, para una mejor comprensión de éstos. La especificación se realiza en forma de plantilla.

Nombre	Crear Capa
Autores	Alba Muñiz Revuelta
Versión	1.0
Prioridad	Media
Criticidad	Baja
Fuente	*Predictia*
Responsable	Alba Muñiz Revuelta
Actor Principal	Aplicación Cliente
Descripción	A partir de un fichero NetCDF se crea una nueva capa de predicción
Objetivo	Obtener datos meteorológicos para visualización mediante iconos
Evento de Activación	Crear capa
Precondición	Tener acceso al fichero NetCDF

Garantías Si Éxito	Se crea una nueva capa
Garantías Mínimas	Si el proceso falla no se modifica nada
Escenario Principal	1. La aplicación cliente hace la petición de los datos de la capa requerida, pasando como parámetro el nombre de los ficheros, las variables requeridas y el tipo de proyección cartográfica. 2. El sistema ejecuta el caso de uso Procesa Datos. 3. El sistema crea una nueva capa con los datos procesados. 4. El sistema retorna la capa.

Nombre	Procesar Datos
Autores	Alba Muñiz Revuelta
Versión	1.0
Prioridad	Alta
Criticidad	Baja
Fuente	*Predictia*
Responsable	Alba Muñiz Revuelta
Actor Principal	Aplicación Cliente
Descripción	El sistema deberá seleccionar los datos adecuados y procesarlos
Objetivo	Obtener solo los datos necesarios y en el formato correcto
Evento de Activación	Procesar Datos
Precondición	Tener acceso al fichero NetCDF
Garantías Si Éxito	Se procesan los datos
Garantías Mínimas	Si el proceso falla no se modifica nada
Escenario Principal	1. El sistema accede al servidor WMS. 2. El sistema obtiene el fichero NetCDF cuyo nombre se pasa como parámetro. 2. a. Si el acceso al fichero falla, se produce un error y se termina el caso de uso. 3. El sistema obtiene los datos de las variables indicadas del fichero NetCDF. 4. El sistema filtra los datos obtenidos en el paso 3 para obtener los del área geográfica indicada. 5. El sistema transforma los datos del punto 4 dependiendo de la representación cartográfica que se pasa como parámetro. 6. El sistema retorna los datos.

Nombre	Animar Predicción
Autores	Alba Muñiz Revuelta
Versión	1.0
Prioridad	Media
Criticidad	Baja
Fuente	*Predictia*
Responsable	Alba Muñiz Revuelta
Actor Principal	Usuario

Descripción	Observar cómo van cambiando las predicciones
Objetivo	Visualizar datos en movimiento
Evento de Activación	Play
Precondición	Estar viendo el mapa y tener al menos una capa seleccionada.
Garantías Si Éxito	Se muestran los datos en movimiento
Garantías Mínimas	Si el proceso falla no se modifica nada
Escenario Principal	1. El usuario pulsa el botón Play. 2. El tiempo de predicción empieza a avanzar. 3. El sistema va cambiando la capa para mostrar las distintas predicciones según la fecha/hora. 4. El sistema desliza el slider para indicar la fecha y la hora de la predicción.
Escenario Alternativo	1. a. El usuario pulsa el botón de Stop. 1. a.1. Si la animación está en proceso, ésta se para. 1. a.2. Si la animación está parada, no pasa nada. 1. b. El usuario pulsa el botón retroceder. 1. b.1. La fecha/hora de la predicción retrocede de una en una y las capas se actualizan. 1. c. El usuario pulsa el botón avanzar. 1. c.1. La fecha/hora de la predicción avanza de una en una y las capas se actualizan.

Nombre	Cambiar Opacidad
Autores	Alba Muñiz Revuelta
Versión	1.0
Prioridad	Baja
Criticidad	Baja
Fuente	*Predictia*
Responsable	Alba Muñiz Revuelta
Actor Principal	Usuario
Descripción	Cambiar la opacidad de cada capa
Objetivo	Visualizar diferentes capas a la vez
Evento de Activación	Cambiar opacidad
Precondición	Tener capas seleccionadas
Garantías Si Éxito	La capa cambiará su opacidad
Garantías Mínimas	Si el proceso falla no se modifica nada
Escenario Principal	1. El usuario selecciona la capa a la que le interesa cambiar la opacidad. 2. El sistema expande un menú. 3. El usuario selecciona el icono de la opacidad. 4. El sistema muestra un slider con la opacidad. 5. El usuario modifica la posición del slider. 6. El sistema cambia la opacidad de la capa.

Nombre	Ver Mapa
Autores	Alba Muñiz Revuelta
Versión	1.0
Prioridad	Media

Criticidad	Baja
Fuente	*Predictia*
Responsable	Alba Muñiz Revuelta
Actor Principal	Usuario
Descripción	Mostrar datos meteorológicos en un mapa
Objetivo	Visualizar las capas deseadas en un mapa
Evento de Activación	Ver Mapa
Precondición	Tener acceso al servicio web
Garantías Si Éxito	Se visualiza el mapa
Garantías Mínimas	Si el proceso falla no se modifica nada
Escenario Principal	1. El usuario accede a la aplicación web desde su navegador. 2. La aplicación cliente pide al servidor WMS sus datos disponibles (a través de la operación getCapabilities de la interfaz WMS). 3. La aplicación cliente crea el mapa, el menú de tiempos y el menú de capas según lo establecido en el fichero de configuración. 4. Por cada capa seleccionada para visualización: 4.1 La aplicación cliente hace una petición al servidor WMS para obtener la capa correspondiente. Los datos necesarios para realizar esta petición se extraen de la información obtenida en el paso 2. 4.2 La aplicación cliente muestra la capa retornada en el mapa.

3.4. ESPECIFICACIÓN DE ITERACIONES

Una vez que ya hemos especificado los requisitos identificamos cuál de ellos se han ido desarrollando en cada iteración.

Iteración	Descripción	Requisitos
1	Mostrar un mapa en el cual se pueda hacer zoom, además de poder moverse por el mapa.	RFC001, RFC001.3, RFC001.4
2	Se le añade el selector de tipos de mapa y capas.	RFC001.1, RFC001.2
3	Se añade el controlador de tiempos y de fechas en las que se realiza la predicción.	RFC001.5, RFC001.6, RFC001.7, RFC001.9, RFC001.10
4	Se añade el cambio de opacidad de la capa al selector de capas.	RFC001.8
5	Se añade la posibilidad de poder adelantar una de las capas seleccionadas para poder verla encima del resto de las capas.	RFC001.12
6	Se crea un fichero para la configuración de	RFC002, RFC002.1 –

	la interfaz del mapa.	RFC002.17
7	Se crea un fichero para la configuración de la interfaz del meteograma.	RFC004
8	Se desarrolla la visualización del meteograma.	RFC003
9	Se crea una clase para procesar los datos y que se cree automáticamente un fichero JSON.	RFS001, RFS002, RFS004
10	A partir de la iteración anterior se desarrolla una clase que procesa el fichero JSON y crea una capa animada.	RFC001.11
11	Se desarrolla una nueva funcionalidad que crea capas a partir de datos.	RFS001, RFS002, RFS003

4. ARQUITECTURA

Es este apartado se tratan las diferentes partes de la arquitectura del sistema.

4.1. MVC

Tanto en el lado servidor como cliente se va a seguir una arquitectura *Modelo-Vista-Controlador* (MVC) [21], para así separar el código en función de sus responsabilidades. En la Ilustración 8 se muestra la estructura genérica del patrón MVC.

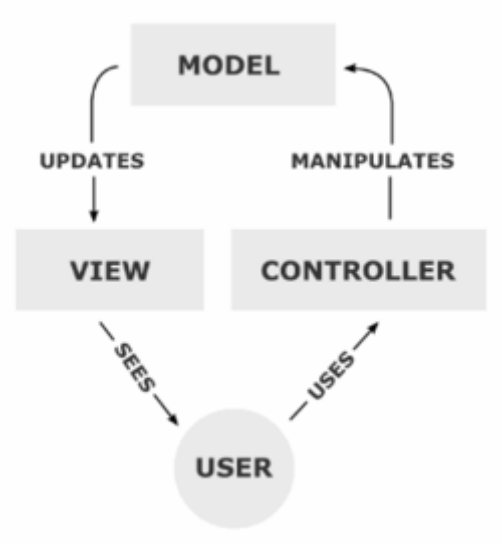

Ilustración 8 Diagrama Patrón MVC[3]

MVC es un patrón de arquitectura software que separa el código en tres capas diferentes, diferenciando los datos y la lógica de negocio de una aplicación, de la interfaz de usuario y la gestión de eventos y comunicaciones. Se utiliza este patrón por la necesidad de un software robusto, donde se potencie el mantenimiento, la reutilización del código y la separación de conceptos.

Consta de las siguientes capas:

- Modelo: módulo que trabaja con los datos, por lo tanto contiene mecanismos para acceder a ellos.
- Vista: módulo que produce la visualización de las interfaces de usuario.
- Controlador: módulo que hace de enlace entre los otros dos. Su función es responder a las acciones que se solicitan en la aplicación.

Aunque en ocasiones puede haber pequeñas variaciones de implementación, el flujo de trabajo normal cuando se aplica este patrón es el siguiente:

[3] https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

- El usuario realiza una solicitud a nuestro sistema. Dicha solicitud llega al controlador.
- El controlador se comunica con modelos y vistas, solicitando los datos necesarios a los modelos y proporcionando la salida pertinente a las vistas, además de realizar las operaciones necesarias según la lógica de negocio.
- En ocasiones las vistas necesitan más información y a través del controlador se solicitan todos los datos que necesitan las vistas a los modelos.
- Las vistas envían la salida al usuario.

4.2. VISTA DE CONECTORES Y COMPONENTES

Tanto para el cliente como para el servidor se ha implementado la arquitectura MVC utilizando *Spring Framework*.

4.2.1. CLIENTE

En la Ilustración 9 se expone el modelo de arquitectura de la aplicación cliente.

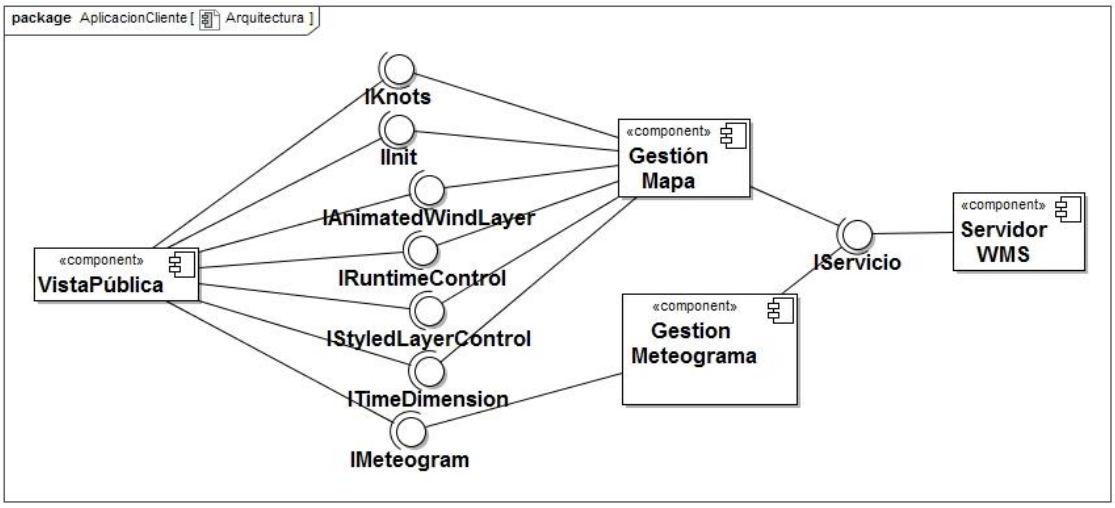

Ilustración 9 Arquitectura de la aplicación cliente

A través de la vista se interactúa con el cliente, que a su vez pide los datos necesarios al servidor. En este caso con el servidor solo se va a trabajar para obtener datos, en ningún momento desde la vista se van a poder modificar los datos.

En este caso el componente *VistaPublica* hace la función de Vista, estando implementado a través de un conjunto de ficheros HTML. Los componentes *GestionMapa* y *GestionMeteograma* realizan las funciones de Controlador, implementando en JavaScript las interfaces que aparecen en la Ilustración 10. Mientras que el *ServidorWMS* proporciona los datos necesarios por lo que realiza la función de Modelo.

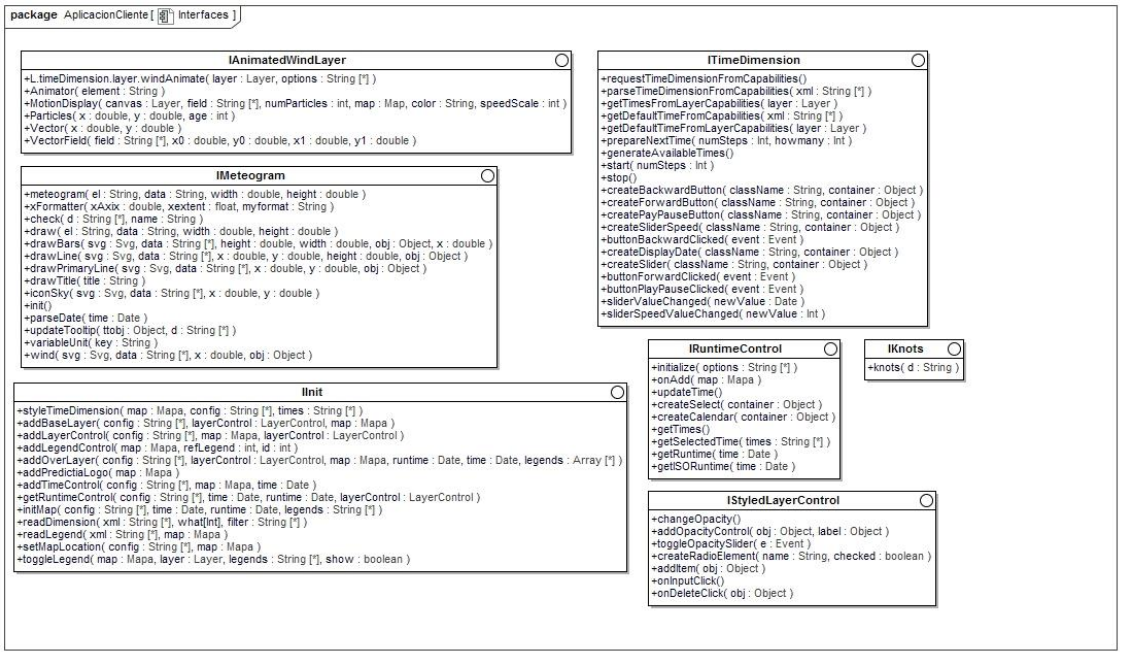

Ilustración 10 Interfaces internas del cliente

La interfaz *IAnimatedWindLayer* es la que implementa toda la lógica para la creación de la capa de viento animada. La interfaz *IInit* es la que implementa la lógica de la configuración de la interfaz del mapa en base a lo establecido en el fichero de configuración. *ITimeDimension* contiene la lógica para la obtención de los datos de las fechas de predicción y las fechas de realización de las predicciones. *IMeteogram* contiene la lógica para la interfaz del meteograma. *IStyledLayerControl* contiene la lógica del control de mapas y capas. *IRuntimeControl* contiene la lógica del control de tiempos. La interfaz *IKnots* simplemente define un método que convierte los m/s en nudos.

4.2.2. SERVIDOR

En la Ilustración 11 se expone el modelo de arquitectura del servidor.

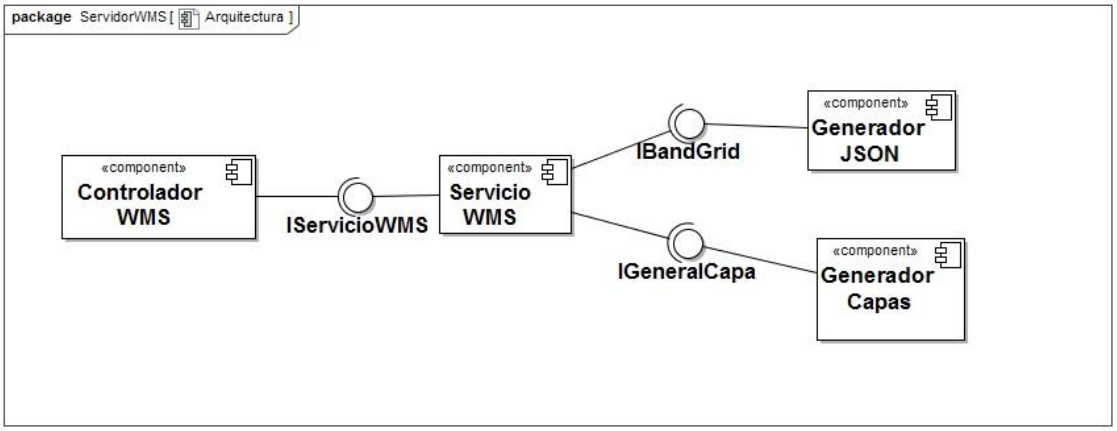

Ilustración 11 Arquitectura del servidor

En esta parte el rol de Vista no existe, ya que serán los posibles clientes quienes lo implementen. Los componentes *ControladorWMS* y *ServicioWMS* ya se encontraban

implementados, habiendo sido necesario extenderlos para dar soporte a la nueva funcionalidad. El componente *ControladorWMS* es quién realiza las funciones de Controlador, mientras que el componente *ServicioWMS* implementa la interfaz *IServicioWMS* que contiene los métodos a través de los cuales se hacen las peticiones desde el Cliente, por lo que hace las funciones de Modelo. El componente *GeneradorJSON* implementa la interfaz *IBandGrid*, la cual implementa la lógica que procesa los datos y crea el fichero JSON con los datos del viento, mientras que el componente *GeneradorCapas* implementa la interfaz *IGeneralCapa*, que implementa la lógica que crea diferentes capas dependiendo de los parámetros que se le pasen en la llamada. Estos dos últimos componentes forman parte del trabajo realizado en este proyecto. En la Ilustración 12 se muestran en detalle sus interfaces.

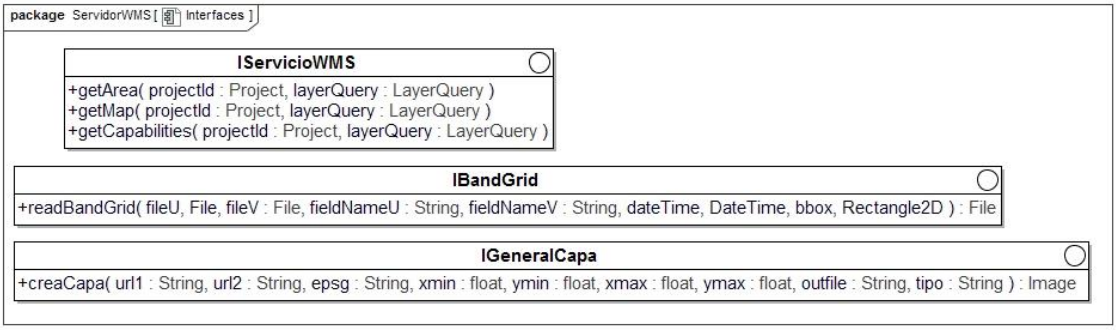

Ilustración 12 Interfaces servidor

La interfaz *IServicioWMS* implementa el estándar WMS. De entre todas las operaciones del estándar, sólo se muestran en la figura las dos que se requieren para implementar el cliente web:

- *getMap* es el método invocado por el cliente para obtener capas de visualización. Este método utilizará el método *creaCapa* del componente *GeneradorCapas* para obtener las nuevas capas de viento y de altas y bajas presiones.
- getCapabilities es el método invocado por el cliente para obtener la información disponible en el servidor para mostrar (capas, dimensiones, estilos, proyecciones geográficas, etc.).

Además, se ha añadido un método adicional a la interfaz, el método getArea, que será invocado por el cliente para obtener los datos del viento en formato JSON. Este método a su vez hace una llamada al método *readBandGrid* del componente *GeneradorJSON* para obtener dicha información.

Para una mejor comprensión de la función y relación entre estos métodos, se muestra a través de un diagrama de secuencia (Ilustración 13), el escenario arquitectónico correspondiente a la visualización de las capas seleccionadas por un usuario en la aplicación cliente.

El usuario accede al cliente WMS a través del navegador web. El cliente pide los datos disponibles al servidor WMS a través del método *getCapabilities*. A partir de esa información y de la configuración de capas seleccionadas por el usuario, el cliente construye el árbol de capas. En el caso de que haya capas seleccionadas en la

inicialización o cuando el usuario seleccione una capa, al ser una capa WMS, el cliente sabe cómo tiene que pedir las imágenes correspondientes (siguiendo el estándar WMS) al servidor WMS gracias a la información obtenida anteriormente. Básicamente, para cada imagen que se quiere representar el cliente le pasa al servidor el área de la capa que desea visualizar (*bounding box*), los tiempos (fecha/hora de la predicción y fecha en la que se realizaron las predicciones), el estilo, elevación, etc. Hay dos formas básicas de definir una capa WMS. La primera, partiendo el espacio del mapa en trozos, por lo que se hará una petición por cada uno de los trozos. La segunda pide una sola imagen para todo el mapa, por lo que se hace una sola petición.

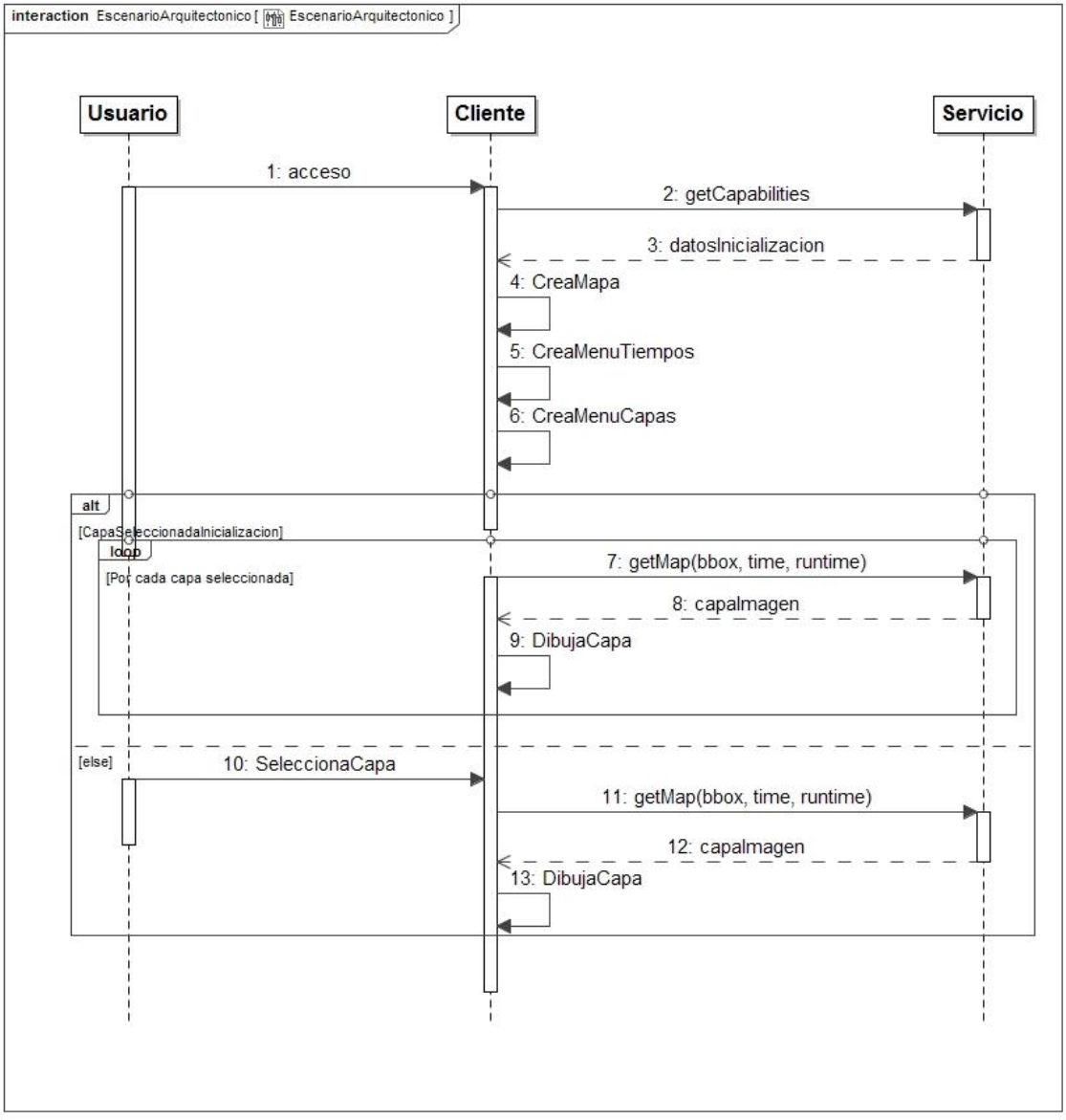

Ilustración 13 Escenario arquitectónico de visualización de capas

4.3. VISTA DE DESPLIEGUE

En la Ilustración 14 se muestra la vista de despliegue. El cliente que se desarrolla durante este proyecto se despliega empaquetado como un archivo WAR (*cliente.war*) en un servidor web Apache al que accede el usuario a través de un navegador. Este WAR

hace las peticiones de datos a otro servidor de la empresa, que contiene un contenedor de Servlets Apache Tomcat donde el servicio está desplegado también como archivo WAR (*servicioWMS.war)*. Este archivo ha sido actualizado para contener el generador de ficheros JSON que crea la capa de viento animada y el generador de capas implementado en Python para la creación de capas tanto de viento como de altas y bajas presiones. Este contenedor de Servlets accede a su vez a una base de datos PostGree, dónde se almacena los datos que permiten acceder al sistema de ficheros que almacena los ficheros NetCDF requeridos. Además tenemos un Renderizador que es el componente que crea las capas utilizando Mapnik y Python.

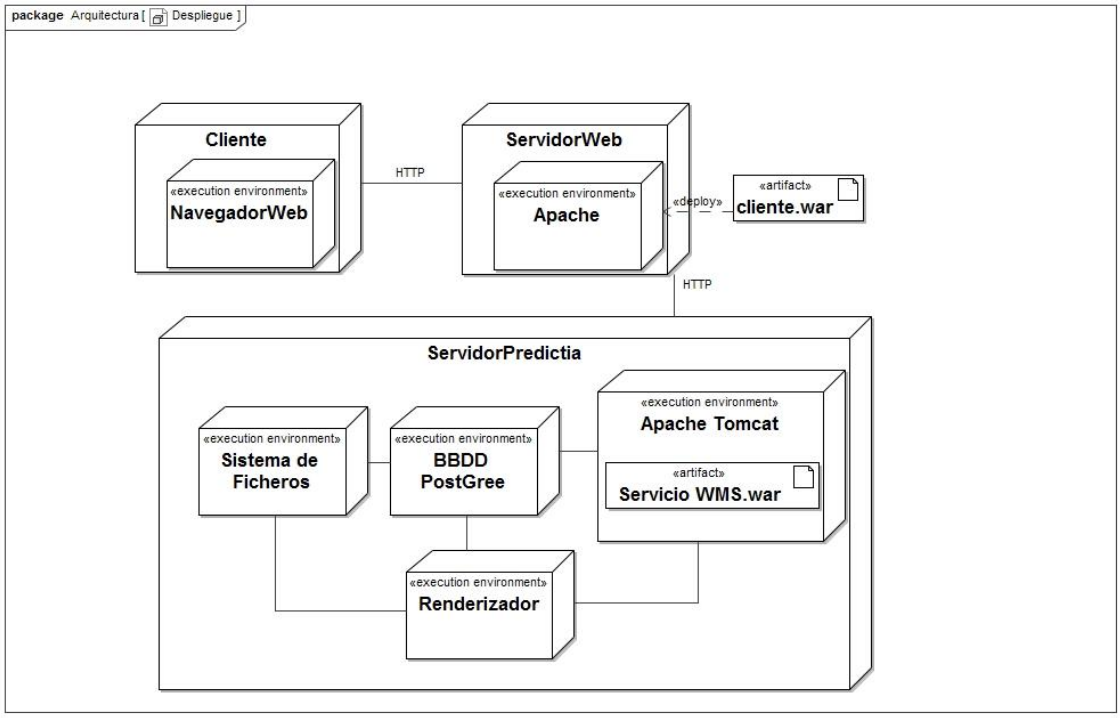

Ilustración 14 Vista de Despliegue

5. IMPLEMENTACIÓN

En este apartado se hace hincapié en el diseño que se ha seguido y principalmente en algunos detalles de la implementación que se ha realizado.

Como ya se ha mencionado anteriormente se ha seguido una metodología iterativa y una arquitectura MVC. Una vez que se tienen claros estos puntos, así como los requisitos se procede a la implementación de cada uno de los componentes.

5.1. CLIENTE

En este apartado nos centramos en la implementación del cliente. La aplicación cliente está basada en HTML para la parte de las interfaces y JavaScript para la implementación de las funcionalidades. En la Ilustración 15 se muestra una captura de la organización del proyecto, con todos los ficheros .html y .js involucrados.

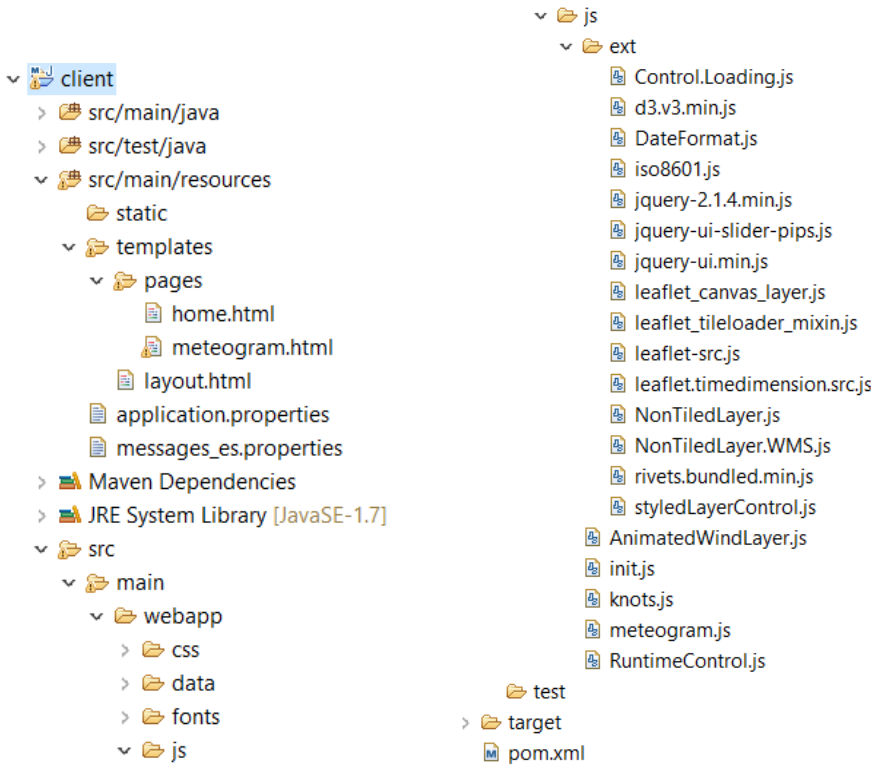

Ilustración 15 Organización de la Aplicación Cliente

La Ilustración 16 muestra la maquetación de la interfaz de usuario que tendrá la aplicación cliente desarrollada.

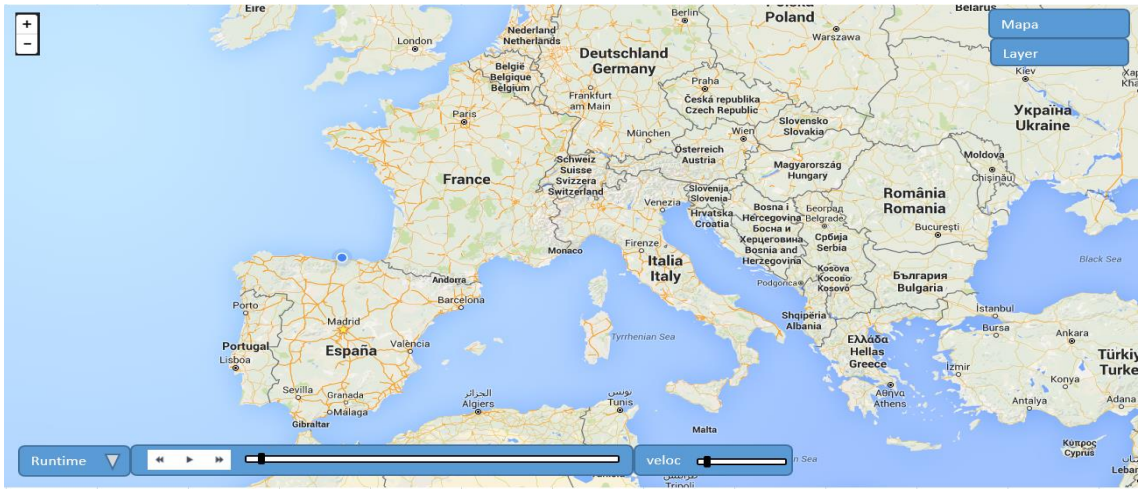

Ilustración 16 Maquetación de la interfaz de usuario

El HTML para la interfaz del mapa simplemente tiene los enlaces con todos los .js y como se muestra en la siguiente imagen, una sección con id=map en el body. Es el fichero JavaScript inicial el que se encarga de crear el mapa en la sección indicada.

```
<body>
	<div layout:fragment="content">
		<div id="map"></div>
	</div>
</body>
```

Fragmento de código 1: Sección creada para el mapa

Por su parte, en la Ilustración 17 podemos ver un fragmento de la interfaz del meteograma.

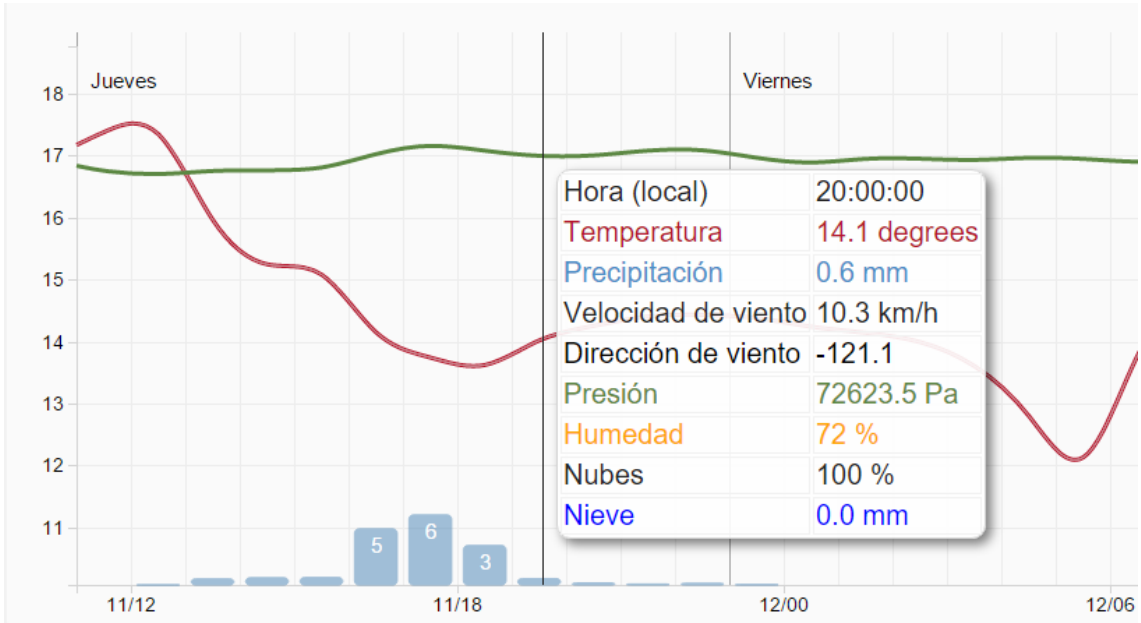

Ilustración 17 Parte del Meteograma

El desarrollo de la interfaz del meteograma tiene un poco más de complejidad, ya que para hacer el panel dinámico con el que se muestran los datos es necesario añadirlos al HTML. A continuación se muestra un fragmento del código requerido.

```
<div id="tooltip">
	<table class="table table-condensed">
		<tr>
			<th>Hora (local)</th>
			<th>{d.datetime}</th>
		</tr>
		<tr class="avg"><td>Temperatura </td><td>{d.temp}</td></tr>
		<tr class="rain"><td>Precipitaci&oacute;n </td><td>{d.rain}</td></tr>
		<tr class="wind"><td>Velocidad del viento
		</td><td>{d.wind_speed}</td></tr>
		<tr class="wind"><td>Direcci&oacute;n del viento
		</td><td>{d.wind_direction}</td></tr>
		<tr
		class="pressure"><td>Presi&oacute;n</td><td>{d.pressure}</td></tr>
		<tr class="hum"><td>Humedad</td><td>{d.humidity}</td></tr>
		<tr class="cloud"><td>Nubes</td><td>{d.cloud} %</td></tr>
		<tr class="snow"><td>Nieve</td><td>{d.snow}</td></tr>
		<tr class="sky"><td>Estado Cielo</td><td><img rv-
		src="d.iconPath"/></td></tr>
	</table>
</div>
```

Fragmento de código 2: Sección del HTML del meteograma

Por otro lado, se ha desarrollado el fichero de configuración para la interfaz del mapa. Con este fichero lo que queremos conseguir es poder configurar casi en su totalidad el aspecto que ofrecerá la interfaz sin necesidad de modificar código. A través del fichero, el usuario indica donde quiere colocar cada uno de los controladores de la interfaz, la dirección del WMS de donde se van a obtener los datos, los estilos de las capas, qué estilos de mapa se van a poder seleccionar o qué capas vamos a dar la opción de mostrar, los estilos de selectores o si queremos ver o no los controles y/o las leyendas.

```
var config = {
	project:{
		wms:'http://datlas.predictia.es/wms-predictia'},
	basemap:{
		name:'Maps',
		layers:[{
			name:'toner-background',
			title:'Toner',
			selected:false,
			maxZoom:18}..},
	overlayer:[{
		name:'Layers',
		layers:[ {
			name:'Clouds',
			type:'wms', // wms, canvas, image
			visible:false,
			styles:'RASTER',
			legend:true,
			layerOptions:{
				layers:'predictia.EuropaCentral.Clouds',
```

Fragmento de código 3: Fichero de configuración de la interfaz del mapa

En el Fragmento de código 3 se muestra como se ha implementado este fichero de configuración que es quien da la información a la aplicación para la configuración inicial. Aquí podemos ver cómo se asigna la dirección del servidor WMS de dónde se obtendrán los datos, y cómo se indica que la interfaz deberá tener menú de mapas y un submenú de capas con una capa para las nubes.

Para la parte de visualización del mapa relacionada con poder aplicar zoom y poder desplazarse por todo el mapa se utiliza una librería de Leaflet. A modo de ejemplo, el Fragmento de código 4 muestra la creación de un mapa usando dicha librería.

```
var map = L.map('map',{
      timeDimension:true,
      timeDimensionOptions:timeDimensionOptions,
            loadingControl:true
      });
```

Fragmento de código 4: Creación mapa usando Leaflet

Se continúa con la implementación del control de capas y estilos de mapas. Para lo cual también se utiliza Leaflet, ya que facilita enormemente el trabajo.

Se comprueba el fichero de configuración de interfaz y se van añadiendo los diferentes estilos de mapa seleccionados a una capa base (`baseLayer`) y las diferentes capas a un submenú (`Overlay`) con el nombre de su grupo, que en este caso es siempre el mismo y sería “Layer”.

En el Fragmento de código 5 se muestra cómo se van creando las diferentes capas de estilo de mapa y en el Fragmento de código 6 como se añaden al basemap.

```
}else if(config.basemap.layers[i].name == 'OpenStreetMap'){
      layerMap = L.tileLayer('http://{s}.tile.openstreetmap.org/
      {z}/{x}/{y}.png');
}else if(config.basemap.layers[i].name == 'OpenTopoMap'){
      layerMap = L.tileLayer('http://{s}.tile.opentopomap.org/
      {z}/{x}/{y}.png');
}else if(config.basemap.layers[i].name ==
      'Thunderforest.OpenCycleMap'){
      layerMap=L.tileLayer('http://{s}.tile.thunderforest.com/cycle/
      {z}/{x}/{y}.png');
```

Fragmento de código 5: Creación de capas según estilo de mapa

```
if(layerMap != null && config.layerControl.baselayer){
      layerControl.addBaseLayer(layerMap,
      config.basemap.layers[i].title,{groupName:nameGroup});
      if(config.basemap.layers[i].selected){
            map.addLayer(layerMap);
            layersAdded++;
```

Fragmento de código 6: Añadir capas a la capa base

En el Fragmento de código 7 se muestra como se crean las capas y se añaden al submenú (`Overlay`).

```
if(type=='wms' || type=='nt-wms'){
	var layer;
	if(type=='wms'){
		layer = L.tileLayer.wms(url,layerOptions);
	}else if(type=='nt-wms'){
		layer = L.nonTiledLayer.wms(url,layerOptions);
	}
	var layWMS = L.timeDimension.layer.wms(layer,options);
	if(layWMS != null){
		layerControl.addOverlay(layWMS,name,
```

Fragmento de código 7: Creación de capas de predicción

El resultado visual sería el siguiente (Ilustración 18):

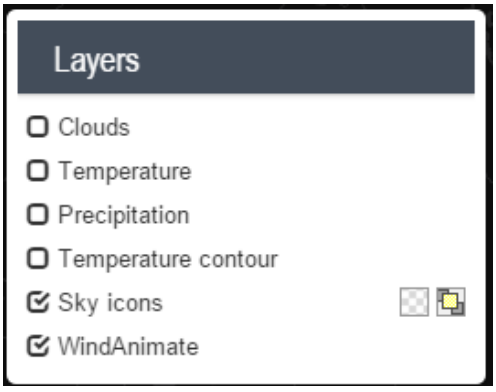

Ilustración 18 Control de Capas

Para la implementación del control de los tiempos de la predicción se utiliza una clase ya implementada y publicada en GitHub, denominada `TimeDimension`[4] a la cual se han realizado algunas modificaciones. Utilizando esta clase y a través del método *getCapabilities* del servidor se obtiene el fichero WMS de donde se extraen los datos tanto de las fechas en las que se hacen las predicciones (Runtime), como de las fechas para las que existe predicción.

Cuando se tienen estos datos se añade el controlador de tiempos a la interfaz con un selector para las fechas de realización de las predicciones. Este controlador puede tener dos formatos, lista desplegable o calendario, y un display de animación para los tiempos de predicción. En este caso podemos ir pinchando en el display para seleccionar la fecha/hora de la que queremos ver la predicción o animar el display automáticamente. También se añade un selector de velocidad para la animación automática.

En el Fragmento de código 8 se muestra la petición de los datos al WMS. Lo primero que se hace es crear la URL a la que se debe hacer la petición, indicando que método se

[4] https://github.com/socib/Leaflet.TimeDimension

quiere invocar, en este caso *getCapabilities*, y posteriormente se extrae la información

```
_requestTimeDimensionFromCapabilities: function() {
       if (this._capabilitiesRequested) {
              return;}
       this._capabilitiesRequested = true;
       var wms = this._baseLayer.getURL();
       var url = wms + "?service=WMS&version=" +
         this._wmsVersion + "&request=GetCapabilities";
       if (this._proxy) {
              url = this._proxy + '?url=' + encodeURIComponent(url);}
       $.get(url, (function(data) {
              this._defaultTime =
              Date.parse(this._getDefaultTimeFromCapabilities(data));
              this._setDefaultTime = this._setDefaultTime ||
              (this._timeDimension &&
              this._timeDimension.getAvailableTimes().length == 0);

              this.setAvailableTimes(this._parseTimeDimensionFromCapabilitie
              s(data));
              if (this._setDefaultTime && this._timeDimension) {
                     this._timeDimension.setCurrentTime(this._defaultTime);
              }
      }).bind(this));},
```

requerida del valor de retorno.

Fragmento de código 8: Método para obtener los tiempos

La visualización se muestra en la Ilustración 19:

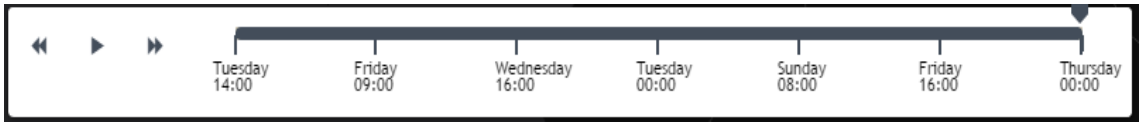

Ilustración 19 Control de Tiempos

En este caso no aparece ni el selector de fecha/hora en la que se realizó la predicción, ni el slider de la velocidad ya que en el fichero de configuración de la interfaz, como se puede observar en el Fragmento de código 9, están configuradas como NONE para que no aparezcan.

```
runtimes:{
       control:'none',// 'none', 'simple', 'calendar'
       //first-x, last-x, como unico elemento de un array
       values: [], // times, first-100, last-100
       defaultValue:[],  // value, 'min', 'max'
       scope:'PT72H',
       format:'dd/mm/yyyy HH:MM'
},
animation:true, // true/false
fps:false, // true/false
```

Fragmento de código 9: Fichero de configuración control de tiempos

Se añade el slider de opacidad para las capas seleccionadas. En el Fragmento de código 10 se muestra como se añade esto al código HTML.

```
container.innerHTML = '<span class="opacity">'+' Opacity:
</span><div class="slider"></div>';
      var slide = $(container).find('.slider');
      slide.slider({
            min:0,
            max:1,
            range:'min',
            step:0.1,
            value:opacity,
```

Fragmento de código 10: Contenedor para el control de opacidad

Para el estilo del controlador de las capas se ha utilizado una clase ya implementada y publicada en GitHub, denominada `styledLayerControl`[5], en la que se han realizado las modificaciones que se han necesitado. En este caso se añade la funcionalidad del slider para cambiar la opacidad de la capa.

En la Ilustración 20 se muestra como se ve este slider y el botón de opacidad en la interfaz.

Ilustración 20 Slider de Opacidad

De esta forma si la capa está seleccionada se muestra el recuadro de la opacidad y si se pincha se muestra el slider debajo del nombre de la capa. Esta funcionalidad es muy útil, ya que si queremos ver varias capas y son opacas, no podemos ver las capas que hay debajo. De esta forma si modificamos la opacidad de la de más arriba podemos ver las capas de debajo.

De la misma manera se añade la capacidad de adelantar una capa, añadiendo un recuadro en el control de manera que si haces click sobre él la capa se adelante.

Ha sido necesario implementar también un fichero de configuración para el meteograma, que sigue el mismo esquema que el fichero de configuración de la interfaz del mapa. Para la implementación del meteograma se usa D3.js, que es una herramienta muy útil para implementar gráficos dinámicos. A continuación podemos ver el Fragmento de código 11 donde se muestra como se dibuja una línea con D3.js.

```
svg.append('path')
      .datum(data)
      .attr('class', 'line avg')
      .attr('d', medianLine)
      .attr('clip-path', 'url(#rect-clip)')
      .style('stroke', colorMean);
```

Fragmento de código 11: Dibujar una línea con D3js

[5] https://github.com/davicustodio/Leaflet.StyledLayerControl

A continuación se expone algún detalle sobre la creación de la capa de viento animada. Con los datos del fichero JSON obtenido del servidor se crea la capa y se van pintando las partículas iterativamente para simular el movimiento de cada una de ellas. En el siguiente cuadro de texto aparece el algoritmo implementado.

```
for (var i = 0; i < this.particles.length; i++){
        var p = this.particles[i];
        if (!this.field.inBounds(p.x, p.y)){
                p.age = -2;
                continue;
        }
        var proj = this.map.latLngToContainerPoint(new L.LatLng(p.y,p.x));
        proj.x = proj.x * scale + dx;
        proj.y = proj.y * scale + dy;
        if (proj.x < 0 || proj.y < 0 || proj.x > w || proj.y > h){
                p.age = -2;
        }
        if (p.oldX != -1){
                var wind = this.field.getValue(p.x, p.y, val);
                var s = wind.length() / this.maxLength;
                var c = 90 + Math.round(350 * s); // was 400
                if (c > 255){
                        c = 255;
                }
                //Particles
                g.strokeStyle = this.colors;
                g.beginPath();
                g.moveTo(proj.x, proj.y);
                g.lineTo(p.oldX, p.oldY);
                g.stroke();
        }
        p.oldX = proj.x;
        p.oldY = proj.y;
}
```

Fragmento de código 12: Algoritmo para creación de partículas en movimiento

De esta manera se van creando capas no opacas que al colocarlas encima de las antiguas van ensombreciendo los trozos de las partículas viejas cada vez más y mostrando por encima la partícula nueva, de manera que parece que dicha partícula está en movimiento. En la Ilustración 21 se muestra una captura de la capa animada de viento.

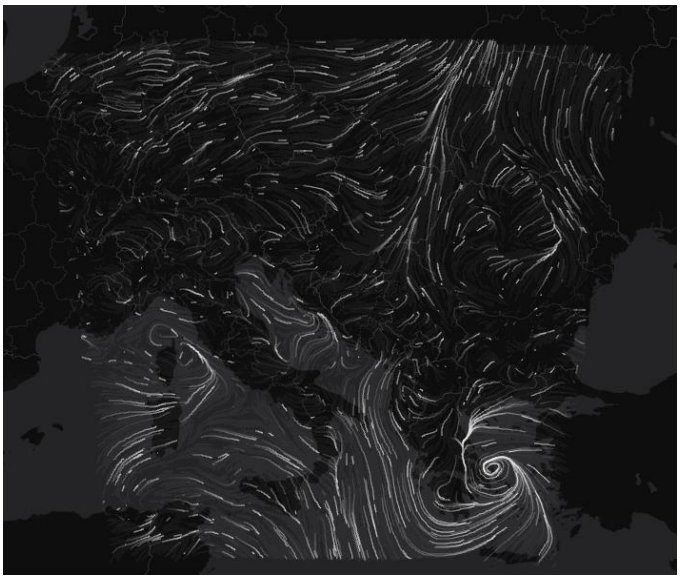

Ilustración 21 Capa Viento Animado

5.2. SERVIDOR

En la parte del servidor lo que se ha implementado es una clase Java denominada *BandGridReader* que implementa el método *readBandGrid,* que como se explicó en el apartado de arquitectura, a su vez es llamado por el método *getArea* que implementa el componente *ServicioWMS* con objeto de obtener los datos del viento en formato JSON.

En la Ilustración 22 se muestra el diagrama de clases Java con el diseño detallado de esta clase.

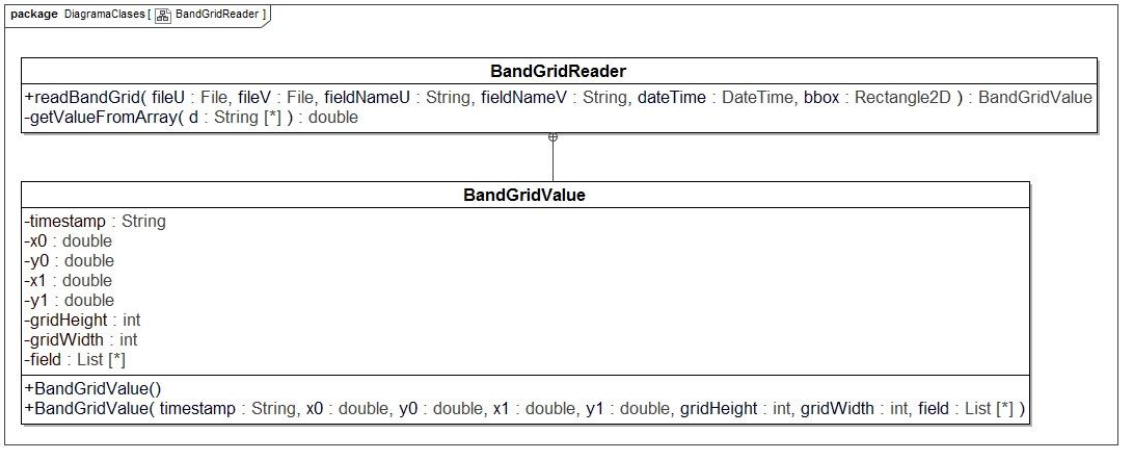

Ilustración 22 Diseño de la clase BandGridReader

La clase BandGridReader obtiene la información necesaria a partir de manipulación de ficheros NetCDF. Estos ficheros NetCDF tienen una organización interna que permite acceder a los datos que pertenecen a una variable determinada. En este caso, solo se necesitarán los datos del viento, por lo que se obtienen los datos de las variables *u* (componente zonal) y *v* (componente meridional). También se requerirán los datos de las presiones por lo que en este caso se accederá a la variable *Pressure_reduced_to_MSL_msl*.

Lo que hace el método *readBandGrid* es acceder a los ficheros, en este caso de las variables *u* (componente zonal) y *v* (componente meridional) del viento, y procesar los datos. Los ficheros iniciales contienen los datos de toda la extensión del mundo y en este caso solo se requiere un espacio más reducido. Además, en estos ficheros los datos se guardan en un formato que no es útil, ya que los valores de las variables pertenecen a una latitud/longitud y es mucho más interesante tenerlos en forma matricial.

Cómo los datos utilizados están almacenados en el formato NetCDF, se utiliza la librería *netcdf-java* para su lectura.

Lo que hace el método es lo siguiente, se comprueba cada una de las parejas latitud/longitud donde hay datos y si esta pareja está dentro del fragmento del mapa que interesa. Si es así, se va llenando una matriz de la siguiente forma: si hay datos para cada una de las posiciones se introducen los datos y en caso contrario se añade un nulo. Esta matriz es transformada automáticamente a un fichero JSON, que es el fichero que se proporcionará al cliente para hacer la animación del viento.

El fichero JSON generado tiene el formato que podemos observar en la Ilustración 23.


```
{"timestamp":"2011-02-01T16:00:00.000Z","x0":-31.0,"y0":31.0,"x1":24.5,"y1":61.0,"gridHeight":61,
"gridWidth":112,"field":[5.63,-0.48,3.55,-0.06,0.87,0.86,-1.5,2.05,-4.39,3.43,-8.19,3.86,-11.2,4.38,-13.12,
5.59,-15.33,5.17,-18.16,4.86,-20.05,6.78,-20.63,8.34,-10.97,12.77,3.4,16.82,8.44,16.95,9.4,16.02,10.02,15.69,
11.22,15.46,12.96,15.27,14.42,14.53,15.71,13.44,16.7,12.09,16.85,10.38,16.34,8.88,15.35,7.9,14.02,7.88,12.57,
8.9,11.3,10.09,10.36,10.72,9.64,10.85,8.8,10.59,7.94,10.06,7.4,9.73,7.11,9.98,6.84,10.64,6.76,11.19,7.07,11.28,
7.11,11.3,6.01,11.1,5.34,10.77,5.03,10.21,4.75,9.46,4.51,8.89,4.15,8.32,3.79,7.98,3.41,7.88,2.89,7.67,2.27,7.15,
1.64,6.42,1.04,5.78,0.61,5.63,0.17,5.76,-0.57,5.57,-1.36,5.01,-1.92,4.51,-2.39,4.11,-2.89,3.64,-3.43,3.1,-3.98,
2.58,-4.56,2.25,-5.25,1.9,6.25,-0.57,4.28,0.05,1.69,1.09,-0.75,2.45,-3.81,3.88,-7.79,4.34,-11.03,4.94,-13.07,6.28,
-15.12,6.11,-17.66,5.84,-19.38,7.57,-20.0,8.92,-11.49,12.64,1.7,16.54,6.98,17.02,8.31,15.69,9.06,15.06,10.14,14.3,
```


Ilustración 23 Muestra del fichero JSON

En el Fragmento de código13 se muestra el algoritmo seguido para realizar las correspondientes comprobaciones y transformaciones.

```
for(int x=xMin; x<=xMax; x++){
      for(int y=yMin; y<=yMax; y++){
            Double lon = x0 + (double)x*(x1-x0)/
                  (double)((xMax-xMin)+1);
            Double lat = y0 + (double)y*(y1-y0)/
                  (double)((yMax-yMin)+1);
            int[] xy = xyAxisU.findXYindexFromLatLon(lat, lon,
                  null);

            if(xy[0]!=-1 && xy[1]!=-1){
                  Double dataU =
                  getValueFromArray(gridU.readDataSlice(tIndex,
                        zIndex,xy[1],xy[0]));
                  Double dataV =
                  getValueFromArray(gridV.readDataSlice(tIndex,
                        zIndex,xy[1],xy[0]));
                  if(!Double.isNaN(dataU) &&
                     !Double.isNaN(dataV)){
                        BigDecimal bdU = new
                        BigDecimal(dataU).setScale(2,
                        RoundingMode.FLOOR);
                        BigDecimal bdV = new
                        BigDecimal(dataV).setScale(2,
                        RoundingMode.FLOOR);
                        field.add(bdU.doubleValue());
                              field.add(bdV.doubleValue());
                  }else{
                        field.add(Double.NaN);
                        field.add(Double.NaN);
                  }
            }else{
                  field.add(Double.NaN);
                  field.add(Double.NaN);
            }
      }
```

Fragmento de código 13: Algoritmo para creación de comprobación

Por otro lado, se ha implementado utilizando Python y las librerías *Basemap* y *Matplotlib*, una funcionalidad nueva para crear capas, en este caso de viento o de altas y bajas presiones, lo más generalizada posible, de manera que dependiendo de los parámetros que pasemos nos crea una tipo de capa u otra. Por el momento se ha

implementado para crear capas de viento tanto con iconos *quiver* (flechas) como con iconos *barbs* (las típicas barbas de representación del viento), y para la creación de una capa de altas y bajas presiones. En la Ilustración 24 se muestra el diseño de esta clase.

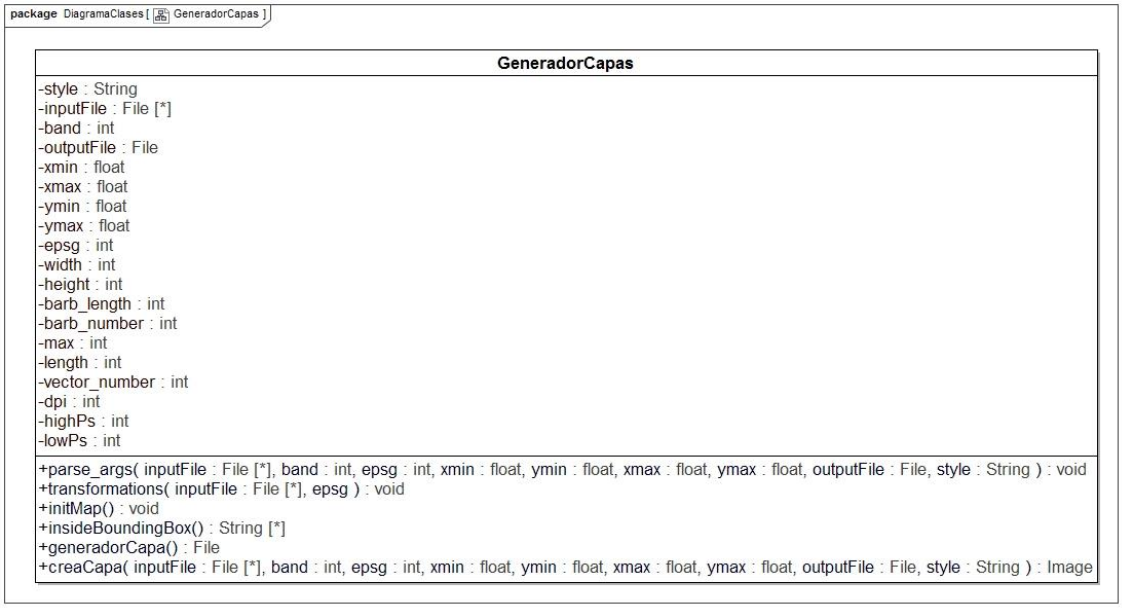

Ilustración 24 Diseño de la clase GeneradorCapas

Para ello el primer paso es pasar los parámetros a variables. Una vez que tenemos esto, con los puntos XYMin y XYMax se crea el contenedor de la capa (donde se van a dibujar los iconos), inicialmente este contenedor se crea en proyección cartográfica EPSG por lo que se transforma a datos de coordenadas. Después se crea la capa sin iconos, se comprueba que los datos están dentro del contenedor de la capa y se crea un espacio de separación entre iconos de forma que un icono no se superponga a otro. Este proceso es prácticamente común para todos los tipos de capa.

Es a la hora de dibujar los iconos cuando el código difiere dependiendo del estilo de capa que se quiera dibujar. Para dibujar los iconos quiver y barbs simplemente se utiliza la función de *quiver* y *barbs* respectivamente que proporciona la librería Basemap. Mientras que en el caso de las presiones, se comprueba coordenada a coordenada si la presión en dicha coordenada es superior al valor de alta presión pasado como parámetro, en cuyo caso se dibuja una H roja en la coordenada y se pinta debajo el valor numérico de la presión, o en el caso de que sea inferior al valor de baja presión pasado como parámetro, se dibuja una L azul en la coordenada y se pinta debajo el valor numérico de la presión. En la Ilustración 25 se muestra como en el caso de quiver y barbs se llama a la función ya implementada por Basemap y en el caso de las altas y bajas presiones se hacen las comprobaciones y se dibuja.

```
    if style == 'quiver':
        m.ax.quiver(dataXp,dataYp,maxLength*dataU/maxValue,maxLength*dataV/maxValue,color='k',pivot='mid',width=maxLength,minlength

    if style == 'barbs':
        m.ax.barbs(dataXp,dataYp,dataU,dataV,length=barb_length,barbcolor='k',linewidth=1,pivot='middle')

if style == 'pressure':
    yoffset = 0.03
            print(yoffset)
    i=-1
    for x in dataP:
        j=0
                i=i+1
        for y in range(len(x)):
        if x[y] < lowPs:
            posX = dataXp[i,j]
            posY = dataYp[i,j]
            plt.text(posX, posY,'L',fontsize=14,fontweight='bold',
                ha='center',va='center',color='b')
            plt.text(posX, posY-yoffset,x[y],fontsize=9,
                ha='center',va='top',color='b',
              bbox = dict(boxstyle="square",ec='None',fc=(1,1,1,0.5)))
```

Fragmento código 14: Llamada al método dependiendo del tipo

El resultado final sería el siguiente:

Para *Quiver*:

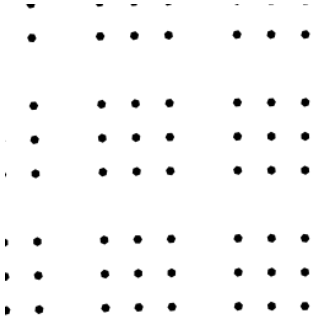

Ilustración 25 Capa de viento tipo Quiver

Para *Barbs*:

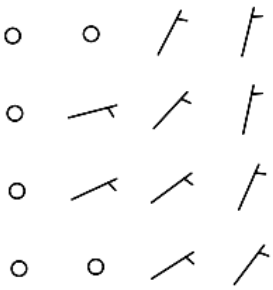

Ilustración 26 Capa de viento tipo Barbs

Para las Presiones:

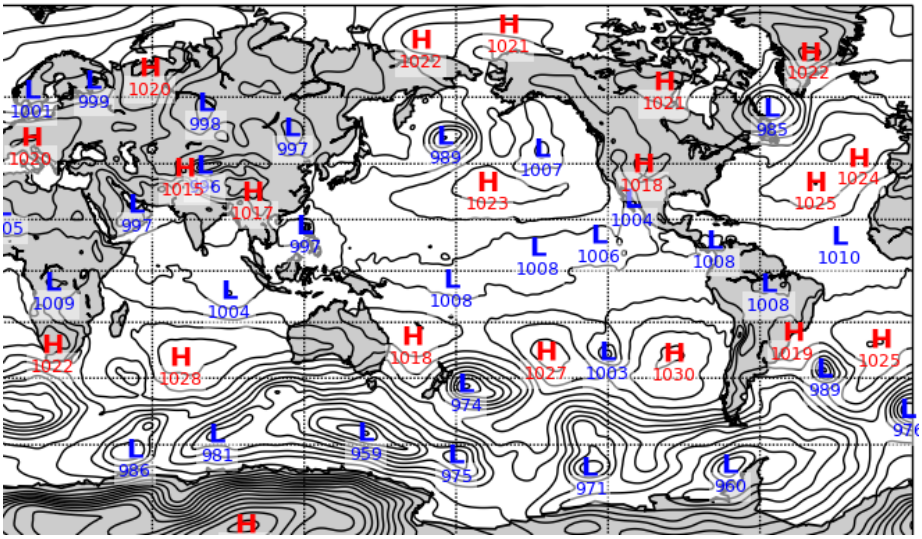

Ilustración 27 Muestra Capa Altas y Bajas Presiones

Para comprobar que las capas concuerdan con los datos y lo esperado, se utiliza la herramienta NetCDF Tools, que muestra los datos por colores y así podemos comprobar comparando colores con iconos si realmente se está haciendo lo que se espera.

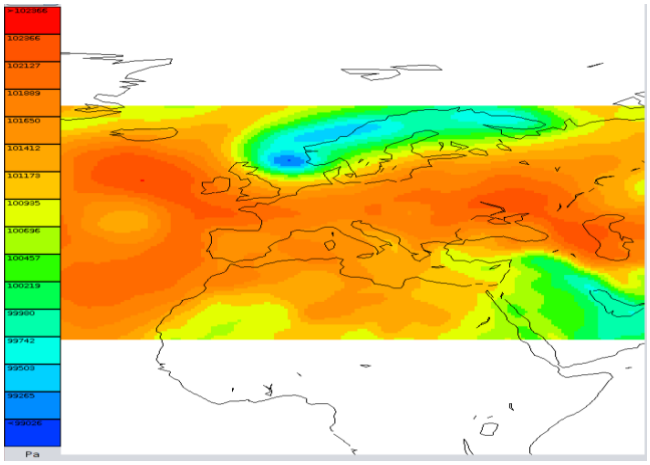

Ilustración 28 Visualización de Datos con NetCDF Tools

6. RESULTADOS

En este apartado se explican las pruebas que se han ido realizando para comprobar que el proyecto desarrollado funciona como debiera y cumple los requisitos.

6.1. PRUEBAS UNITARIAS

Este tipo de pruebas se realiza para comprobar el funcionamiento aislado de cada uno de los módulos del sistema, asegurándose que cada módulo funcione correctamente.

En este proyecto se ha utilizado este tipo de pruebas para comprobar en la parte del servidor que la creación del fichero JSON para la animación del viento funcionaba correctamente. Para ello se realizaron pruebas con JUnit comprobando que al ejecutar el método *readBandGrid* con ficheros válidos se crea un objeto *BandGridValue* con los datos correctos, mientras que al ejecutarlo con ficheros no válidos devuelve un error.

Además se comprobó en la parte de la creación de capas con Python que la capa que se crea es correcta comparando con los resultados que muestra la herramienta NetCDF Tools.

6.2. PRUEBAS DE INTEGRACIÓN

En las pruebas de integración se comprueba que cada una de los módulos funciona correctamente en conjunto con el resto de módulos. En este proyecto como ya se mencionó con anterioridad se ha seguido una metodología iterativa. Por lo que en cada una de las iteraciones se comprueba que la funcionalidad recién añadida funciona correctamente junto con las funcionalidades anteriores, es decir, que las nuevas funcionalidades no interfieran con el funcionamiento de las anteriores y todo funcione correctamente según lo establecido en los requisitos y casos de uso.

Estas pruebas se han llevado a cabo desde el navegador, viendo visualmente sobre la interfaz que todo funcionaba correctamente.

6.3. PRUEBAS DE ACEPTACIÓN

En este tipo de pruebas el cliente comprueba que cada uno de los casos de uso funciona correctamente. Como en este proyecto es todo muy visual, el cliente ha ido comprobando en cada iteración que el proyecto funcionaba correctamente. Cada uno de los requisitos y casos de uso han sido comprobados, así como las diferentes configuraciones de las interfaces.

7. CONCLUSIONES Y TRABAJOS FUTUROS

Para cerrar la memoria se realiza un balance del trabajo realizado, deteniéndose tanto en los resultados obtenidos como en los conocimientos adquiridos tanto técnicos como personales. También se hace una pequeña pausa en las posibles ampliaciones que se podrían realizar en el sistema.

7.1. CONCLUSIONES

Una vez finalizado el proyecto, nos hemos tomado un tiempo para valorar el trabajo realizado.

Los requisitos se han cumplido en su totalidad. Algunos de ellos han tenido que ser modificados ya que la lógica no permitía desarrollarlo como se pensó desde un principio, como en el caso del cambio de posición de las capas de predicción, que se pretendía poner en cualquier orden pero no ha sido posible. El resultado final ha sido que simplemente se pueda adelantar una capa.

Se ha utilizado una metodología iterativa incremental, lo que ha sido muy eficaz para este proyecto, ya que se pudo ir desarrollando poco a poco e ir comprobando que todo funcionaba tal como se esperaba y la interfaz de usuario quedaba atractiva visualmente.

Con el framework Spring Web que se basa en el patrón de diseño Modelo-Vista-Controlador también se ha acertado. Es un patrón modular que nos facilita enormemente el mantenimiento y la posibilidad de añadir nuevas funcionalidades sin tener que modificar todo el código. Además de la gestión de dependencias con Maven que hacen este trabajo aún más sencillo.

Por todo lo anteriormente expuesto creo que el resultado final del proyecto es el apropiado, con una interfaz totalmente configurable por el cliente, muy intuitiva y fácil de usar para los usuarios. Se pueden ver las predicciones tanto en modo gráfico como en modo numérico, así como ver la reproducción automática de las predicciones.

En cuanto a los conocimientos adquiridos, en el plano técnico, he aprendido muchas cosas, además de poner en práctica muchos de los conocimientos adquiridos durante la ingeniería, he utilizado diferentes lenguajes de programación, así como muchas de las librerías que proporcionan los lenguajes utilizados.

Además esta estancia en *Predictia* ha contribuido a la formación de lo que es realmente la vida laboral, no teniendo un profesor detrás, aprendiendo a solucionar los problemas por uno mismo, buscando la información necesaria en internet y no teniendo miedo a los nuevos retos.

7.2. TRABAJOS FUTUROS

Las aplicaciones software de este tipo, siempre dan pie a poder añadir nuevas funcionalidades o mejorar las que ya tiene.

En este apartado mencionamos algunas de los posibles trabajos futuros:

- Aunque la aplicación funciona si la visualizamos desde el navegador de un dispositivo móvil, la interfaz no queda muy bien visualmente. Por lo que mejorar la interfaz de modo que sea compatible con cualquier dispositivo puede ser un trabajo futuro interesante.

- Un selector de zona, es decir, que el usuario pueda seleccionar el país, provincia, ciudad, etc. del que desea ver la predicción. Por lo general a la gente le interesan las predicciones de su ciudad, y estaría bien poder seleccionar la ciudad y que el mapa muestre únicamente las predicciones para ésta.

- Añadir vista desde cámaras en vivo. Aunque se desconoce el coste de este servicio, parece interesante por ejemplo poner imágenes en directo de los principales puertos de montaña y de la costa de algunos de los lugares más relevantes.

REFERENCIAS

[1] Web de la Agencia Estatal de Meteorología *http://www.aemet.es/es/portada*

[2] Web de la Empresa Predictia *http://www.predictia.es/*

[3] Wikipedia WMS *http://www.opengeospatial.org/standards/wms*

[4] Web Mapnik *http://mapnik.org/*

[5] Web NetCDF *http://www.unidata.ucar.edu/software/netcdf/*

[6] Web Java *https://www.java.com/es/*

[7] Web Python *http://www.es.python.org/*

[8] Tutorial JavaScript *http://www.w3schools.com/js/*

[9] Tutorial CSS *http://www.w3schools.com/css/*

[10] Tutorial HTML *http://www.w3schools.com/html/*

[11] Web Leaflet *http://leafletjs.com/*

[12] Web D3.js *http://d3js.org/*

[13] Web Matplotlib/Basemap *http://matplotlib.org/basemap/*

[14] Web JUnit *http://junit.org/*

[15] Web Spring *https://spring.io/*

[16] Web Maven *https://maven.apache.org/*

[17] Web SublimeText *http://www.sublimetext.com/*

[18] Web Eclipse *https://eclipse.org/*

[19] Web Subversion *https://subversion.apache.org/*

[20] Web MagicDraw *http://www.nomagic.com/products/magicdraw.html*

[21] Freeman, E. y Robson, E. (2004). *Head First Design Patterns.* O'Reilly.

[22] Web de consulta StackOverflow *http://stackoverflow.com/*

[23] Web de consulta AskUbuntu *http://askubuntu.com/*

[24] Post Formato Vectorial y Raster *http://sig.cea.es/tipos_SIG*

[25] Documentación Librería gdal *http://www.gdal.org/*

[26] Sommerville, I. (2005). *Ingeniería del Software.* ADDISON-WESLEY.